Decentralized Link Analysis in Peer-to-Peer Web Search Networks

Dissertation
zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
der Naturwissenschaftlich-Technischen Fakultät I
der Universität des Saarlandes

Josiane Xavier Parreira

Max-Planck-Institut für Informatik

Saarbrücken
2009

Dekan der Naturwissenschaftlich-Technischen Fakultät I	Prof. Dr. Joachim Weickert
Vorsitzender der Prüfungskommission	Prof. Dr. Jens Dittrich
Berichterstatter	Prof. Dr.-Ing. Gerhard Weikum
Berichterstatter	Prof. Dr. Peter Triantafillou
Berichterstatter	Priv.-Doz. Dr.-Ing. Hannah Bast
Beisitzer	Dr. Martin Theobald
Tag des Promotionskolloquiums	22.07.2009

Acknowledgment

First and foremost, I would like to thank my advisor Prof. Dr.-Ing. Gerhard Weikum for his great guidance and the many interesting and helpful discussions. I would also like the thank the entire "Database and Information Retrieval" group at the Max-Planck Institute for Informatics, for creating such a great work environment. To my family and friends in Vitória, Brazil, that supported me even through the distance, I am very thankful. Many thanks go to my friend, co-author and academic sibling Sebastian. It was great to work with him in so many interesting problems. Thanks to Rali, Hans, Esteban and Dana for being such good friends and a special thanks to Cris, for always being there for me. I would also like to acknowledge the fellowships that I received from the International Max-Planck Research School and from the Dynamically Evolving Large-scale Information Systems European Project during my PhD studies.

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form in einem Verfahren zur Erlangung eines akademischen Grades vorgelegt.

Saarbrücken, den 22.07.2009

(Unterschrift)

Kurzfassung

Die Berechnung von Autoritäts- oder Reputationswerten für Knoten eines Graphen, welcher verschiedene Entitäten verknüpft, ist von großem Interesse in Web-Anwendungen, z.B. in der Analyse von Hyperlinkgraphen, Web 2.0 Portalen, sozialen Netzen und anderen Anwendungen. Die Lösung des Problems besteht oftmals im Kern aus der Berechnung des dominanten Eigenvektors einer Matrix, die vom zugrunde liegenden Graphen abgeleitet wird. Obwohl diese Analysen in einer zentralisierten Art und Weise berechnet werden können, gibt es gute Gründe, diese Berechnungen auf mehrere Knoten eines Netzwerkes zu verteilen, insbesondere bezüglich Skalierbarkeit, Datenschutz und Zensur. In der Literatur finden sich einige Methoden, welche die Berechnung beschleunigen, indem der zugrunde liegende Graph in nicht überlappende Teilgraphen zerlegt wird. Diese Annahme ist in Peer-to-Peer-System allerdings nicht realistisch, da die einzelnen Peers ihre Graphen in einer nicht synchronisierten Weise erzeugen, was inhärent zu starken oder weniger starken Überlappungen der Graphen führt. Darüber hinaus sind Peer-to-Peer-Systeme per Definition ein lose gekoppelter Zusammenschluss verschiedener Benutzer (Peers), verteilt im ganzen Internet, so dass Netzwerkcharakteristika, Netzwerkdynamik und mögliche Attacken krimineller Benutzer unbedingt berücksichtigt werden müssen.

In dieser Arbeit liefern wir die folgenden grundlegenden Beiträge. Wir präsentieren JXP, einen verteilten Algorithmus für die Berechnung von Autoritätsmaßen über Entitäten in einem Peer-to-Peer Netzwerk. Wir präsentieren Trust-JXP, eine Erweiterung von JXP, ausgestattet mit einem Modell zur Berechnung von Reputationswerten, die benutzt werden, um bösartig agierende Benutzer zu identifizieren. Wir betrachten, wie JXP robust gegen Veränderungen des Netzwerkes gemacht werden kann und wie die Anzahl der verschiedenen Entitäten im Netzwerk effizient geschätzt werden kann.

Darüber hinaus beschreiben wir in dieser Arbeit neuartige Ansätze, JXP in bestehende Peer-to-Peer-Netzwerke einzubinden. Wir präsentieren eine Methode, mit deren Hilfe Peers entscheiden können, welche Verbindungen zu anderen Peers von Nutzen sind und welche Verbindungen vermieden werden sollen. Diese Methode basiert auf verschiedenen Qualitätsindikatoren, und wir zeigen, wie Peer-to-Peer-Anwendungen, zum Beispiel JXP, von diesen zusätzlichen Relationen profitieren können.

Abstract

Analyzing the authority or reputation of entities that are connected by a graph structure and ranking these entities is an important issue that arises in the Web, in Web 2.0 communities, and in other applications. The problem is typically addressed by computing the dominant eigenvector of a matrix that is suitably derived from the underlying graph, or by performing a full spectral decomposition of the matrix. Although such analyses could be performed by a centralized server, there are good reasons that suggest running theses computations in a decentralized manner across many peers, like scalability, privacy, censorship, etc. There exist a number of approaches for speeding up the analysis by partitioning the graph into disjoint fragments. However, such methods are not suitable for a peer-to-peer network, where overlap among the fragments might occur. In addition, peer-to-peer approaches need to consider network characteristics, such as peers unaware of other peers' contents, susceptibility to malicious attacks, and network dynamics (so-called churn).

In this thesis we make the following major contributions. We present JXP, a decentralized algorithm for computing authority scores of entities distributed in a peer-to-peer (P2P) network that allows peers to have overlapping content and requires no a priori knowledge of other peers' content. We also show the benefits of JXP in the Minerva distributed Web search engine. We present an extension of JXP, coined *TrustJXP*, that contains a reputation model in order to deal with misbehaving peers. We present another extension of JXP, that handles dynamics on peer-to-peer networks, as well as an algorithm for estimating the current number of entities in the network.

This thesis also presents novel methods for embedding JXP in peer-to-peer networks and applications. We present an approach for creating links among peers, forming *semantic overlay networks*, where peers are free to decide which connections they create and which they want to avoid based on various usefulness estimators. We show how peer-to-peer applications, like the JXP algorithm, can greatly benefit from these additional semantic relations.

Zusammenfassung

Das Ordnen von Suchergebnissen (z.B. Webseiten, Benutzer oder Fotos) anhand verschiedener Qualitätsmerkmale ist ein wichtiger Bestandteil in Informationssystemen, wie sie zum Beispiel im Internet oder Web-2.0-Gemeinschaften auftreten. Eine besonders in den letzten Jahren stark beachtete Familie dieser Qualitätsmaße sind die so genannten Autoritäts- oder Reputationsmaße, deren Berechnung auf der Analyse des dominanten Eigenvektors des zugrunde liegenden Graphen beruht. Dieser Graph ist z.B. durch die Adjazenzmatrix von Webseiten oder durch Freundschaften in sozialen (Web-) Gemeinschaften definiert.

Obwohl diese Analysen von einem einzigen Server berechnet werden können, gibt es gute Gründe, sie auf mehrere Knoten (Peers) eines Netzwerks zu verteilen. Zum einen ist die Berechnung der Matrixzerlegung rechentechnisch sehr teuer und die Speicheranforderungen sind sehr hoch, da die Matrizen extrem groß sein können (obwohl oft sehr dünn besetzt). Zum anderen haben Benutzer oftmals Bedenken bezüglich Datenschutz und Anonymität, was eine Herausgabe der Daten zu einem zentralen Anbieter erschwert oder. unmöglich macht. Durch die speziellen Eigenschaften von Peer-to-Peer-Systemen können beide Punkte adressiert werden. Verteilte Berechnungen unter Ausnutzung der Ressourcen der beteiligten Peers (im Internet oder einem Data-Center) ermöglichen eine effiziente und skalierende Lösung. Zudem bleiben die Daten ausschließlich im lokalen System des Besitzers; nur ein kleiner Teil, der zur verteilten Berechnung benötigten Informationen wird bei Bedarf von Peer zu Peer übertragen, was nicht nur effizient ist, sonder auch bzgl. Datenschutz und Anonymität große Vorteile besitzt.

In der Literatur finden sich einige Methoden, welche die Berechnung beschleunigen, indem der zugrunde liegende Graph in nicht überlappende Teilgraphen zerlegt wird. Diese Annahme ist in Peer-to-Peer-System allerdings nicht realistisch, da die einzelnen Peers ihre Graphen in einer nicht synchronisierten Weise erzeugen, was inhärent zu starken oder weniger starken Überlappungen der Graphen führt. Darüber hinaus sind Peer-to-Peer-Systeme per Definition ein lose gekoppelter Zusammenschluss verschiedener Benutzer (Peers), verteilt im ganzen Internet, so dass Netzwerkcharakteristika, Netzwerkdynamik und möglichen Attacken krimineller Benutzer unbedingt berücksichtigt werden müssen.

In dieser Arbeit stellen wir JXP vor, einen Algorithmus zum Berechnen von Autoritätsmaßen über Entitäten, die in einem Peer-to-Peer-System verteilt

sind. Der Algorithmus macht keine Annahmen über die Verteilung der Daten oder über die Überlappung der Daten einzelner Peers. JXP kombiniert lokale Berechnungen mit Treffen zwischen Peers, bei denen bestimmte Zwischenergebnisse und Informationen über die Graphstruktur ausgetauscht werden. Diese Informationen, die ein Peer durch Treffen mit anderen Peers erhält, werden komprimiert gespeichert und beeinflussen nicht die Größe des lokalen Teilgraphen. Dies ermöglicht eine geringe Speicherbelegung sowie eine effiziente Berechnung zur Laufzeit der lokalen Algorithmen. Der bei den Treffen der Peers anfallende Datenverkehr ist ebenfalls vergleichsweise gering, da nur Teile der Graphstruktur und nicht die eigentlichen Daten ausgetauscht werden.

Theoretische und experimentelle Analysen zeigen, dass die durch JXP berechneten Autoritätswerte gegen die wahren Autoritätswerte (bei zentralisierter Berechnung auf allen Daten) konvergieren. Des Weiteren verdeutlichen die Analysen, dass bereits eine kleine Anzahl von Treffen zwischen Peers ausreicht, um eine gute Annäherung an die korrekten Autoritätswerte zu erreichen. Um den praktischen Nutzen von JXP zu verdeutlichen, haben wir JXP in Minerva integriert, einer auf dem Konzept der Peer-to-Peer-Netzwerke basierende verteilte Suchmaschine. Die von JXP berechneten Werte können in Minerva nicht nur zum üblichen Ordnen von Suchergebnissen benutzt werden, sondern darüber hinaus zum Selektieren von Peers, welche sich besonders für eine gegebene Anfrage anbieten. Neben dem eigentlichen JXP-Algorithmus präsentieren wir Erweiterungen, die die speziellen Eigenschaften von Peer-to-Peer-Netzwerken adressieren: Peers können unehrlich handeln und falsche Informationen austauschen; zudem kann das gesamte Netzwerk instabil sein, was dazu führt, dass Peers kommen und gehen oder ihre Inhalte ändern. Um das Problem der unehrlichen Peers zu lösen, haben wir TrustJXP entwickelt, eine Erweiterung von JXP mittels eines Modells zur Reputationsberechnung. Mit Hilfe von TrustJXP kann der Einfluss der unehrlichen Peers im System reduziert werden. Wir betrachten, wie JXP robust gegen Veränderungen des Netzwerkes gemacht werden kann und wie die Anzahl der verschiedenen Entitäten im Netzwerk effizient geschätzt werden kann.

Zusätzlich zu JXP haben wir uns im Rahmen dieser Arbeit mit der Konstruktion effizienter semantischer Netzwerke beschäftigt. Hier besteht das Problem darin, die im System vorhanden Peers bezüglich ihrer Inhalte zu gruppieren, um eine effiziente Anfrageverarbeitung zu ermöglichen. Das semantische Netzwerk besteht aus Verweisen zwischen Peers, die genutzt werden, um Anfragen entlang dieser Verweise an Peers weiter zu leiten, die sich am besten für eine gegebene Anfrage eignen, also gute Suchergebnisse liefern können. Solch eine Anordnung der Peers ist auch für JXP interessant, da durch die semantischen Verweise Peers mit ähnlichen Inhalten schneller gefunden werden können, was die Konvergenzgeschwindigkeit von JXP erhöhen kann. Wir präsentieren p2pDating, einen Ansatz zum Erzeugen dieser semantischen Netzwerke. p2pDating arbeitet wie JXP ebenfalls mit Treffen zwischen Peers, bei denen Informationen über den Inhalt der Datenkollektionen ausgetauscht werden. Diese Informationen bestehen z.B. aus Angaben zur Größe der Datenkollektion, aus der Verteilung (Verwendung)

des Vokabulars (zur Bestimmung der thematischen Ähnlichkeit) oder aus statistischen Beschreibungen der Dokumente, was zur Bestimmung der Überlappung benutzt werden kann.

Summary

Ranking entities (e.g., pages, users, photos, etc.) in social networks, Web graphs, and other relational structures is important in many applications such as Web search or Web 2.0 communities. A widely used family of measures to analyze authority, trust, or reputation consists of computing the principal eigenvector of a matrix derived from the underlying graph (e.g., a weighted adjacency matrix for Web pages or a weighted friendship/acquaintance matrix for the users of a social network).

Although such analyses could be performed by a centralized server, there are good reasons that suggest running theses computations in a decentralized manner across many peers. First, eigenvector computations are computationally expensive and require a large amount of memory, as the underlying matrices can be huge (despite their sparseness). Thus, harnessing the resources of a peer-to-peer network, on the Internet or within a data center, may offer a cost-efficient scalable solution. Second, users may care about privacy and autonomy and thus prefer a solution where they keep their parts of the data on their own computers, rather than completely delegating all data and analyses to central server. This consideration also leads to a decentralized peer-to-peer setting with data and computation spread across peers. While there exist a number of approaches for speeding up the analysis by partitioning the graph into disjoint fragments, these methods are not suitable for a peer-to-peer network, where overlap among the fragments might occur. In addition, peer-to-peer approaches need to consider network characteristics, such as peers unaware of other peers' contents, susceptibility to malicious attacks and peer dynamics.

We present JXP, an algorithm for computing authority scores of entities distributed in a peer-to-peer network. The algorithm assumes no predefined partitioning of the entities among peers, and overlaps among different peers' collections are allowed. Moreover, no a priori knowledge of other peers' content is required. JXP combines local computations with meetings among peers for exchanging knowledge about the link structure. This external knowledge is stored locally in a compressed way, which does not alter the size of the local subgraph. Therefore, storage costs remain low and local computations are fast throughout the execution of the algorithm. Costs of message exchange are also low, given that only the link structure, and not the content of entities, is needed. Theoretical and experimental analyses show that scores computed by JXP converge to the true authority scores that one would obtain by a centralized

computation, and that after an acceptable number of meetings the JXP scores are a good approximation of the correct values. We also show the benefits of JXP in the Minerva distributed Web search engine, where the JXP scores can be used for addressing the problems of query routing and ranking.

Besides the basic algorithm, we present extensions that address important properties inherent to peer-to-peer systems: peers may be dishonest and report false information, and the network is very dynamic, with peers constantly joining, leaving, and changing their contents. The former issue is addressed by *TrustJXP*, where JXP is combined with a reputation model in order to detect and amortize the influence of cheating peers on the scores computed. For coping with dynamics on peer-to-peer networks, we show that small modifications on what is locally stored enable peers to detect and react to changes in the network. We also present an algorithm for estimating the current number of entities in the network based on hash sketches and sliding windows.

In addition to the JXP framework, we also consider the problem of creating and maintaining semantic overlay networks, i.e., network organizations where peers are grouped according to their contents and/or interests. Semantic overlay networks are very useful for many typical peer-to-peer applications, for instance, query routing, where queries should be ideally sent only to peers that are able to provide meaningful results. They are also beneficial for our JXP algorithm, where ideally peers meet only those peers that are able to provide relevant information. We present an approach for creating semantic overlay networks, coined *p2pDating*, which also works via peer meetings, where each peer is free to decide which connections it creates and which it wants to avoid based on various usefulness estimators.

Contents

Chapter 1

Introduction

1.1 Motivation

Link analysis has been developed over the past 20 years in various fields including discrete mathematics (graph theory), social sciences (social network analysis) and computer science (graph as a data structure). Recently this area has attracted a wider attention for its applicability in Web search and Web 2.0 communities, where analyzing the authority or reputation of entities that are connected by a graph structure and ranking these entities is an important issue. Usually this issue is addressed by computing the dominant eigenvector of a matrix that is suitably derived from the underlying graph, or by performing a full spectral decomposition of the matrix. In the context of Web graphs, authority scoring, based on the Eigenspace analysis of a suitably defined graph of Web links, endorsements, or interactions, is an established tool for ranking information units (Web pages, sites, peers, social groups, etc.) by their relative importance [Cha02, BRRT05, LM06a]. As Google [1] has impressively demonstrated with its PageRank algorithm, such authority information can be exploited for improving the rank of search results. Social communities is another concept that has lately been explored to improve the search experience (e.g., del.icio.us, flickr.com). With billions of people from different parts of the world contributing with their input, the task of identifying the "hot spots" of a community becomes crucial. The community users interact in a way that results in community graphs that allow authority analyses similar to the PageRank-style analyses on Web graphs. Such community graphs naturally arise in various applications, by different means of user interaction, with respect to a wide variety of entities, and with varying notions of authority (e.g., product ratings, opinions on other people' blogs or photos, bibliographic references, etc.). Although such analyses could be performed by a centralized server, they are computationally expensive and require a large amount of memory, as the underlying matrices can be huge (despite their sparseness), which suggests running theses computations in a decentralized manner across many sites.

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

Peer-to-peer (P2P) technology has emerged as a compelling paradigm for large-scale file sharing, publish-subscribe, and collaborative work, as it provides great scalability and robustness to failures [SW05]. Thus, harnessing the resources of a peer-to-peer network, on the Internet or within a data center, may offer a cost-efficient scalable solution. Moreover, a lot of research has been dedicated to P2P Web search applications; spreading the functionality and data of a search engine across thousands or millions of peers. Such an architecture is being pursued in a number of research projects (e.g., [SMW+03, CAPMN03, BMT+05b, KNOT06, BMPC07, PRL+07]) and could offer various advantages: i) lighter load and smaller data volume per peer, and thus more computational resources per query and data unit, enabling more powerful linguistic or statistical learning methods; ii) with each peer being close to the human user and the user trusting its local software and controlling the degree of sharing personal information and collaboration with other peers, there is a great opportunity for leveraging user behavior such as explicit or implicit feedback in the form of query logs, click streams, or bookmarks; iii) a decentralized approach could provide better immunity to search result distortion by the bias of big providers, commercial interests, or even censorship. So this consideration also leads to a decentralized peer-to-peer setting with data and computation spread across peers.

While there exist a number of approaches for speeding up link analysis by distributing the link graph among multiple sites [KHMG03, WD04, AW03, BLMP06], these methods work only when the overall Web graph is partitioned into disjoint fragments, which is the case when partitions are formed by the sites that own the pages, and therefore are not suitable in the context of a peer-to-peer Web search engine. In addition, peer-to-peer approaches also need also to consider network characteristics, such as peers unaware of other peers' contents, susceptibility to malicious attacks, and network dynamics — so-called churn.

Another challenge in P2P Web search applications is query routing, i.e., how to efficiently select promising peers for a particular information need, given that the total number of relevant peers in a network is not known a priori and peer relevance also varies from peer to peer. In this context, Semantic Overlay Networks (SONs) [ACMHP04, BMR03, CGM04, TXKN03] appear as a network organization that improves query performance while maintaining a high degree of peer autonomy. Peers with semantically similar content are connected through an overlay network, and a peer can belong to multiple overlay networks (e.g., if its contents is diverse). Queries are routed only to the appropriate overlay networks, according to its semantics, increasing the chances that matching information (e.g. files, documents) will be found quickly, and reducing the load on peers having unrelated content. Determining which SONs a peer should join is a challenge itself. In most of early approaches, an algorithm that classifies the peers' contents into one or more predefined classes was used. Each of these classes define a SON. This leads to a fixed configuration of the SONs, so that the performance is highly dependable on a good choice of the classification algorithm and the classes, and it also requires that all peers use these same algorithm and

classes, which is undesirable.

1.2 Contributions

1.2.1 JXP — Decentralized Computation of Authority Scores

We address the problem of computing authority scores in a general P2P system with potentially overlapping graph fragments distributed across peers of a large network. We consider the architecture of a P2P search engine where each peer is autonomous, crawls Web fragments and indexes them locally according to the user's interest profile, and collaborates with other peers for query routing and execution. Queries would often be executed locally on the user's personalized "power search engine", and occasionally forwarded to other peers for better results. In such a setting, PageRank-style scores are still crucial for the ranking of search results, but the local Web fragment of a peer may be too small or incomplete for a meaningful link analysis.

JXP (Juxtaposed Approximate PageRank) is an algorithm for coping with the above situation: dynamically computing, in a decentralized P2P manner, global authority scores when the Web graph is spread across many autonomous peers with arbitrarily overlapping graph fragments and the peers are a priori unaware of other peers' fragments. In the JXP algorithm, each peer computes the authority scores of the pages that it has in its local index, by locally running the standard PageRank algorithm. A peer gradually increases its knowledge about the rest of the network by meeting with other, randomly chosen, peers and exchanging information, and then recomputing the PageRank scores of local interest. Theoretical and experimental analyses show that scores computed by JXP converge to the true PageRank scores that one would obtain by a centralized computation, and that after an acceptable number of meetings the JXP scores are a good approximation of the correct values. We also show the benefits of having the JXP scores in the Minerva distributed Web search engine, where the JXP scores can be used for addressing the problems of query routing and ranking.

1.2.2 TrustJXP — JXP Extension to Untrustful Networks

Since high authority scores can bring benefits for peers, it is expected that malicious peers would try to distort the results of the algorithm, by providing different (usually higher) scores for some of their local pages. P2P networks are generally vulnerable to malicious agents that can cheat in order to get more benefits. [MGM06] points out that P2P architectures for information sharing, search, and ranking must integrate a complete *reputation system*. Reputation systems operate by collecting information on the behavior of the peers, scoring each peer based on good vs. bad behavior, and allowing the system to take countermeasures against suspicious peers.

We present a trust model that integrates decentralized authority scoring with an equally decentralized reputation system. Our approach is based on anomaly detection techniques that allow us to detect a suspicious peer based on the deviation of its behavior from some common features that constitute the usual peer profile. Our method combines an analysis of the authority score distribution and a comparison of score rankings for a small set of pages. The JXP algorithm is then enhanced to avoid the impact of malicious peers. We call this enhanced version *TrustJXP*.

1.2.3 JXP under P2P Dynamics

In a P2P network peers are constantly joining and leaving the network, so that the full content is not always available. Moreover, peers might change what they store: for instance, a peer might become interested in a different topic and start to store information about this new topic. This has a big impact on the computation of authority scores, since links might as well change. We propose methods to adapt the JXP algorithm to work under dynamics. We also present a method for estimating the number of entities currently available in the network. The estimator combines multiple hash sketches [FM85] in a sliding window manner, allowing the estimator to deal with entities being removed from the network which is not directly supported by hash sketches.

1.2.4 p2pDating — Creation/Maintenance of SONs

To overcome the restriction that all peers have to use the same classification algorithm and predefined classes, we propose a new method for creating dynamically evolving Semantic Overlay Networks that gives more autonomy to the peers when deciding which SONs they should join. The method, coined *p2pDating*, works by rearranging the links on the overlay networks, according to the peers' criteria of a "good" neighbor or "friend", and using caching to remember the peers that were defined as friends. Possible measures for deciding if a peer should be considered a friend or not could be, for instance, the level of overlap between documents from the peer and documents from the candidate for being a friend, the similarity between their documents, the prior query history of the peers, level of trust, etc. A peer also has the option to delete an already established link with a friend, if it has either changed its selection criteria or found more interesting peers.

We show how peers acting autonomously can form context-rich SONs, and how the proposed SONs can be utilized during query routing in P2P web search engines, and also by our own JXP algorithm for devising a strategy for choosing peers for a meeting.

1.3 Publications

Various aspects of this thesis have been published as journal, workshop, and conference articles. The JXP algorithm was initially introduced in [PW05] and later improvements and extensions were published in [PDMW06, PCD+08]. In addition, a demonstration of the algorithm is presented in [PMB+07]. The TrustJXP algorithm is the topic of [PDCW07], whereas the JXP under dynamics was considered in [PMW08]. In the SONs context, the p2pDating algorithm is the topic of [PMW07]. Works that i have published in the context of social networks include [BCK+07, BCK+08, SCK+08a, CKM+08a, CKM+08b, SCK+08b].

1.4 Outline of this Thesis

The remainder of this thesis is organized as follows. Chapter 2 gives general background on matrix theory, peer-to-peer networks, and social networks. Chapter 3 presents an overview of existing work in the area of link analysis. Chapter 4 presents the JXP algorithm, its theoretical analysis, extensive experimental evaluation, and applications. Extensions of the JXP framework are presented in the subsequents chapters: TrustJXP is introduced in Chapter 5, and the methods for handling P2P dynamics are presented in Chapter 6. The p2pDating algorithm for creating and maintaining dynamically evolving Semantic Overlay Networks is presented in Chapter 7. Finally, Chapter 8 concludes this thesis and points out future research directions.

Chapter 2

Background

2.1 Markov Chains

A Markov chain is a stochastic process where, given the present state, future states are independent of the past states. In other words, the description of the present state fully captures all the information that could influence the future evolution of the process. This characteristic is known as the *Markov property.* A Markov chain can be described as follows. We have a finite set of *states,* $S = \{s_1, s_2, \ldots, s_r\}$ [1]. The process starts in one of these states and moves successively from one state to another. Each move is called a *step.* Given the values of random variables, $X_0, X_1, \ldots, X_n, \ldots$, denoting the states at time steps $0, 1, \ldots, n, \ldots$, respectively, a Markov chain satisfies the following property for all natural numbers n and states in S

$$Prob\{X_{n+1} = s_{n+1} | X_0 = s_0, X_1 = s_1, \ldots, X_n = s_n\} = Prob\{X_{n+1} = s_{n+1} | X_n = s_n\}.$$

If the chain is currently in state s_i, then it moves to state s_j at the next step with a probability denoted by p_{ij}, and this probability does not depend upon which states the chain was in before the current state.

The probabilities p_{ij} are called *transition probabilities.* The process can remain in the state it is in, and this occurs with probability p_{ii}. An initial probability distribution, defined on S, specifies the starting state. This may be done by specifying a particular state as the starting state or by assuming a uniform probability distribution for the starting state.

Applications of Markov chains can be found extensively throughout biological, physical, and social sciences, as well as business and engineering. For instance, consider the canonical example of a Markov chain: the weather in the Land of Oz [KST74]. The Land of Oz has many nice things, but not good weather. They never have two nice days in a row. If they have a nice day, they are just as likely to have snow as rain the next day. If they have snow or rain,

[1] The state space may be discrete or continuous (real-valued). In this work we consider only the case where the state space is discrete and finite.

they have an even chance of having the same the next day. If there is change from snow or rain, only half of the time is this a change to a nice day. With this information we form a Markov chain as follows. We take as states the kinds of weather R, N, and S. From the above information we determine the transition probabilities, that can be represented in matrix form as

$$\mathbf{P} = \begin{array}{c} \\ \mathrm{R} \\ \mathrm{N} \\ \mathrm{S} \end{array} \begin{array}{c} \begin{array}{ccc} \mathrm{R} & \mathrm{N} & \mathrm{S} \end{array} \\ \begin{pmatrix} 1/2 & 1/4 & 1/4 \\ 1/2 & 0 & 1/2 \\ 1/4 & 1/4 & 1/2 \end{pmatrix} \end{array}.$$

The matrix $\mathbf{P}$ of the example above, is called the *transition matrix* or *stochastic matrix*. The single-step transition matrix can be generalized to an n-step transition matrix whose elements are $p_{ij}^{(n)} = Prob\{X_{m+n} = j | X_m = i\}$. These elements can be obtained from the single-step transition probabilities by the following recursive formula:

$$p_{ij}^{(n)} = \sum_k p_{ik}^{(l)} p_{kj}^{(n-l)}, \text{ for } 0 < l < n.$$

This is called the *Chapman-Kolmogorov* equation. The proof can be found in [Ste94]. In matrix notation, the equations are written as

$$P^{(n)} = P^{(l)} P^{(n-l)}.$$

Note that

$$P^{(n)} = PP^{(n-1)} = P^n,$$

i.e., the matrix $\mathbf{P}^n$ gives the probability that the Markov chain, starting in state s_i, will be in state s_j after n steps.

2.1.1 Probability Distributions

In Markov chain analysis we are often interested in determining the probability that the chain is in a given state at a particular time step. The probability that the chain is state i at step n is denoted by $\pi_i(n)$, i.e.,

$$\pi_i^{(n)} = Prob\{X_n = i\}.$$

A row vector containing the probability distribution on the set of states is called a *probability vector*. Given the initial state distribution and the transition matrix, the state probabilities at any step can be obtained by

$$\pi_i^{(n)} = \sum_k p_{ki}^{(n)} \pi_k^{(0)},$$

which in matrix notation becomes

$$\pi^{(n)} = \pi^{(0)} P^n,$$

where $\boldsymbol{\pi}^{(0)}$ is the initial state probability vector.

If for some probability vector the following holds

$$\mathbf{v} = \mathbf{vP},$$

then we say that $\mathbf{v}$ is a *stationary distribution*. Given the initial state probability $\boldsymbol{\pi}^{(0)}$, if the limit

$$\lim_{n\to\infty} \boldsymbol{\pi}^{(n)}$$

exists, then this limit is called *limiting distribution*, and we write

$$\boldsymbol{\pi} = \lim_{n\to\infty} \boldsymbol{\pi}^{(n)}.$$

2.1.2 Steady-State Distributions of Ergodic Markov Chains

Ergodic Markov chains are defined as chains where all states are positive-recurrent and aperiodic. Positive-recurrent means that the average number of steps needed to return to a state for the the first time after leaving it is a finite number; aperiodic means that if we count all possible number of steps for which returning to the same state is possible, the greatest common divisor of these numbers is one.

For ergodic Markov chains the limiting distribution is guaranteed to exist and it is independent of the initial probability distribution. The limiting probabilities of an ergodic chain are often referred to as *equilibrium* or *steady-state* probabilities, in the sense that the initial state distribution $\boldsymbol{\pi}^{(0)}$ has disappeared. The equilibrium probabilities can be uniquely obtained by solving the matrix equation

$$\boldsymbol{\pi} = \boldsymbol{\pi}\mathbf{P}, \text{ with } \boldsymbol{\pi} > 0 \text{ and } ||\boldsymbol{\pi}||_1 = 1.$$

The transition matrix of an ergodic chain also has an interesting property: as $n \to \infty$, the powers $\mathbf{P}^n$ approach a limiting matrix $\mathbf{W}$ with all rows being the same vector equal to the steady-state probability vector $\boldsymbol{\pi}$.

Considering again the Land of Oz example, it can be shown that successive powers of the transition matrix $\mathbf{P}$ is

$$\mathbf{P}^{\infty} = \begin{array}{c} \\ \text{R} \\ \text{N} \\ \text{S} \end{array} \begin{array}{c} \begin{array}{ccc} \text{R} & \text{N} & \text{S} \end{array} \\ \begin{pmatrix} .4 & .2 & .4 \\ .4 & .2 & .4 \\ .4 & .2 & .4 \end{pmatrix} \end{array},$$

and we have $\boldsymbol{\pi} = (.4, .2, .4)$. In matrix theory, the equilibrium distribution corresponds to the left eigenvector associated with the dominant eigenvalue of matrix $\mathbf{P}$, which, in turn, is always equal to one.

2.1.3 Power Iteration Method

The limiting probabilities can be computed by different methods. One of the most well-known methods is the *power iteration* method, an iterative algorithm designed to compute the dominant eigenpair $(\lambda_1, \mathbf{x})$ of a matrix. Given the matrix $\mathbf{P}$ the power iteration algorithm starts with a random vector $\mathbf{v}_0$, and consecutively computes the following iteration

$$\mathbf{v}_{k+1} = \frac{\mathbf{v}_k \mathbf{P}}{||\mathbf{v}_k \mathbf{P}||_1}.$$

It can be proved that the vector $\mathbf{v}_k$ converges to the eigenvector associated with the dominant eigenvalue, and that μ_k, defined as

$$\mu_k = \frac{\mathbf{v}_k \mathbf{P} \mathbf{v}_k^T}{\mathbf{v}_k \mathbf{v}_k^T},$$

converges to the dominant eigenvalue. Recall that in case of transition matrices associated with ergodic Markov chains, the dominant eigenvalue is always one and since the probabilities are normalized, the computation can be simplified, leading to

$$\pi_{k+1} = \pi_k \mathbf{P}.$$

In practice, the iteration is repeated until we do not observe major differences between the vectors. One advantage of the power iteration method is that it does not perform a matrix decomposition, and hence it can be used when $\mathbf{P}$ is a very large sparse matrix.

2.1.4 Stochastic Complementation

Despite having methods to compute the stationary distribution without performing a matrix decomposition, there are cases where the computation is still very expensive, for instance when the number of states in the chain is too large.

For ergodic chains with a large number of states, one approach is to decompose the chain into several smaller sub-chains. Each smaller chain would have its own stationary distribution that would ideally be independent of the states of the other sub-chains; therefore, computing these distributions should be faster, since each sub-chain has fewer states, and can be performed in parallel. Then the stationary distribution of the original chain could be obtained by coupling back together the smaller distributions. This can be achieved under certain conditions by applying the concept of *stochastic complementation* [Mey89, Ste94].

Given an ergodic Markov chain, the state space is first partitioned into k subsets. The transition matrix $\mathbf{P}$ associated with this chain can be represented as

$$\mathbf{P}_{N\times N} = \begin{pmatrix} \mathbf{P}_{11} & \mathbf{P}_{12} & \dots & \mathbf{P}_{1k} \\ \mathbf{P}_{21} & \mathbf{P}_{22} & \dots & \mathbf{P}_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{P}_{k1} & \mathbf{P}_{k2} & \dots & \mathbf{P}_{kk} \end{pmatrix},$$

where all diagonal blocks are square. Now, for a given index i, we define $\mathbf{P}_i$ as the principal block sub-matrix of $\mathbf{P}$ obtained by deleting the i^{th} row and i^{th} column of blocks from $\mathbf{P}$, and $\mathbf{P}_{i*}$ and $\mathbf{P}_{*i}$ as follows

$$\mathbf{P}_{i*} = \begin{pmatrix} \mathbf{P}_{i1} & \dots & \mathbf{P}_{i,i-1} & \mathbf{P}_{i,i+1} & \dots & \mathbf{P}_{ik} \end{pmatrix},$$

and

$$\mathbf{P}_{*i} = \begin{pmatrix} \mathbf{P}_{1i} \\ \vdots \\ \mathbf{P}_{i-1,i} \\ \mathbf{P}_{i+1,i} \\ \vdots \\ \mathbf{P}_{ki} \end{pmatrix},$$

i.e., $\mathbf{P}_{i*}$ is the i^{th} row of blocks with $\mathbf{P}_{ii}$ removed, and $\mathbf{P}_{*i}$ is the i^{th} column of blocks with $\mathbf{P}_{ii}$ removed. The stochastic complement of $\mathbf{P}_{ii}$ in $\mathbf{P}$ is then defined as the matrix

$$\mathbf{S}_{ii} = \mathbf{P}_{ii} + \mathbf{P}_{i*}(\mathbf{I} - \mathbf{P}_i)^{-1}\mathbf{P}_{*i},$$

where $\mathbf{I}$ is the identity matrix. It can be proved (see [Mey89]) that all stochastic complements are stochastic matrices, therefore, for every $\mathbf{S}_{ii}$, there is a vector $\mathbf{s}_i$ such that

$$\mathbf{s}_i = \mathbf{s}_i\mathbf{S}_{ii},$$

in other words, $\mathbf{s}_i$ is the stationary distribution vector of $\mathbf{S}_{ii}$.

Coming back to the transition matrix $\mathbf{P}$, let $\boldsymbol{\pi}$ be the stationary distribution vector, also partitioned according to the same k subsets,

$$\boldsymbol{\pi} = (\boldsymbol{\pi}^{(1)}\ \boldsymbol{\pi}^{(2)}\ \dots\ \boldsymbol{\pi}^{(k)}).$$

For each $\boldsymbol{\pi}^{(i)}$ the following holds:

$$\mathbf{s}_i = \frac{\boldsymbol{\pi}^{(i)}}{\boldsymbol{\pi}^{(i)}\mathbf{e}} \quad (\mathbf{e} \text{ is a column of ones}).$$

So far we have shown how the stationary distribution vector is related to the stochastic complements, but we are still not able to compute it, given the stationary distributions from the stochastic complements. For that we need to introduce another matrix, called the *coupling matrix*. The coupling matrix $\mathbf{C}$ is a $k \times k$ ergodic matrix whose entries are defined by

$$c_{ij} \equiv \mathbf{s}_i \mathbf{P}_{ij} \mathbf{e}.$$

Given the stationary distribution vector of $\mathbf{C}$, ξ,

$$\xi = (\xi_1 \ \xi_2 \ \ldots \ \xi_k),$$

its elements ξ_i are known as *coupling factors*, since they can be combined with the stationary distributions of the stochastic complements to produce π in the following way

$$\pi = (\xi_1 \mathbf{s}_1 \ \xi_2 \mathbf{s}_2 \ \ldots \ \xi_k \mathbf{s}_k).$$

2.1.5 Iterative Aggregation/Disaggregation Methods

Computing the stochastic complements of an ergodic matrix is still considerable expensive, so people have considered alternatives that provide approximations for the stationary distribution, rather than the exact values, but at a much lower cost. One well-known family of such algorithms is the *Iterative Aggregation/Disaggregation Methods* [Ste94], which usually starts with an initial approximation and tries to refine it by performing a light-weight computation. If the refinement is still unsatisfactory, a new iteration is performed.

One of these methods is particularly useful for the following case: given a Markov chain represented by its transition matrix $\mathbf{P}$, its stationary distribution, and an updated chain, represented by $\mathbf{P}'$, how to compute that stationary distribution, or an approximation of it, of $\mathbf{P}'$ while keeping computation costs low. For this task we can explore the *lumpability* property of Markov chains: given a Markov chain, we can create a new process, called *reduced chain*, where a subset of the states is masked out, i.e., we observe the original chain only when it is in a state that does not belong to this subset. The lumpability property tells you that this reduced chain is also a Markov chain [KS63]. Let ϕ and π be the stationary distribution vectors of $\mathbf{P}$ and $\mathbf{P}'$, respectively. The vector ϕ can be combined with $\mathbf{P}'$ to build an aggregated Markov chain having a transition probability matrix $\mathbf{A}$ that is smaller in size than $\mathbf{P}'$. The stationary distribution $\mathbf{a}$ of $\mathbf{A}$ is then used to generate an estimate of the true distribution π.

First, the state space S, $S = S_1, S_2 \ldots S_N$, of the Markov chain is partitioned into $S = G \cup \overline{G}$, with $G = S_1, S2, \ldots, s_n$ and $\overline{G} = S_{n+1}, S_{n+2}, \ldots, S_N$.. The states in G are those "near" the updates, i.e., states that are likely to have been affected by the updates. The other subset $\bar{G}$ consists of all other states, i.e., states whose stationary probabilities are unlikely to have been affected (or have been affected in a negligible way). The transition matrix and its stationary distribution can then be represented as

$$\mathbf{P}'_{N \times N} = \begin{pmatrix} \mathbf{P}'_{11} & \mathbf{P}'_{12} \\ \mathbf{P}'_{21} & \mathbf{P}'_{22} \end{pmatrix}$$

$$\pi = \left(\begin{array}{ccc|ccc} \pi_1 & \ldots & \pi_n & \pi_{n+1} & \ldots & \pi_N \end{array} \right),$$

where the states were also partitioned according to G and $\overline{G}$, and n is the cardinality of G. The stationary probabilities from the previous distribution ϕ that correspond to the states in $\overline{G}$ are placed in a row vector $\overline{\phi}$, and the states in $\overline{G}$ are lumped into one superstate to create a smaller aggregated Markov chain whose transition matrix $\mathbf{A}$ is the $(n+1)\times(n+1)$ matrix given by

$$\mathbf{A} = \begin{pmatrix} \mathbf{P'}_{11} & \mathbf{P'}_{12}\mathbf{e} \\ \tilde{\mathbf{s}}\mathbf{P'}_{21} & 1-\tilde{\mathbf{s}}\mathbf{P'}_{22}\mathbf{e} \end{pmatrix},$$

where

$$\tilde{\mathbf{s}} = \frac{\overline{\phi}}{\overline{\phi}\mathbf{e}}.$$

The stationary distribution of $\mathbf{A}$ is given by $\mathbf{a}$, where

$$\mathbf{a} = (a_1,\ a_2,\ \ldots,\ a_n,\ a_{n+1}).$$

The stationary distribution of the updated chain, π, can be estimated by combining the stationary distribution of $\mathbf{A}$ with the previous stationary probabilities of the states in $\overline{G}$. The estimate, represented by $\tilde{\pi}$ is given by

$$\tilde{\pi} = (a_1,\ a_2,\ \ldots,\ a_n \mid \overline{\phi}).$$

It can be demonstrated [Mey89] that when there is absolutely no change in the stationary probabilities that correspond to states in $\overline{G}$, then

$$a_i = \begin{cases} \pi_i & \text{for } 1 \le \mathrm{i} \le \mathrm{n} \\ \overline{\pi}\mathbf{e} & \text{for } \mathrm{i} = \mathrm{n}+1 \end{cases}$$

i.e., the stationary distribution for the states in G obtained with the aggregated matrix $\mathbf{A}$ are equal to the stationary distribution of the same states in the updated transition matrix $\mathbf{P'}$.

2.2 Peer-to-peer Networks

2.2.1 Overview

In recent years, peer-to-peer (P2P) technology has become a compelling paradigm for large-scale file sharing, publish-subscribe, and collaborative work. While becoming popular mainly in the context of file sharing applications such as Napster, Gnutella, or BitTorrent, the P2P paradigm can be used to access any kind of distributed data and is rapidly making its way into distributed data management and offering possibilities for previously unseen Internet applications. P2P computing has enormous potential benefits regarding scalability, reliability, efficiency, flexibility, and resilience to failures and dynamics [SW05].

With the storage now distributed among many sites, an issue that arises is the *lookup problem*: where to store, and how to find a certain data item in a distributed system without any centralized control or coordination [BKK+03].

In contrast to traditional client-server systems, where the data is provided by dedicated physical entities that are explicitly referenced (e.g., by means of a Uniform Resource Locator (URL)), P2P systems store data in multiple, distant, transient, and potentially unreliable locations within the network. One of the predominant challenges of a P2P system, thus, is to efficiently locate data that is stored in the network. Its ability to do so even in the case of node failures and the resulting resilience in the presence of network dynamics constitute the potential benefits of a P2P system. P2P architectures can be classified according to how they address the lookup problem. The two main approaches are unstructured and structured architectures [SW05].

Unstructured P2P architectures do not rely on any central entity or any other form of explicit knowledge about the location of data within the network, when searching for a particular information. Instead, each node recursively forwards requests to all other peers that it is aware of (neighbors or a judiciously chosen subset), in an attempt to locate all relevant data in the network. In order to reach all appropriate peers, a node broadcasts each message it receives to other peers, regardless of whether they store relevant data or not. This approach is known as message flooding and effectively leads to a breadth-first search strategy. Each message is assigned a Time-to-live (TTL) value, which a peer decreases by one when forwarding a message, to avoid infinite loops and to control the number of messages being generated by one query being issued. An advantage of this approach is the fact that it is not necessary to pro-actively maintain the network, e.g., upon node joins and leaves. Also, there is no enforcement of the storage location for data items, as they can be located anywhere in the network. In other words, the data stored on a node is unrelated to the node's position in the network. However, message flooding has a high bandwidth consumption cost, and there is no guarantee that all the relevant data will be found. Popular implementations of this paradigm include Freenet [CMH+02] and early version of the Gnutella protocol [SW05]. Other examples of unstructured P2P networks are based on epidemic (or gossiping) protocols [VvS03], where information is randomly disseminated across the peers in order to keep the network connected.

Structured P2P architectures superimpose certain overlay structures to map nodes and data items into a common address space, enabling a unique mapping from data items to nodes given the current state of the network. For this purpose, each node manages a small number of pointers to carefully selected other peers (typically $O(logN)$, where N is the number of nodes in the network); routing along these paths eventually leads to the globally agreed-on peer that is currently responsible for a given data item, commonly with $O(logN)$ message hops. Distributing the responsibilities as uniformly as possible over the nodes in the network provides balanced storage and retrieval loads among all nodes. On top of this routing functionality, it is straightforward to implement what is known as *Distributed Hash Tables (DHTs)*: a hash-table-like data structure that allows the insertion and retrieval of (key, value)-pairs. For insertion or retrieval of a (key, value)-pair, turn to the peer currently responsible for the

key in the network as defined by the structured P2P network. This peer stores and maintains all (key, value)-pairs for the same key. Note that, in contrast to unstructured P2P architectures, the placement of data is no longer arbitrary, but determined by the overlay network. Examples are Chord [SMK+01], Pastry [RD01], and CAN [RFH+01].

2.2.2 Distributed Information Retrieval

An application that is gaining momentum is P2P Web search, where the functionality and data of a search engine is spread across peers [SMW+03, CAPMN03, BMT+05b, KNOT06, BMPC07, PRL+07]. It is important to point out that Web search is not simply keyword filtering, but involves relevance assessment and ranking search results. In this architecture each peer has a full-fledged search engine, with a focused crawler, an index manager, and a top-k query processor. Each peer can compile its data at its discretion, according to the user's personal interests. Queries can be executed locally on the small-to-medium-sized personalized corpus, but they can also be forwarded to other, appropriately selected, peers for additional or better search results. For this application, the P2P paradigm has a number of potential advantages over centralized search engines with very large server farms:

- The load per peer is much lower than the load per computer in a server farm, so that the P2P-based global computer could afford much richer data representations, e.g., utilizing natural-language processing, and statistical learning models, e.g., named entity recognition and relation learning.
- The local search engine of each peer is a natural way of personalizing search results, by learning from the user's explicit or implicit feedback given in the form of query logs, click streams, bookmarks, etc. In contrast, personalization in a centralized search engine would face the inherent problem of privacy by aggregating enormous amounts of sensitive personal data.
- The P2P network is the natural habitat for collaborative search, leveraging the behavior and recommendations of entire user communities in a social network. A key point is that each user has full and direct control over which aspects of her behavior are shared with others, which ones are anonymized, and which ones are kept private.

Query Routing

One of the key issues to make P2P Web search feasible is query routing: judiciously selecting a small subset of remote peers that are expected to be good sources of information for a specific query from an a-priori unlimited number of peers. A key goal from a performance viewpoint is to minimize the number of individual collections that have to be gathered in order to achieve good result quality (usually measured in terms of recall in this distributed setting). As such, research in P2P search enjoys a large overlap with research on distributed

information retrieval and can highly benefit from existing work. However, the peculiarities of a P2P architecture require a different view on some key aspects. For example, the absence of a centralized indexing facility together with the difficulties to calculate global metrics in this large and highly dynamic network hamper the use of traditional methods for collection selection. An number of projects have been tackling this problems with different approaches. Examples are CORI [CLC95], GlOSS [GGMT99], and Minerva [BMT+05a, MBTW06].

Result Merging

The second key issue is result merging: when the peers that have been selected during query routing return their top-ranked local results, these results have to be combined into a single, comprehensively ranked result list, which is eventually displayed to the user. As we are dealing with the local query execution results of a large number of autonomous peers, each peer individually has the freedom to deploy its favorite document scoring model, rendering scores mutually incompatible and incomparable. Even if they had agreed on a common document scoring model, most of them rely on (global) statistical knowledge that is not readily available for a large-scale distributed system. The obvious solution, the usage of local statistics, again leads to scores that are inherently incomparable across peer boundaries. There are many different approaches to address this problem: some work with having all peers agree in a common scoring function that can be computed locally [CCH92], while others opt for recomputing the scores of documents once the all peers have return their results. Another approach tries to overcome the problem of lack of global knowledge by having the peers collaborate to compute estimates of those global values, like global document frequency [BMTW06]. Each peer would then be able to produce document scores that are comparable to other peers scores.

2.2.3 Semantic Overlay Networks

Semantic Overlay Networks (SONs) [ACMHP04, BMR03, CGM04, TXKN03] are a network organization that improves query performance while maintaining a high degree of peer autonomy. Peers with semantically similar content are connected through an overlay network, and a peer can belong to multiple overlay networks (e.g., if its contents is diverse). Queries are routed only to the appropriate semantic overlay networks, according to its semantics, increasing the chances that matching information (e.g. files, documents) will be found quickly, and reducing the load on peers having unrelated content. There are many challenges when building SONs, regarding how peers are assigned to SONs and to which SONs a query should be sent. According to the initial idea, peers should be evenly distributed among SONs, so that we can answer queries fast, as we have to ask fewer peers; and each peer should belong to a small number of SONs, so that each peer has to handle only a small number of connections. However, in the real world, the distribution of peers over topics is expected to be very skewed and dynamic as many peers will have contents belonging to some very

popular topics that are constantly changing, whereas the number of peers which topics are less common will be low. Moreover, in most of early approaches, an algorithm that classifies the peers' contents into one or more predefined classes was used. Each of these classes defines a SON. This leads to a fixed configuration of the SONs, so that the performance is highly dependable on a good choice of the classification algorithm and the classes, and it also requires that all peers use these same algorithm and classes, which is undesirable.

2.2.4 Trust

In general P2P networks are vulnerable to malicious agents that can cheat in order to get more benefits. Attacks by anonymous malicious peers have been observed on today's popular peer-to-peer networks, the most common being inauthentic file attacks, wherein malicious peers respond to virtually any query providing "fake files" that are tampered with or do not work. The complete lack of accountability of the resources that peers share on the network offers an almost ideal environment for malicious peers and mandates the introduction of reputation systems that help to assess the quality and trustworthiness of peers. There are many issues related to the design of a decentralized reputation system, and a good overview can be found in [MGM06]. According to [KSGM03], there are five issues that are important to address in any P2P reputation system:

- The system should be self-policing, that is, the shared ethics are defined and enforced by the peers themselves and not by some central authority.
- The system should maintain anonymity of peers, i.e., a peer's reputation should be associated with an opaque identifier (such as nickname) rather than with an externally associated identity (such as a peer's IP address).
- The system should not assign any profit to newcomers as that would encourage malicious peers with poor reputations to continuously change their opaque identifiers to obtain newcomers status.
- The system should have minimal overhead in terms of computation, infrastructure, storage, and message complexity.
- The system should be robust to malicious collectives of peers who know one another and attempt to collectively subvert the system.

Examples of reputation methods are the ones that work by peers collaborating to assign trust scores to other peers based on past interactions (e.g. EigentTrust [KSGM03] and the other presented in [XL04a]), and those that analyze peer activity on the network in order to identify peers whose behavior deviates from the typical peer-traffic profile (e.g. SeAl [NT04] and other work in [SBWS05]).

2.2.5 Dynamics

P2P networks have a dynamic nature: peers are autonomous and free to decide when to join and when to leave the network [SW05]. The effect of these independents arrivals and departures is known as *churn*. Moreover, peers might change what they store; for instance, a peer can become interested in a different topic and start to store information about this new topic instead. Dynamics plays a big role in any P2P application, since the content that is available in the networks varies over time. The impact can be smoothed by applying techniques for replicating data [PNT06, DR01], where multiple copies of the same item are stored at different peers, therefore increasing the changes that this item is available at some particular time. But with or without replication, peers still need to know how to locate data; so the references to other peers need to be constantly updated. In structured P2P networks there are protocols for peers joining or leaving the network, which include notifying other peers and reconstructing the peer connections [RGRK04]. However peers might unexpectedly leave the network, for example, due to a failure, and inconsistencies in the network might appear. Therefore, some systems also periodically check the availably of peers, which can be done efficiently by piggybacking messages into the already existing communication. In P2P web search applications that rely on documents statistics, there is the additional effort of keeping these statics updated, so the scores can be correctly computed.

2.3 Social Networks

2.3.1 Overview

The advent of online social community platforms (e.g., Flickr, del.icio.us, MySpace, Facebook, or YouTube) has changed the way users interact with the Internet. While previously most users were mere information consumers, those platforms are offering an easy and hassle-free way for typical users to also publish their own content, making the users also information producers. On these social platforms, users are encouraged to share photos, videos, opinions, to rate content, but also to explore the online community and to find people with similar interest profiles. In this sense, online social community platforms not only change the way people interact with the Internet, but also the way users interact with each other. While differing in the type of content that they focus on (e.g., blog entries, photos, videos, bookmarks), almost all online social community platforms work similarly. Initially, users must register in order to join the community. Once registered, they start to produce information, ideally by publishing their own documents and by adding tags (or ratings, comments, etc) to other content already available in the community. The platforms also offer a way to maintain a list of friends and means to keep friends informed about your latest content items. The size of your friend network is often considered as your reputation in the network; making new "friends" often seems at least as important as publishing new content. While initially many users populate the

list of friends with people they already know from the offline world (e.g., family members and school mates) or other online communities, as time goes by they typically identify previously unknown users that they share common interests with and also add those users to the friends list. One particularly interesting feature of these communities is the widely-used opportunity to attach manually generated annotations, so-called tags, to content items [HJSS06, BMCMA09]. In this context, tags can be considered precise descriptions of content items, flavored with the respective personal interest of the user who generated the tag. Most online communities offer comfortable and intuitive ways to explore new content items based on these tags, e.g., via tag clouds [DKM+06, HRS07]. Thus, tagging has emerged as an important asset to explore the fast growing communities in order to identify interesting content and users.

2.3.2 Scoring Models and Query Processing

The typically high quality of user-generated tags suggests to leverage this "wisdom of the crowds" for effective methods to identify and rank high-quality and high-authority content in the communities, but at the same time, the fast-growing amount of data calls for particularly efficient (i.e., fast and scalable) methods to fulfill this task. The existing, traditional algorithms for searching on the Web fall short of being effective in social networks, as they disregard the social component and focus on the content quality only. This makes a strong case for novel methods that exploit the additional features of social networks, i.e., the presence of different entities (users, documents, tags) and their mutual relationships.

Recently, a lot of research has been devoted to developing various forms of community-aware ranking methods that includes identifying important entities inside communities [HJSS06, ZAA07, BXW+07], measuring similarity among tags [XBCY07], and integrating user-user relationships into the scoring function [SCK+08a]. Aspects of user communities have also been considered for peer-to-peer search, most notably, for establishing "social ties" between peers and routing queries based on corresponding similarity measures, (e.g., similarities of queries issued by different peers) [BCK+07, PGW+08, MGD06, DNP05].

Chapter 3

State of the Art in Link Analysis

3.1 Link Analysis

Links can be found everywhere, connecting all sorts of entities. For instance, people are connected to those they know, scientific papers are connected through citations among them, cities are connected through transportation routes, etc. In most of the cases we can identify graph structures, where the entities are the nodes in the graph and the links are the graph edges. The analysis of such link structures has been proven very useful, for instance, to predict where new links will be formed [BA99]. Recently this area has attracted a wider attention for its applicability in Web search and Web 2.0 communities, where analyzing the authority or reputation of entities that are connected by a graph structure and ranking these entities is an important issue. We proceed by giving examples of the types of graphs that appear in the context of Web and Web 2.0. Then we talk about two well known methods for link analysis and some of their variations in both centralized and decentralized settings.

3.2 The Web and Other Types of Graph

The Web is a system of interlinked documents accessed via the Internet. Web pages may contain text, images, videos, and other multimedia items and are connected to other Web pages through hyperlinks. As an example, consider the excerpt of the Web document depicted in Figure 3.1. We can see that the document makes references to other Web documents by placing hyperlinks to those documents in its text. The Web documents and the hyperlinks can be modeled as a graph, where the documents corresponds to the nodes in the graph and the hyperlinks to the edges, as we can also see in Figure 3.1.

With the advent of Web 2.0, the process of creating and sharing documents over the Internet became a lot easier. In social communities, the content shared

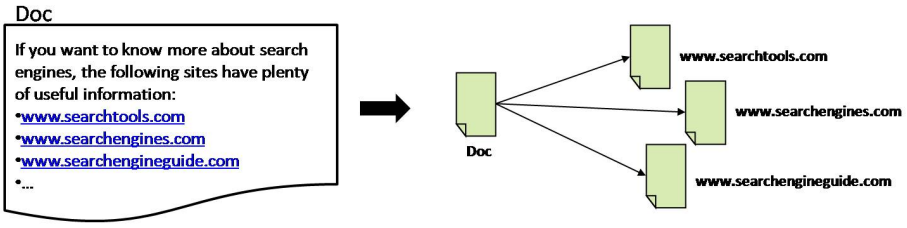


Figure 3.1: Web documents and hyperlinks modeled as a graph.

is not restricted to Web pages, and it might as well be a pictures, videos, references to book, etc. These online communities also allow more interaction among users and the entities shared, through the concepts of *online friendships*, i.e., connecting to other users who share common interests, and *tagging*, i.e., annotating a resource with keywords that describe its content. The relationship among users, tags and documents can also be modeled as a graphs. A example of a social graph is shown in Figure 3.2. Other forms of social graph are also found in literature (e.g., the social content graph suggested in [AYBB07]).

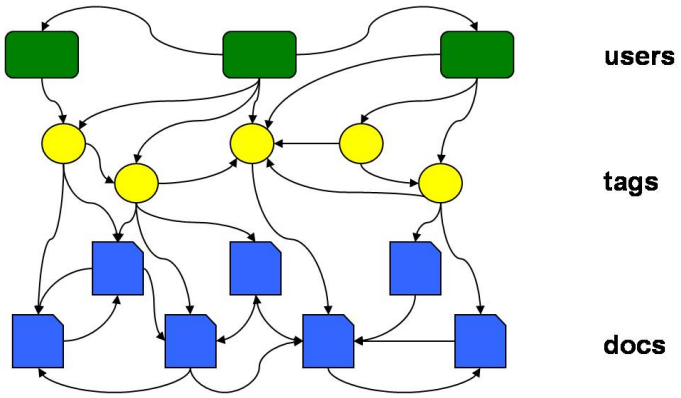


Figure 3.2: Interactions in social communities modeled as graph.

In both scenarios, there is a strong need to identify important entities for a particular need. For instance, given a query, we want to identify, out of the (usually) thousands of documents that match the query terms, the small set of the most "authoritative" ones, or in a network of users, find the most influential users. The former is a key element in search result ranking, the latter is important for marketing strategies, in order to reach as many people as possible by contacting only a few. Link-based authority ranking is based on treating links as endorsements. An entity p, by placing a link to entity q, is in some way conferring authority to q. Link-based authority ranking has received great attention in the literature. It has started with the seminal works of Brin and Page [BP98] and Kleinberg [Kle98], and after these, many other models

and techniques have followed. Good surveys of the many improvements and variations are given in [Cha02, LM03, BRRT05, Ber05].

3.3 The InDegree Algorithm

The first and the simplest of all link analysis algorithms uses page popularity (also known as page visibility [Mar97]) as a ranking factor. The popularity of a page is measured by the number of pages that link to this page. We refer to this algorithm as the InDegree algorithm, since it ranks pages according to their indegree in the graph. This heuristic is also used in the field of bibliometric analysis [Gar79], where the importance of a publication is measured by the number of citations it has received. The InDegree was applied by several search engines in the early days of Web search [Mar97]. However, later approaches have shown that the algorithm is not sophisticated enough to capture the authoritativeness of a node, and that the origins of the incoming links play an important role.

3.4 HITS

Hypertext-Induced Topic Search (or HITS), proposed by Kleinberg [Kle98], considers that pages can be *authorities*, if they contain good resources, or *hubs*, if they contain links to good authoritative pages. [1] Endorsement is conferred on authorities through hubs. In this framework, every page can be thought of as having two roles. The hub role captures the quality of the page as a pointer to useful resources, and the authority role captures the quality of the page as a resource itself. If we make two copies of each page, we can visualize the graph as a bipartite graph where hubs point to authorities (see Figure 3.3).

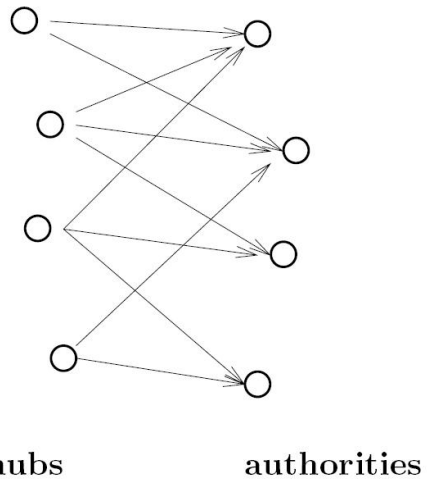


Figure 3.3: Hubs and authorities [Kle98].

[1] A page can be at the same time a good authority and a good hub.

There is a mutual reinforcing relationship between the hubs and authorities. A good hub is a page that points to good authorities, while a good authority is a page pointed to by good hubs. In order to quantify the quality of a page as a hub and an authority, it associates with every page a hub and an authority weight. Following the mutual reinforcing relationship between hubs and authorities, the hub weight is defined as the sum of the authority weights of the nodes that are pointed to by the hub, and the authority weight as the sum of the hub weights that point to this authority. Let **a** and **h** denote the vectors of the authority and hub weights, respectively, where a_i and h_i , are the authority and hub weight of node i. We have that

$$a_i \sim \sum_{j \in Pred(i)} h_j \quad \text{and} \quad h_i \sim \sum_{j \in Suc(i)} a_j,$$

where $Pred(i)$ and $Suc(i)$ are the sets of predecessors and successors of page i, respectively. In matrix notation we have,

$$\mathbf{a} = \alpha \mathbf{A}^T \mathbf{h} \quad \text{and} \quad \mathbf{h} = \beta \mathbf{A} \mathbf{a},$$

where A is the graph's adjacency matrix, and α and β are constants that appear due to normalization of the hub and authority vectors.

Given a user query, the algorithm first constructs a query specific graph as follows: a start set of pages matching the query, called the *root set*, is obtained from a search engine. The root set is then expanded by following the links that enter and leave it, up to a certain number, forming the so-called *base set*. The pages in the base set and the link among them form the query specific graph. The authority and hub scores are then computed in the query specific graph. Since the scores are computed in a graph that is formed according to the query, we say that they are *query dependent scores*. One of the drawbacks of the HITS algorithm is that its performance is highly dependent on the choice of the base set, and it is often the case where the base set contains pages not relevant to the query topic, which might lead to topic drift problem, i.e., the most highly ranked authorities and hubs might not be about the original topic.

3.5 PageRank

PageRank [BP98, PBMW98] is probably the most well-known algorithm for computing authority scores, since part of the success of Google's search engine is credited to it. Differently from HITS, PageRank does not distinguish between hubs and authorities: if page p has a link to page q then the author of p is implicitly endorsing q, i.e., giving some importance to page q. How much p contributes to the importance of q is proportional to the importance of p itself. A simplified example is given by Figure 3.4.

This recursive definition of importance is captured by the stationary distribution of a Markov chain that describes a random walk over the graph, where we start at an arbitrary page and in each step we choose a random outgoing

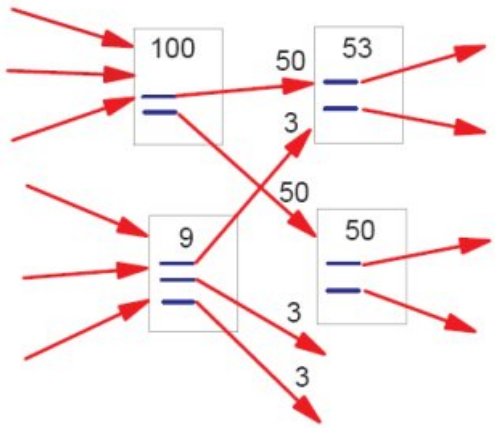


Figure 3.4: Simplified PageRank Calculation [PBMW98].

edge from the current page. To ensure the ergodicity of this Markov chain (i.e., the existence of stationary page-visit probabilities), additional random jumps to uniformly chosen target pages are allowed with small probability $(1-\varepsilon)$. Formally, the PageRank of a page q is defined as

$$PR(q) = \varepsilon \times \sum_{p|p \to q} \frac{PR(p)}{out(p)} + (1-\varepsilon) \times \frac{1}{N}.$$

where N is the total number of pages in the link graph, $PR(p)$ is the PageRank score of the page p, $out(p)$ is the outdegree of p, the sum ranges over all link predecessors of q, and $(1-\varepsilon)$ is the random jump probability, with $0 < \varepsilon < 1$ and usually set to a value like 0.85.

PageRank values are usually computed by initializing a PageRank vector with uniform values $1/N$, and then applying a power iteration method (see Section 2.1.3), with the previous iteration's values substituted in the right-hand side of the above equation for evaluating the left-hand side. This iteration step is repeated until sufficient convergence, i.e., until the PageRank scores of the high-authority pages of interest exhibit only minor changes. Note that by dividing the PageRank score of a page by its outdegree it means that the scores are normalized and convergence is guaranteed.

This computation considers the complete graph, regardless of the query, therefore we say that PageRank scores are *query independent*.

3.6 Incremental, Online, and Distributed Link Analysis

With increasing scale of the Web and its constant changes, (re-)computing PageRank scores has become a very expensive task. Lately a lot of research has

been dedicated to distributed alternatives to PageRank computing (including our own work). With the advent of P2P networks [Abe01, SMK+01, RFH+01, RD01] approaches that aim at distributing the computation over peers in such networks have also appeared. Roughly the methods proposed can be classified into three groups: the ones where the computation is sped up by partitioning the graph into smaller subgraphs, the ones that focus on incremental updates of the scores when only small parts of the graph have changed, and the approaches that focus on P2P systems.

3.6.1 Graph Partitioning

For speeding up the PageRank computation, Kamvar et al. [KHMG03] exploit the topology of the Web graph. In their work, they observed that the Web graph has a nested block structure, where the number of links among pages in the same host is much larger than the number of links among pages at different hosts, with the same reasoning applying to the domain level. They present the BlockRank algorithm for computing PageRank by taking into account the block structure of the Web graph. The BlockRank algorithm works by executing the following steps:

1. Split the graph into blocks according to the domain.
2. Compute the Local PageRank for each block;
3. Estimate the relative importance of each block ("BlockRank").
4. For each page, combine its Local PageRank score with the BlockRank score of the block it belongs to.
5. Use the combined scores from the previous step as the starting vector for the standard PageRank algorithm.

The Local PageRank scores are obtained by running the standard PageRank algorithm inside the block, where only links among pages inside the block are considered. The BlockRank is computed by running the standard PageRank algorithm in the so-called *block graph*, where each vertex in the graph corresponds to a block in the Web graph, and an edge between two pages in the Web is represented as an edge between the corresponding blocks, with proper weighting (see [KHMG03] for details). The scores for the starting vector of the Global PageRank are computed by multiplying, for each page, the Local PageRank score by the BlockRank score of the block the page belongs to. Local PageRank scores of each block, as well as BlockRank scores, are normalized, therefore the sum of the scores of the starting vector is also 1. Experiments have shown that the BlockRank algorithm can speed up the PageRank computation by a factor of 2.

Other works that also works by partitioning the graph opt for computing an approximation of the PageRank scores, instead of the exact values. Since most of the applications are only interested in the correct ranking order, regardless of

the absolute score values, approximated PageRank values are a much cheaper solution to the problem. The algorithm presented by Wang et al. [WD04] resembles the BlockRank algorithm, in the sense that there is also a Local PageRank computation at server level where only intra-server links are considered, combined with a ServerRank score for every server, which measures the importance of each server based on the inter-server links. Both algorithms, however, have different goals: while the BlockRank algorithm uses the Local PageRank scores together with the BlockRank scores as a starting point for the global PageRank computation, the algorithm from [WD04] combines Local PageRank scores and ServerRank scores to provide an approximation of the global PageRank score, without the need of the performing the standard PageRank algorithm on the complete graph. Moreover, the computation of the ServerRank scores is also done in a distributed manner, with servers exchanging messages among them, which makes the algorithm completely decentralized.

A similar approached is pursued in the work by Wu et al. [WA05], where a Layered Markov Model is used to distinguish transitions among Web sites and Web pages. A DocRank score is assigned to each page but considering only the links within the Web site the pages belongs to. Each Web site also receives a score, called SiteRank, according to the inter-site links. Scores at page and site level are combined to produce the final ranking.

Another work closely related to the aforementioned approaches is the one by Broder et al. [BLMP06], which presents a graph aggregation method in which pages are partitioned into hosts and the stationary distribution is computed in a two-step approach, combining the stationary distributions inside each host and the stationary distribution of a coarse-grained inter-host graph.

Following a different approach, but also without requiring the Web matrix to be stored in one place, Abiteboul et al. [APC03] propose the OPIC algorithm. OPIC stands for online page importance computation, and as the name says the authority of the pages are computed on the fly. It works by randomly (or otherwise fairly) visiting Web pages in a long-running crawl process and performing a small step of the PageRank power iteration method for the page and its successors upon each such visit. Two values, called *cash* and *history* are kept for each page. Initially every page gets some initial cash value. When a page is visited, its current cash is added to the history and distributed equally to all outgoing neighbors of the page. The cash value is then set to zero. The value store in the history, i.e., the sum of the cash obtained by the page since the start of the algorithm reflects the importance of the page. For dealing with changes in the graph, the authors propose a variant of the OPIC algorithm, called Adaptive OPIC, where the history keeps only the cash received during a particular time window.

3.6.2 Incremental Updates

Web and social graphs are constantly changing, mostly by the insertion of new nodes and links. Since a full recomputation would be very expensive, some

methods opt for performing an approximate incremental computation of the scores. The works by Chien et al. [CDK+03] and Langville et al. [LM06b] are based on the aggregation/disaggregation methods from Markov chains theory (see Section 2.1.5). Given a set of link changes in the Web Graph, they identify a small portion of the Web graph in the vicinity of the changes, and model the rest of the Web as a single node in this small graph, using the knowledge of the scores distributions from a previous computation. The PageRank scores computed on this small graph are then used to approximate the PageRank scores on the original graph. There is no exact way of determining the number of pages that should be left unaggregated; the higher the number of pages in the small graph, the better the approximation but the computation becomes more expensive. Both works suggest an heuristic that considers pages that are more likely to be affected by the changes in the Web graph, by observing how the PageRank masses from nodes directly affected by the changes dissipate over through the graph.

Instead of trying to approximate the scores of all pages in the graph, the work by Chen et al. [CGS04] focus on approximate only the PageRank scores of a subset of interest. It also starts by constructing a small graph around the target pages, but instead of aggregating the pages that doesn't belong to this small graph, their approach is to use a heuristic to estimate the PageRank of each boundary page of the small graph and run the standard PageRank algorithm on the subgraph, in each step putting the estimated values into the boundary pages, adding the random jump value to the internal pages, and removing any flow leaving the subgraph. Since this algorithm does not require the state aggregation step, its computation is faster than the previous approaches. However, it does not provide scores for all pages. Here again, the choice of the size of the small graph affects the accuracy of the estimation.

3.6.3 P2P-oriented Approaches

In P2P networks, the Web graph is not nicely partitioned according to the server or host levels, so the graph partitioning approaches can not be applied in a P2P scenario. Among the approaches that do not assume a particular data partitioning there is the work by Kempe et al. [KM04], which performs a decentralized spectral decomposition of the graph. They propose an algorithm for computing the top k eigenvectors of a symmetric matrix (i.e., a square matrix that is equal to its transpose), and singular vectors of arbitrary matrices. Computing PageRank scores is a special case of the algorithm, since the PageRank scores vector corresponds to the singular vector associated with the singular value equals to one. The algorithm is based on a decentralized implementation of *Orthogonal Iteration*, a simple method for computing eigenvectors. In this decentralized version, each peer is assumed to know all the incoming links for their pages and is able to communicate the computed values through the pages' outgoing links. Starting with a random initial approximation for the values, each peer improves the previous scores, using the information received from incoming links, and

propagates the updated scores through the outgoing links. This procedure is repeated until the values converge. Convergence is guaranteed and the number of rounds needed for convergence depends on the number of pages and the mixing time of a random walk on the graph. One of the main drawbacks however, is that the algorithm requires the computation to be synchronized, i.e., all peers need to perform one round of the computation and propagate the results before the next round can start.

The work by Canright et al. [CEMJ05] also presents a fully distributed method, inspired by the power method, for the calculation of the principal eigenvector of generic matrices. It also works by recomputing local scores, upon receiving updated scores from incoming neighbors, and propagating the newly computed scores to the outgoing neighbors. Synchronization is also required, but authors relax this constrain by introducing a parameter, called δ, that determines the duration of each round, avoiding the need of a global synchronization mechanism.

Although these two previous approaches eliminate the need of a particular data partitioning, they still require the partitions to be *disjoint*, which makes them suitable for certain classes of distributed systems and also for accelerating link analysis on a cluster of computers, but less attractive for a P2P environment. In a P2P network, disjoint partitioning might be a strong constraint, given that in most P2P networks peers are completely autonomous and crawl and index Web data at their discretion, resulting in arbitrarily overlapping graph fragments. Another drawback of these approaches is the need of some sort of synchronization mechanism (either central or distributed). The algorithm proposed by Sankaralingam et al. [SSB03] for distributed computation of PageRank, on the other hand, is totally asynchronous; every time a message with update scores is received from incoming links, peers compute updated values for the PageRank scores of their local pages and propagating them to the outgoing neighbors. If a page is replicated at different peers, only one of these peers will be responsible for computing the score for the page, and pointers to all other copies of the page need to be maintained, so that all copies of the page can contain the correct computed Pagerank. Even though this is a step towards dealing with overlap, finding all copies and maintaining pointers to them might become very expensive. Shi et al. [SYYW03] present a similar approach, but the communication among peers is reduced by distributing the pages among the peers according to some load-sharing function. However, even after applying load balancing the message cost might still be too high; therefore other approaches, including our own JXP algorithm, opt for pair-wise communications among peers, upon requests for updated scores.

Another example of decentralized PageRank computation for P2P networks is the work by Sozio et al. [SPCW08], where peers exchange information through peer meetings. The algorithm works as follow: each peer asynchronously and independent of other peers, chooses another peer in the network and exchange information, that is then used for refining the local scores. The updated scores will be sent to another peer only when a new meeting occurs, instead being

broadcasted to the peer's neighbors. This last algorithm also addresses the problem of peers acting maliciously, by replicating the pages across the network in a way that a majority of the copies are placed in honest peers, with high probability.

Chapter 4

The JXP Algorithm

The goal of the JXP algorithm (Juxtaposed Approximate PageRank) is to approximate global authority scores by performing local computations only, with low storage costs, and a moderate number of interactions among peers. It runs on every peer in the network, where each peer stores only its own local fragment of the global graph. The algorithm does not assume any particular assignment of entities (e.g., pages) to peers, and overlaps among the link-graph fragments of the peers are allowed.

4.1 The Algorithm

The idea of the algorithm is simple, yet it is quite powerful. Starting with the local graph G of a peer, i.e., the local collection of connected pages, it first extends G by adding one special node, the *world node*. The algorithm then consists of two components:

1. Computation of authority scores performed by each peer on its extended local graph,

2. Interaction with other peers, chosen at random.

Through the interaction with other peers, peers obtained updated information about other peers' contents, which is locally stored in a way that does not hurt scalability and is used in a subsequent local authority computation for refining the scores. After a modest number of interactions, the scores obtained, called JXP scores, already provide a good approximation of the true PageRank scores given by a centralized computation on the union of the local graphs. An analysis of the algorithm shows that, with a sufficient number of peer meetings, scores are guaranteed to converge to the true PageRank values. The following sections explain the JXP algorithm in detail.

4.1.1 Extended Local Graph

The local graph of each peer is extended by adding a special node, coined *world node*. The purpose of the world node is to represent the part of the global graph that is not stored in the peer. Condensing information in one node is very important, otherwise if we add all pages learned in the peer meetings we may end up having each peer storing the complete web graph. Figure 4.1 illustrates how the global Web graph is represented at a peer; the graph on the left represents the global Web graph. In this example we have three peers, Peers A, B and C, that have crawled parts of the Web. Note that there is overlap among the peers' collections. On the right we see how the Web graph is represented at Peer A: the yellow circles are the local pages, i.e., pages that are stored at the peer, and they appear on the extended local graph like they are in the original Web graph. All other pages (in the example, the red and green circles) are external pages, i.e., pages that are store in other peers, and in the extended local graph they are replaced by the world node.

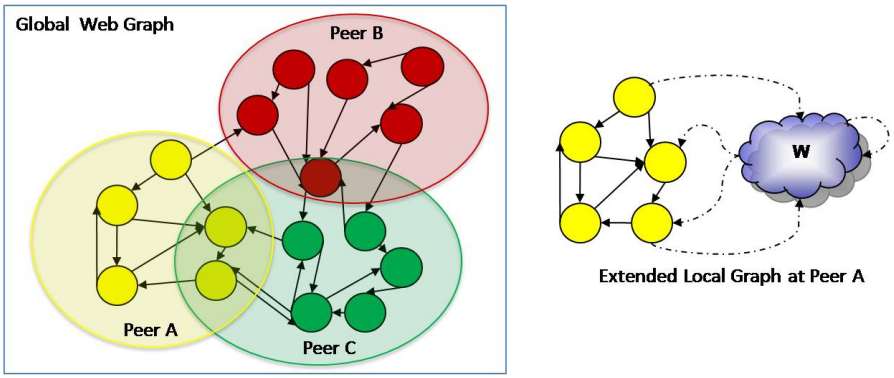


Figure 4.1: How the global Web graph is modeled by the extended local graph. Yellow circles represent local pages, whereas external pages are shown in red and green.

The world node has special features, regarding its own score and how it is connected to the local graph. As it represents all the pages that are not stored in the peer, we take all the links from local pages to external pages and make them point to the world node. In the same way, as the peer learns from external links that point to one of the local pages, we assign these links to the world node. For a better approximation of the amount of authority that is received from external pages, we weigh every link that comes from the world node based on how much of the authority score is received from the original page that owns the link. For this purpose, each peer keeps at its world node the following information:

- a list of external pages that have links to a local page;
- the current known scores for the pages in this list;

- the outdegree of the pages in this list.

For example, if there is an external page j with a link to a local page i, we will represent this link as a link from the world node w to page i, and its weight will be given by

$$weight(j) = \frac{\alpha_j}{out(j)} \frac{1}{\alpha_w},$$

where α_j and α_w are the current JXP scores of page j and the world node, respectively, and $out(j)$ is the outdegree of page j. The total weight of the link from the world node to a local page i will be the sum over the weight contributions of all external pages represented at the world node that points to i. More details in Section 4.2.

Another special feature of the world node is that it contains a self-loop link, that represents links from external pages pointing to other external pages. The score of the world node is equivalent to the sum of the JXP scores of the pages it represents. The scores computation is always done at the extended graph after a peer meeting.

4.1.2 Peer Meetings

Since local information is not sufficient to estimate global authority scores, peers improve their knowledge by meeting other peers in the network and exchanging, through messages, the information they currently have that is relevant for link authority computation. The content of a message can be described a list of tuples of the form

$$< source_{id}, target_{id}, out(source_{id}), \alpha_{source_{id}} >,$$

where $source_{id}$ are the identifiers of pages from the local graph or stored in the world node, and $target_{id}$ are the pages pointed by $source_{id}$.

The information exchanged is then combined by both of the two meeting peers, asynchronously and independently of each other. This works as follows: upon receiving the other peer's message, the peer checks for pages that have links pointing to some of the local pages. This is done by checking for every tuple if the $target_{id}$ is one of the local pages. If $source_{id}$ page is also part of the local graph, the tuple is discarded, otherwise one page would be represented more than once). All tuples that satisfy this constraints are then added to the world node, by updating the links from the world node to the local pages as follows (for simplicity $source_{id}$ and $target_{id}$ are replaced by j and i, respectively):

$$p^t_{wi} = p^{t-1}_{wi} + weight(j),$$

where p^t_{wi} and p^{t-1}_{wi} are the weights of the link from the world node to page i at the current meeting and at the previous meeting, respectively. If p^{t-1}_{wi} is equal to zero, it means that there was no link from the world node to the page prior to the meeting, therefore a new link will be added. A non-zero value of p^{t-1}_{wi} means

that there is already a link, and its weight will be simply updated. More details in Section 4.2. Note that the size of the extended graph never increases, and it is just a small fraction of the global graph. This guarantees the scalability of JXP scores computation. An example of a meeting step is given in Figure 4.2, which shows two peers before and after they have exchanged information. The list of external pages that have links to a local page is shown next to each world node, and the information added or updated is highlighted in red.

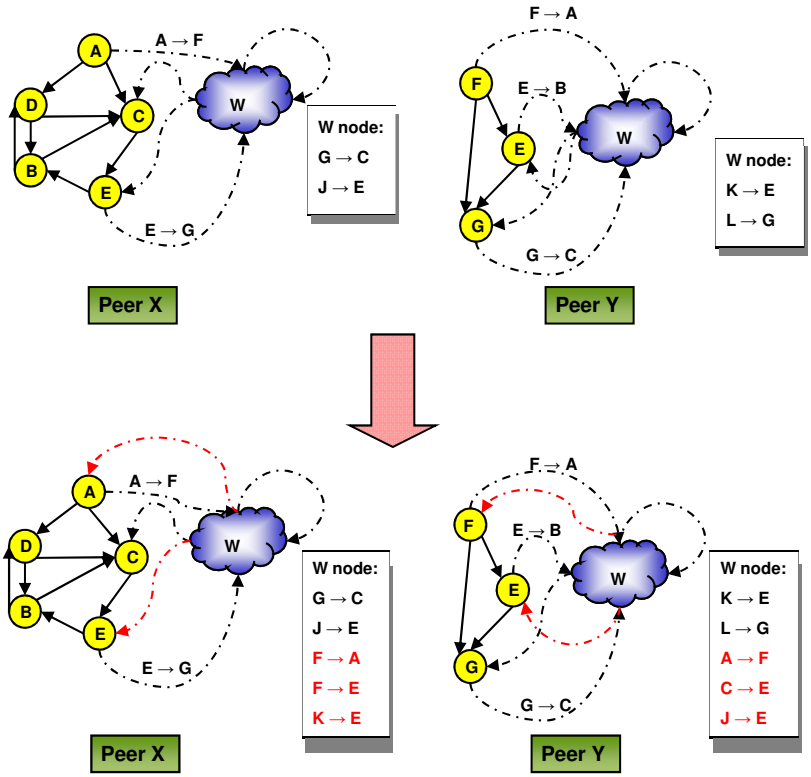


Figure 4.2: Illustration of a peer meeting, where information added or updated is highlighted in red.

Meetings are asynchronous, and multiple meetings with the same peer are needed to get the most updated scores. In addition, data fragments are obtained by each peer independently, so overlaps among local graphs can occur. Therefore, when updating the world node by adding new pages, it can be the case that the page is already there, and its score might be different for the score currently reported by the meeting peer. In these cases, we need to decide which score to keep. Considering the authority mass transfer, it is intuitive that, from meeting to meeting, more and more authority mass is given to local pages as the peer learns about more incoming links; so the score of the world node should always reduce until the point it is equal to the sum of the true PageRank scores of the external pages (we will address this property in Section 4.2, where we

prove that this is indeed the case). Based on this consideration, when faced with two different scores for a particular page, we always take the bigger one of the two scores,i.e.,

$$\alpha_j^t = max(\alpha_j^{t-1}, \alpha_j^{msg}),$$

where α_j^t and α_j^{t-1} are the scores of page j (stored in the world node) at the current meeting and at the previous meeting, respectively, and α_j^{msg} is the score of j in the message received during the meeting. The equation also applies if page j is not yet stored at the world node. In this case the value α_j^{t-1} is set to zero.

The rationale of always taking the biggest score is justified by the fact that the world node's score is monotonically non-increasing in the sequence of peer meetings. Finally, it is important to emphasize that information from local pages given by other peers are not considered, since the peer itself is able to compute the scores for those pages.

4.1.3 JXP Scores

Given the extended local graph G' (local pages plus world node), and the meeting procedure we can now explain how the JXP scores are computed. JXP scores corresponds to the PageRank scores on the extended local graph, where the random jump probabilities to each local page is inversely proportional to the size of the global graph (N), and the probability of a random jump to the world node is proportional to the number of pages it represents. More formally,

$$\alpha_i = \varepsilon \times \sum_{\substack{j|j\to i \\ j\in G'}} \frac{\alpha_j}{out(j)} + (1-\varepsilon) \times RJ(i) \tag{4.1}$$

where

$$RJ(i) = \begin{cases} \frac{N-n}{N}, & \text{if } i \text{ is the world node} \\ \frac{1}{N}, & \text{otherwise} \end{cases}$$

and n is the size of the local graph, and a link from page j to page i is represented by $j \to i$.

Before the execution of the PageRank algorithm, an initialization procedure, described in Algorithm 4.1, is performed. This procedure creates the world node and attach it to the local graph, sets the initial JXP scores and runs the PageRank algorithm on the extended graph, to improve the initial scores.

Since the world node represents all external pages, in the extended local graph G', any link from a local page to an external page is represented as a link from the local page to the world node. The link weight from a page i to w is defined as

$$p_{iw} = \sum_{\substack{i|i\to j \\ j\notin G}} \frac{1}{out(i)}.$$

Algorithm 4.1 Initialization Step

1: **input:** Local graph G and size (number of pages) of global graph N
2: $n \leftarrow size(G)$
3: $G' \leftarrow G \cup w$
4: **for each** $i \in G$ **do**
5: set p_{iw}
6: **end**
7: $p_{ww} = 1$
8: set initial scores α^{init}
9: $\alpha^0 = PageRank(G', \alpha^{init}, n, N)$

The world node also has a self-loop link, representing all transition probabilities among external pages. Since we know that transition probabilities are normalized and initially there is no other link leaving the world node the initial p_{ww} value is one.

The vector α^{init} contains the initial JXP scores, and for every page i in G', the initial score is given by

$$\alpha_i^{init} = \begin{cases} \frac{N-n}{N}, & \text{if } i \text{ is the world node} \\ \frac{1}{N}, & \text{otherwise} \end{cases}$$

The function *PageRank*() takes as input the extended local graph, the initial score vector, the local and global graph sizes and performs the equation described in Equation 4.1, outputting a vector that contains the updated JXP scores.

JXP assumes that the total number of pages in the global graph is known or can be estimated with decent accuracy and consistently among all peers. This is not a critical assumption; there are efficient techniques for distributed counting with duplicate elimination [JMB05, KDG03, BMTW06], and we show later that a wrong estimate of the number global graph size only causes a rescaling on the JXP scores, while the ranking order of the pages is preserved.

After the initialization step, peers are ready to start meetings and exchange information. JXP scores are then refined at every new meeting, after the world node is updated, by re-running the PageRank algorithm on the updated extended local graph. Pseudocode for JXP algorithm is shown in Algorithm 4.2.

The function *selectPeer*() chooses a peer in the network for message exchanging. It can either choose a random peer or use a more sophisticated heuristic, as we will show later. The list of tuples with the local information is created with a *createMsg*() method. Upon receiving the response from the chosen peer, the information contained in the message received is added to the world node. As we explained earlier, the content of the message is first filtered by keeping only tuples that represent links from external pages to local pages. The relevant tuples are then added to the world node, and the weight of the links from the world node are updated as follows: the transition probabilities to local pages are given by

Algorithm 4.2 JXP Algorithm

1: **input:** extended local graph G', JXP score vector α^{t-1}, n and N
2: **do forever**
3: $P \leftarrow selectPeer()$
4: $msg \leftarrow createMsg(G', \alpha^{t-1})$
5: $send(P, msg)$
6: $msg_P \leftarrow receiveMsg(P)$
7: $relSet \leftarrow relevantSet(msg_P)$
8: **for each** $j \in relSet$ **do**
9: $\alpha_j^t = max(\alpha_j^{t-1}, \alpha_j^{msg_P})$
10: **end**
11: update transition probabilites from w
12: $\alpha^t = PageRank(G', \alpha^{t-1}, n, N)$
13: **end**

$$p_{wi}^t = p_{wi}^{t-1} + \sum_{\substack{j|j\rightarrow i \\ j\in relSet}} weight(j)$$

$$p_{ww} = 1 - \sum_{i\in G} p_{wi},$$

where *relSet* is the set of pages in the message received that are relevant for the peer, i.e., external pages that have links to local pages. After updating the weights of links from the world node to local pages, the weight of the self-loop link is updated in a way that that sum of link weights leaving the world node is one. After the link weights are updated the *PageRank*() function is called again, and new updated scores are computed.

Note that since pages stored at the world node are not explicitly represented at the extended local graph their scores are not affected during the PageRank computation and are only updated when another peer reports a higher score for them. By doing so, every peer is only responsible for computing the scores of pages it stores. Next, we proceed with a more theoretical analysis of the algorithm.

4.2 Mathematical Analysis and Convergence Guarantee

The theoretical analysis of the algorithm provide important properties of the JXP scores, as well as a proof for the correctness of the JXP method. We show that JXP scores converge to the correct values, the global PageRank scores of the individual pages, or equivalently, the stationary visiting probabilities of the underlying global Markov chain.

Our analysis builds on the theory of state aggregation in Markov chains [Cou77, Ste94, Mey00, KS63]. However, applying this theory to our setting

is not straightforward at all, and we use it only for particular aspects. State-aggregation techniques assume complete knowledge of the Markov chain and are typically used to speed up the convergence of computations (see, e.g., [LM06b, CDK+03]). In contrast, our P2P setting poses the difficulty that each peer has only limited knowledge of the Web graph and the resulting Markov Model. Moreover, this restricted view differs from peer to peer.

For the proof we assume that there are no changes in the network, so there exists a global Web graph with N pages, a global transition matrix $\mathbf{C}_{N\times N}$ and a global stationary distribution vector π. The element c_{ij} of $\mathbf{C}$ is equal to $1/out(i)$ if there is a link from page i to page j, and zero otherwise. After adding the random jump probabilities we have a transition matrix $\mathbf{C}'$

$$\mathbf{C}' = \varepsilon\ \mathbf{C} + (1-\varepsilon)\frac{1}{N}\mathbf{1}_{N\times N}.$$

Every peer has a local graph G, subgraph of the global web graph, that corresponds to the set of pages it has crawled. Pages that are not in G are considered to be on the set $\overline{G}$. The local graph is extended by adding the world node. The set of external pages that are represented in the world node w is given by W, and for every page r in W we store the information about its outdegree, $out(r)$, and current JXP score $\alpha(r)$, both learned from a previous meeting. The number of local pages is given by n. Associated with each extended local graph we have a local transition matrix $\mathbf{P}$ that has the following format

$$\mathbf{P}_{(n+1)\times(n+1)} = \left(\begin{array}{ccc|c} p_{11} & \cdots & p_{1n} & p_{1w} \\ \vdots & \ldots & \vdots & \vdots \\ p_{n1} & \cdots & p_{nn} & p_{nw} \\ \hline p_{w1} & \cdots & p_{wn} & p_{ww} \end{array}\right),$$

where

$$p_{ij} = \begin{cases} \frac{1}{out(i)} & \text{if } \exists\ i \to j\ \&\ out(i) \neq 0, \\ \frac{1}{N} & \text{if } out(i) = 0 \\ 0 & \text{otherwise} \end{cases}, \text{ and}$$

$$p_{iw} = \sum_{\substack{i|i\to r \\ r\notin G}} \frac{1}{out(i)},$$

for every i, j, $1 \leq i, j \leq n$. Note that if a page has no outlinks we replace the respective zero row of $\mathbf{P}$ by $\frac{e^T}{N}$, where e is a column of ones, i.e., we make the dangling page point to every page in the graph.

The transition probabilities from the world node, p_{wi} and p_{ww}, change during the computation, so they are defining according to the current meeting t,

$$p_{wi}^t = \sum_{\substack{r|r\to i \\ r\in W^t}} \frac{\alpha(r)^t}{out(r)} \cdot \frac{1}{\alpha_w^{t-1}}\ , \text{ and}$$

$$p^t_{ww} = 1 - \sum_{i=1}^{n} p^t_{wi}.$$

For the JXP computation, random jumps are also added, with the particularity that the random jumps to the world node are made proportional to the number of pages it represents. This gives us the following transition matrix

$$\mathbf{P}' = \varepsilon\ \mathbf{P} + (1-\varepsilon)\frac{1}{N}\mathbf{1}_{(n+1)\times 1}\left(\begin{array}{ccc|c} 1 & \dots & 1 & (N-n) \end{array}\right), \tag{4.2}$$

which has a stationary distribution vector α

$$\alpha = \left(\begin{array}{ccc|c} \alpha_1 & \dots & \alpha_n & \alpha_w \end{array}\right)$$

that corresponds to the JXP scores.

4.2.1 Initialization Procedure

We start with a local transition matrix, $\mathbf{P}^0$, with all p_{wi} elements equal to zero since the peers start with no knowledge about external pages. The element p_{ww} is consequently set to 1,

$$\mathbf{P}^0_{w*} = \left(\begin{array}{ccc|c} 0 & \dots & 0 & 1 \end{array}\right).$$

The local JXP scores vector is initially set to

$$\alpha^{init} = \left(\begin{array}{ccc|c} \frac{1}{N} & \dots & \frac{1}{N} & \frac{N-n}{N} \end{array}\right).$$

The PageRank computation is then performed using the transition matrix $\mathbf{P}'^0$ and an updated value for the local authority scores vector α^0 $(t=0)$ is obtained.

4.2.2 The Meeting Step

As described earlier, the meeting process consists of adding new links, or updating existing links from the world node to the local pages, and performing the PageRank algorithm using the updated transition matrix.

Consider the follow local transition matrix and its local JXP scores vector at meeting $(t-1)$ $(t \geq 1)$

$$\mathbf{P}^{t-1}_{(n+1)\times(n+1)} = \left(\begin{array}{ccc|c} p_{11} & \dots & p_{1n} & p_{1w} \\ \vdots & \dots & \vdots & \vdots \\ p_{n1} & \dots & p_{nn} & p_{nw} \\ \hline p^{t-1}_{w1} & \dots & p^{t-1}_{wn} & p^{t-1}_{ww} \end{array}\right),$$

$$\alpha^{t-1} = \left(\begin{array}{ccc|c} \alpha^{t-1}_1 & \dots & \alpha^{t-1}_n & \alpha^{t-1}_w \end{array}\right).$$

For the sake of simplicity, we split the meeting step, by considering only one link addition/update at a time. Assuming that during meeting t a link to page i has been added or updated, we can express p_{wi} at time t as

$$p_{wi}^t = p_{wi}^{t-1} + \delta.$$

Since the authority scores of external pages on the meeting step can only increase or remain unchanged we can assure that the value of δ is always non-negative.

As the transition probability from the world node to itself is always adjusted to compensate for changes of the other transition probabilities we can also write

$$p_{ww}^t = p_{ww}^{t-1} - \delta.$$

The transition matrix at meeting t can then be written as

$$\mathbf{P}^t = \mathbf{P}^{t-1} + \mathbf{E},$$

where

$$\mathbf{E} = \left(\begin{array}{ccccccc|c} 0 & & & \dots & & & 0 & 0 \\ \vdots & & & \dots & & & \vdots & \vdots \\ 0 & & & \dots & & & 0 & 0 \\ \hline 0 & \dots & 0 & \delta & 0 & \dots & 0 & -\delta \end{array}\right),$$

which leads to an updated JXP scores vector

$$\alpha^t = \left(\begin{array}{ccc|c} \alpha_1^t & \dots & \alpha_n^t & \alpha_w^t \end{array}\right).$$

The following two theorems describes important properties about the JXP scores.

Theorem 4.2.1 *The JXP score of the world node, at every peer in the network, is monotonically non-increasing.*

Proof The proof is based on the study of the sensitivity of Markov Chains made by Cho and Meyer [CM00]. From there we can state that by increasing p_{wi} by δ and decreasing p_{ww} by the same amount, the following holds

$$\frac{\alpha_w^{t-1} - \alpha_w^t}{\alpha_w^{t-1}} = \alpha_w^t \ \ \delta \ m_{iw},$$

where m_{iw} is the mean first passage time from page i to the world node (i.e., the expected number of steps for reaching w when starting in i, in the underlying Markov chain). Rearranging the terms on the equation we have

$$\alpha_w^t - \alpha_w^{t-1} = -\alpha_w^{t-1} \ \alpha_w^t \ \delta \ m_{iw}.$$

Since all the values on the right side of the equation are non-negative we can assure that

$$\alpha_w^t - \alpha_w^{t-1} \leq 0. \quad \blacksquare$$

Theorem 4.2.2 *The sum of scores over all pages in a local graph, at every peer in the network, is monotonically non-decreasing.*

Proof The proof follows from Theorem 4.2.1 and the fact that the following equality holds

$$\sum_{i \in G} \alpha_i + \alpha_w = 1. \quad \blacksquare$$

4.2.3 Scores Bounds

We now proceed by showing how the JXP scores and the global PageRank scores are related. The next theorem shows that the global PageRank values are an upper bound for the JXP scores.

Theorem 4.2.3 *Consider the true stationary probabilities (PageRank scores) of pages $i \in G$ (the local graph) and the world node w, π_i and π_w, and their JXP scores after t meetings α_i^t and α_w^t. The following holds throughout all JXP meetings:*
$0 < \alpha_i^t \leq \pi_i$ *for* $i \in G$ *and* $\pi_w \leq \alpha_w^t < 1$.

Proof We know that for every page $i \in G$:

$$\pi_i = \frac{1-\varepsilon}{N} + \varepsilon \sum_{\substack{j|j \to i \\ j \in G}} \frac{\pi_j}{out(j)} + \varepsilon \sum_{\substack{j|j \to i \\ j \in \overline{G}}} \frac{\pi_j}{out(j)},$$

and

$$\alpha_i^t = \frac{1-\varepsilon}{N} + \varepsilon \sum_{\substack{j|j \to i \\ j \in G}} \frac{\alpha_j^t}{out(j)} + \varepsilon \sum_{\substack{j|j \to i \\ j \in W^t}} \frac{\alpha_j^t}{out(j)} \frac{\alpha_w^t}{\alpha_w^{t-1}}.$$

where W^t is the set of external pages that are represent in the world node at meeting t.

We prove the claim about the α_i^t values by induction on t; the proof for the claim on the world node follows directly from the fact that the score vector is normalized. The claims that $\alpha_i > 0$ and $\alpha_w^t < 1$ are trivial to show.

For $t = 0$ we consider the situation that a given peer with graph G knows only its local graph and has no information about the world node other than the total number of pages, N. Thus the peer assumes that the only transfer of score mass from w to any node in G is by random jumps, which is the minimum transfer that is possible. Since G includes outgoing links to w, a local PageRank computation based on this setting cannot overestimate and will typically underestimate the scores of pages in G.

Now assume that the claim holds for all meetings up to and including t, and consider the $t+1^{st}$ meeting.

First we observe that because of $\alpha_w^t \leq \alpha_w^{t-1}$ (by Theorem 4.2.1), $W^t \subseteq \overline{G}$, and the induction assumption $\alpha_j^t \leq \pi_j$, the following upper bound holds for the third summand (abbreviated as β_i):

$$\varepsilon \sum_{\substack{j|j\to i \\ j\in W^t}} \frac{\alpha_j^t}{out(j)} \frac{\alpha_w^t}{\alpha_w^{t-1}} \leq \varepsilon \sum_{\substack{j|j\to i \\ j\in \overline{G}}} \frac{\pi_j}{out(j)} := \beta_i.$$

Now consider the following upper bound for α_i^{t+1}:

$$\alpha_i^{t+1} \leq \frac{1-\varepsilon}{N} + \varepsilon \sum_{\substack{j|j\to i \\ j\in G}} \frac{\alpha_j^{t+1}}{out(j)} + \beta_i.$$

In the $t+1^{st}$ meeting page i could increase its α_i value in three ways: a) by learning about an additional page $x \in W^{t+1}$ with $x \notin W^t$ that points to i, b) by learning that a previously known page $x \in W^t$ that points to i has a higher value $\alpha^{t+1}(x)$ than the last time that a peer with x in its local graph was met (i.e., at some previous iteration $t' < t+1$), or c) the value α_j^{t+1} of some incoming neighbor j from the peer's own local graph G ($j \in G$) has a higher value than in previous iterations. No other cases are possible.

The last case is impossible unless one of the cases a) or b) occurs, simply because all outdegrees are fixed and, without any external changes, the local PageRank computation on G will reproduce the scores computed in earlier iterations. But by the induction assumption we have $\alpha_i^t \leq \pi_i$ for all previous t. In the first and second case we can conservatively assume the upper bound β_i for whatever increased score the pages in W^{t+1} may transfer to i or any other pages in G. Thus we have

$$\begin{aligned} \alpha_i^{t+1} &\leq \frac{1-\varepsilon}{N} + \varepsilon \sum_{\substack{j|j\to i \\ j\in G}} \frac{\alpha_j^{t+1}}{out(j)} + \beta_i \\ &\leq \frac{1-\varepsilon}{N} + \varepsilon \sum_{\substack{j|j\to i \\ j\in G}} \frac{\pi_j}{out(j)} + \beta_i = \pi_i. \end{aligned}$$

∎

Theorem 4.2.3 does not explicitly reflect the fact that pages from two local graphs can overlap. We assumed that in these cases the pages are treated as local pages, and we take their α_j values from the peer's local bookkeeping. However, because all peers, by Theorem 4.2.3, invariantly underestimate the true stationary probability of these pages, we can safely use the maximum of the α_j values from the two peers in a meeting: the maximum is still guaranteed to be upper-bounded by the true PageRank score π_j.

4.2.4 Proof of Convergence

Theorem 4.2.3 is a safety property in that it shows that we never overestimate the correct global PageRank scores. What remains to be done is to show liveness in the sense that JXP makes effective progress towards the true PageRank scores. The argument for this part is based on the notion of fairness from concurrent

programming theory (see, e.g., [Lam02]): a sequence of events is fair with respect to event e if every infinite sequence has an infinite number of e occurrences. In our setting, this requires that in an infinite number of P2P meetings, every pair of peers meet infinitely often. Truly randomized meetings with uniform distribution have this property, but there are other ways as well. A similar argument has been used in [APC03] for online page importance.

Theorem 4.2.4 *In a fair series of JXP meetings, the JXP scores of all pages converge to the true global PageRank scores.*

Proof The fairness property ensures that at some point, say after the t^{th} meeting, every peer knows all its incoming neighbors, the complete sets $\{j|j \rightarrow i, j \in \overline{G}\}$ for all $i \in G$. At this point, the only reason why a peer's local JXP score α_i^t for some page i may still underestimate the global PageRank score π_i is that the JXP scores of the incoming neighbors from outside of G may also be underestimated, i.e., $\alpha_j^t < \pi_j$ for some $j \in W$. We show that this situation cannot hold indefinitely, once all the incoming links from external pages are completely known.

There are two cases to consider. The first case is when the world node's JXP score $\alpha_w^{\hat{t}}$ has converged at some point $\hat{t} \geq t$ so that $\alpha_w^{\hat{t}} = \pi_w$ holds (strictly speaking, the difference between the α and the π value is below some ξ that can be made arbitrarily small; we simplify the argument for simpler notation). At this point, we can infer that $\sum_{i \in G} \alpha_i^{\hat{t}} = \sum_{i \in G} \pi_i$. So if some $\alpha_i^{\hat{t}}$ is still strictly below its PageRank score π_i, some other page $j \in G$ must have an $\alpha_j^{\hat{t}}$ value strictly higher than its PageRank score π_j. But this is impossible because of Theorem 4.2.3.

The second case is that $\alpha_w^{\hat{t}} < \pi_w$ holds and stays invariant in all subsequent meetings. But then we have $\alpha_w^{\hat{t}+1} = \alpha_w^{\hat{t}}$ which implies:

$$\begin{aligned}\alpha_i^{\hat{t}+1} &= \frac{1-\varepsilon}{N} + \varepsilon \sum_{\substack{j|j \rightarrow i \\ j \in G}} \frac{\alpha_j^{\hat{t}+1}}{out(j)} + \varepsilon \sum_{\substack{j|j \rightarrow i \\ j \in \overline{G}}} \frac{\alpha_j^{\hat{t}+1}}{out(j)} \\ &= \frac{1-\varepsilon}{N} + \varepsilon \sum_{j|j \rightarrow i} \frac{\alpha_j^{\hat{t}+1}}{out(j)}.\end{aligned}$$

This is the very same fixed point equation that we have for the true PageRank scores, the π_i values. We know that this fixed point equation has a unique solution [BP98, KS63, Ste94]; thus the above equation must have the same solution as the equation for the π_i values, and so the JXP scores eventually equal the PageRank score. (Again, strictly speaking, the difference drops below some ξ that can be chosen arbitrarily small.) ∎

4.3 Storage and Network Bandwidth Costs

For a P2P application to scale, it is fundamental that each peer's storage and network bandwidth costs remain under a certain limit. The cost analysis that

follows shows that the JXP algorithm meet these constraints.

At each peer, the size of the extended graph is fixed and equal to $(n+1)$ pages (local pages + world node). For each local page the storage requirement is fixed and corresponds to its unique identifier (ID), the list of outgoing links, and current JXP score. The increase on the storage occurs only at the world node, where the lists of external pages with incoming links to local pages are kept. For each of these pages we need only to store its ID, outdegree, current JXP score, and the IDs of the local pages to which it points to. With that we can approximate the authority mass transfer from this page to the local pages. Denoting by SC_P the storage cost at peer P we can write:

$$SC_P = SC_G + SC_w$$

$$SC_G = \sum_{i \in G} (SC_{ID} + SC_{score} + out(i) \cdot SC_{ID})$$

$$SC_w = \sum_{i \in G} \sum_{\substack{j | j \in W \\ j \to i}} (SC_{ID} + SC_{score} + SC_{out})$$

The peer's local graph and world node are denoted by G and w, W is the set of pages represented at w, $out(i)$ is the outdegree of entity i, and SC_{ID}, SC_{score}, SC_{out} are the costs of storing the identity, scores and outdegree of an page, respectively.

The local storage cost is linear in the number of incoming and outgoing links of local pages. An extensive study of the Web structure [BKM+00] has shown that the indegree and outdegree distributions follow a power law (Pareto distribution) and on average each page has only a handful of in-links and out-links. Therefore storage cost is $O(n)$, where n is the size of the local graph.

Exchanging information among peers has also a low cost. This is mainly due to the fact that the analysis is carried out on the link structured of the graph only, without the need of the actual contents of the pages in the graph. Moreover, since meetings are asynchronous, peers can decide when is the best time to send a message to another peer, and the message itself can additionally be broken into smaller sub-messages and be piggybacked onto the existing communication among peers.

When sending a message, the peer do not know a priori what is stored in the other peer, so all the information about the extended local graph is sent. Therefore communication costs are similar to storage cost, i.e., $O(n)$.

An experimental evaluation on the communication costs was made and is presented later in this chapter. Moreover, we will later introduce techniques that can further reduce communication costs, where compact representation of the links of a peer can be used to find potential peers for a meeting, and avoid the exchange of information with peers that do not significantly contribute for the local computation.

4.4 Robustness Against Wrong Estimates of Graph Size

The JXP algorithm assumes knowledge of the total number of distinct pages in the P2P network in order to compute the random jumps probabilities and correctly converge to the global PageRank values. Although there are efficient techniques for distributed counting with duplicate elimination [JMB05, KDG03, BMTW06], the need for knowing this global quantity could be a problem.

Our studies have found that the true value of the number of pages in the network is only needed when we are interested in the correct absolute values for the stationary probabilities of the pages. For cases where the absolute values are not needed, as long as the *ranking* is correct, any choice for the random jump probability is sufficient, as long as the value for the global number of pages is the same across all peers and greater than the largest local collection.

To formalize this result about different values for computing the random jump probabilities we redefine the transition matrix from Equation 4.2 as follows

$$\mathbf{P}'(\mathbf{X}) = \varepsilon\ \mathbf{P} + (1-\varepsilon)\frac{1}{X}\mathbf{1}_{(n+1)\times 1}\left(\begin{array}{ccc|c} 1 & \ldots & 1 & (X-n) \end{array}\right),$$

where X is the value used to replace the global number of pages N. When N is known, we have $X = N$ and the results are the same as given on the previous sections.

The convergence of the JXP algorithm for different choices of X is guaranteed by the following theorem.

Theorem 4.4.1 *The JXP local transition matrices, at every peer, are always stochastic, for any choice of $X > n$.*

Proof By inspection of the matrix $\mathbf{P}'(\mathbf{X})$ we can see that it satisfies all three conditions for being stochastic [Ste94]

1. $p'_{ij} \geq 0$ for all i, j,
2. $\sum_j p'_{ij} = 1$ for all i,
3. At least one element in each column differs from zero.

The first and third conditions require that $X > n$. ∎

Theorem 4.4.1 guarantees that there exists a stationary distribution vector $\alpha(X)$

$$\alpha(X) = \left(\begin{array}{ccc|c} \alpha_1(X) & \ldots & \alpha_n(X) & \alpha_w(X) \end{array}\right) \tag{4.3}$$

associated with each local matrix.

Although this result does not mathematically relate the $\alpha_i(X)$ values with the π_i values, our experiments indicate that $\alpha_i(X)$ values, with $X \neq N$ are related to $\alpha_i(N)$ by a scaling factor, which results in the ranking orders to remain unchanged.

4.5 Experimental Evaluation

An extensive experimental evaluation was conducted to assess the practical behavior of the JXP algorithm. The results presented in this section test algorithm's performance and accuracy.

4.5.1 Data Sets

We evaluated the performance of the JXP algorithm on a collection of pages from the Amazon.com website and on a partial crawl of the Web graph. The Amazon data contains information about products (mostly books) offered by Amazon.com.[1] The data was obtained in February 2005, and the graphs were created by considering the products as nodes in the graph. For each product, pointers to similar recommended products are available in the collection (see Figure 4.3). These pointers define the edges in our graphs. Products are also classified into one or more categories.

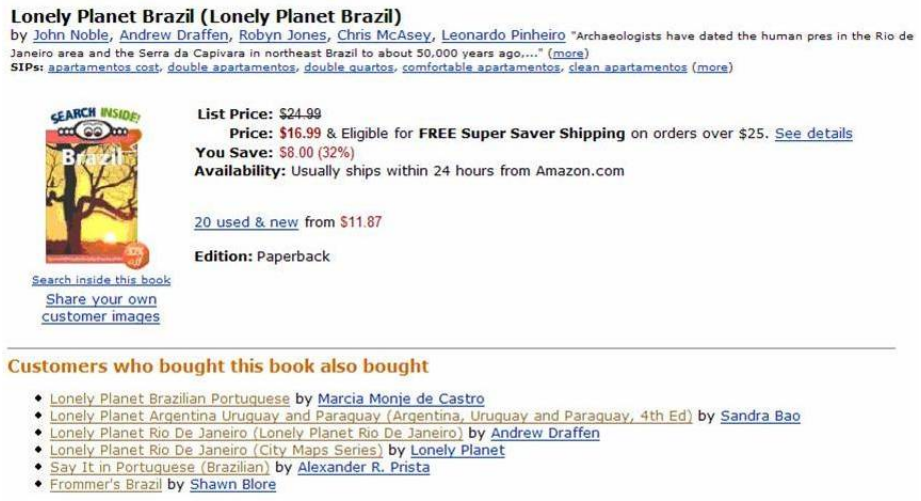


Figure 4.3: Example of an Amazon product page.

We have thematically grouped together some of the original categories, so in the end we had a total of 10 categories, as shown in Table 4.1. In total there are 120,564 pages and 541,551 links in the Amazon data.

The Web Crawl collection was obtained in January 2005, using the Bingo! focused crawler [STS+03]. We first trained the crawler with a manually selected set of pages; then, new pages were fetched and automatically classified into one of 10 predefined categories described at Table 4.2. In total there are 250,760 pages and 3,123,841 links in the Web Crawl data.

[1] *http://www.amazon.com.*

Category	Description
1	“Cooking, Food & Wine”, “Gay & Lesbian”, “Health, Mind & Body” “Home & Garden”, “Parenting & Families”
2	“Arts & Photography”, “Comics & Graphic Novels”, “Entertainment” “Outdoors & Nature”, “Sports”, “Teens”, “Travel”
3	“Children’s Books”, “Horror”, “Literature & Fiction” “Mystery & Thrillers”, “Romance”
4	“Nonfiction”
5	“Business & Investing”, “Computers & Internet”, “Engineering”
6	“Science”, “Science Fiction & Fantasy”
7	“Professional & Technical”
8	“Religion & Spirituality”
9	“Biographies & Memoirs”, “Reference”
10	“History”, “Law”, “Medicine”

Table 4.1: Amazon dataset categories.

Category	Description
1	Arts
2	Finance
3	Health
4	Movies
5	Music
6	Natural Sciences
7	Nature
8	Politics
9	Sports
10	Travel

Table 4.2: Web Crawl dataset categories.

We checked the degree of connectivity to assure that the PageRank computation was meaningful in these datasets. Figures 4.4 and 4.5 show the indegree and outdegree distributions, on a log-log scale for the two collections.

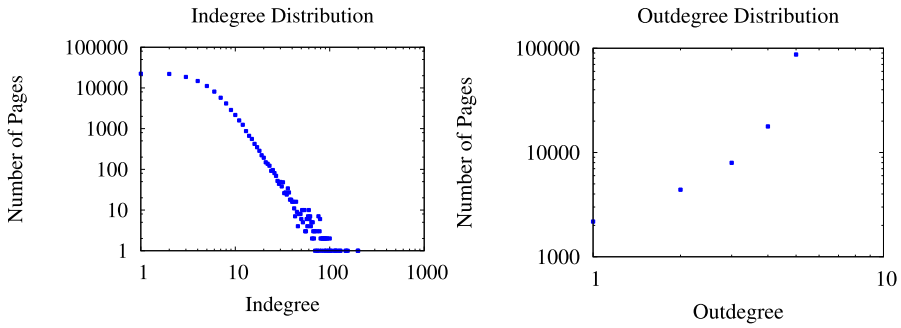


Figure 4.4: Indegree and outdegree distributions for the Amazon dataset.

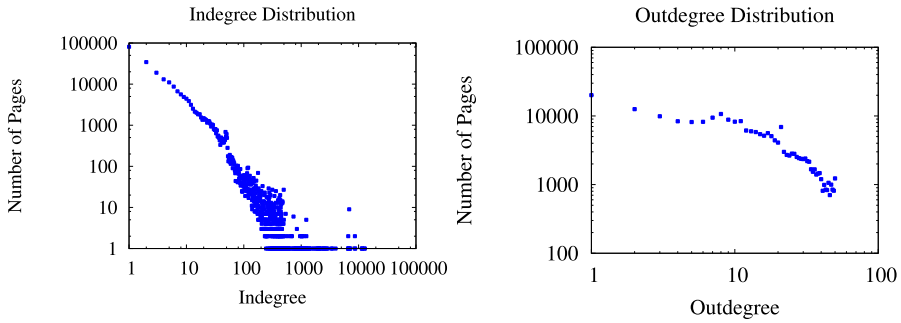


Figure 4.5: Indegree and outdegree distributions for the Web Crawl dataset.

Except for the outdegree distribution of the Amazon dataset, we can observe the power-law distribution, which is also the standard assumption for the complete Web graph. We thus expect that our experiments are fairly indicative for the behavior at Internet scale. The reason why the Amazon collection does not follow this pattern is that related products shown to users are bounded by a small number (in our case the highest outdegree observed was 5), given that showing too many related products is not useful to users.

4.5.2 Setup

JXP peers are implemented in Java 5.0. Local graphs are obtained by having the peers performing independent crawls on the datasets, starting with a set of random seeds pages and following the links and fetching pages in a breadth-first approach, up to a certain predefined depth. Note that due to the crawling strategy there is no guarantee that peers will select and download all available pages in the collection, hence the collection stored at the entirety of peers is a subset of the original one. For a meeting, a peer contacts a randomly chosen

peer in the network, and asks for its current local knowledge. We assume that there is an underlying mechanism which may be invoked by any peer, in order to contact another peer to exchange information with.

4.5.3 Performance Metrics

For evaluating the performance we compare the authority scores given by the JXP algorithm against the true PageRank scores of pages in the complete collection. Since, in the JXP approach, the pages are distributed among the peers and for the true PageRank computation the complete graph is needed, in order to compare the two approaches we construct a total ranking from the distributed scores by essentially merging the score lists from all peers. Note that this is done for the experimental evaluation, it would neither be needed nor desired in the real P2P network. We do this periodically after a fixed number of meetings in the network. Since overlaps are allowed and no synchronization is required, it can be the case that a page has different scores at different peers. In this case, the score of the page on the total ranking is considered to be the average over its different scores.

The total top-k ranking given by the JXP algorithm and the top-k ranking given by traditional, centralized PageRank are compared using *Spearman's footrule distance* [FKS03, DKNS01], defined as

$$F(\sigma_1, \sigma_2) = \sum_{i \in D} |\sigma_1(i) - \sigma_2(i)|,$$

where D is the set of pages that belongs to at least one of the two top-k rankings, $\sigma_1(i)$ and $\sigma_2(i)$ are the positions of the page i in the first and second top-k ranking. In case a page is present in one of the top-k rankings and does not appear in the other, its position in the latter is considered to be $k+1$. We normalize the Spearman's footrule distance to obtain values between 0 and 1, with 0 meaning that the rankings are identical, and 1 meaning that the rankings have no pages in common. We also use the *Linear score error* measure, which is defined as the average of the absolute difference between the JXP score and the global PageRank score over the top-k pages in the centralized PageRank ranking, i.e.,

$$LinearScoreError(score_1, score_2) = \frac{\sum_{i \in Z} |score_1(i) - score_2(i)|}{k},$$

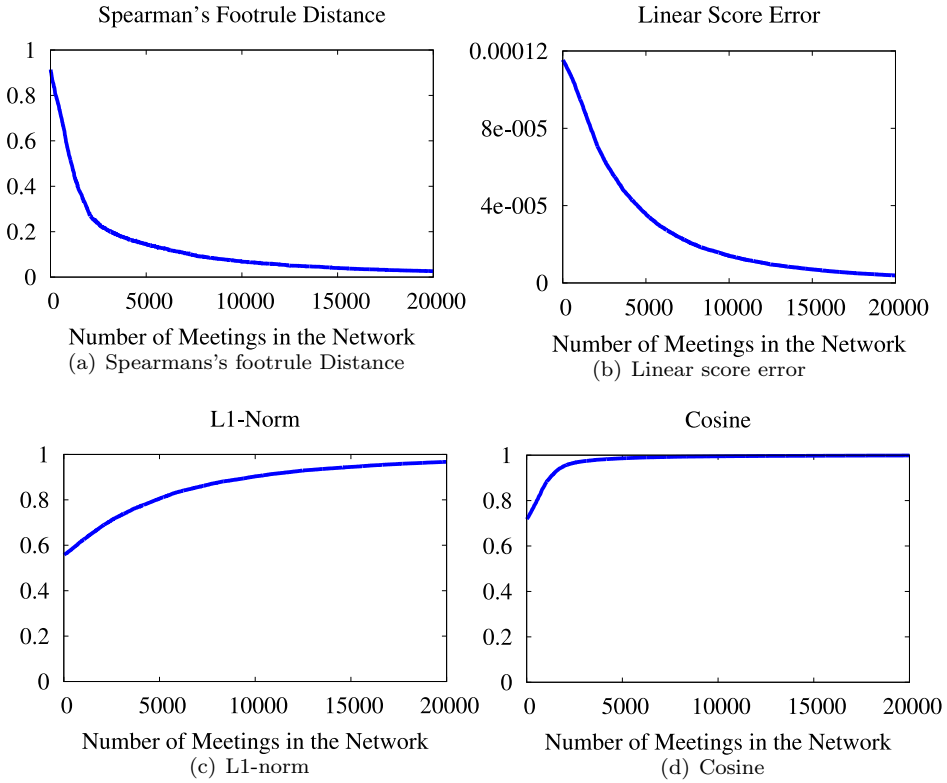
where Z is the set of pages belonging to the top-k ranking in the centralized setting.

In addition, we have computed the cosine between the two full ranking vectors, i.e., the vectors containing all pages in the network, and the L1-norm of the vector containing the JXP scores of all pages (since scores are normalized, the L1 norm for the global PageRank vector is 1).

4.5.4 Results

Accuracy and Convergence

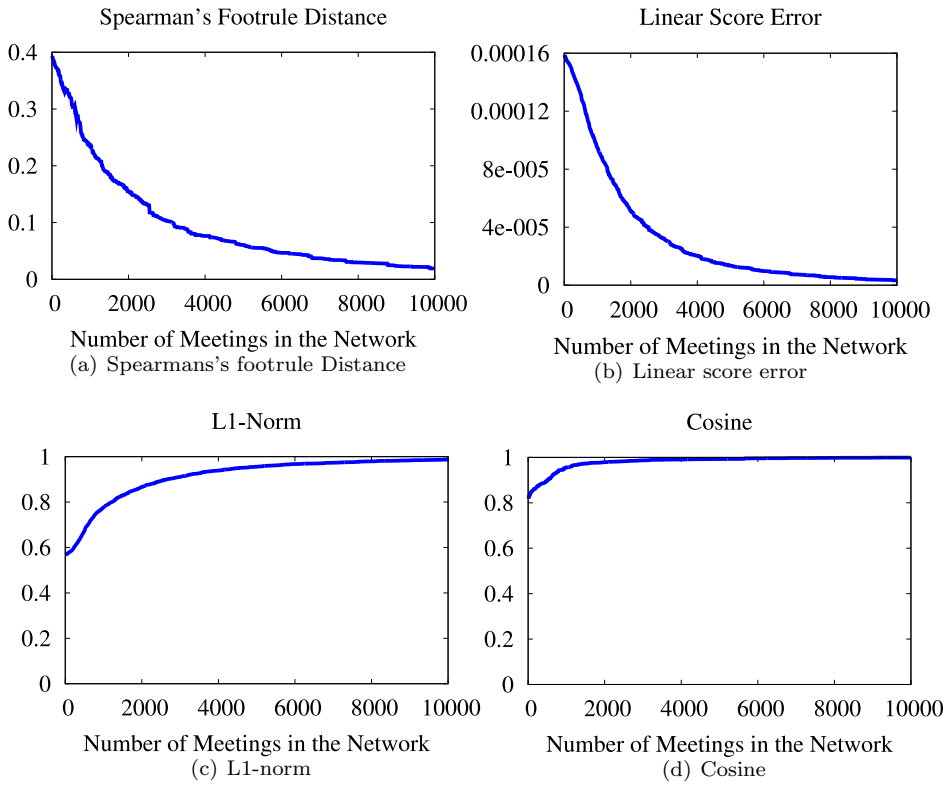
First of all, we studied the general behavior of the JXP method, to test whether it serves its purpose as a P2P approximation of global PageRank. Figures 4.6 and 4.7 show results for the Amazon collection and the Web Crawl collection, respectively. In both cases we have a P2P network with 100 peers, and the scores of the top-1000 highest ranked pages were used. The charts show the measures as functions of the total number of peer meetings in the network.



(a) Spearmans's footrule Distance (b) Linear score error (c) L1-norm (d) Cosine

Figure 4.6: JXP Performance at Amazon Dataset

We see that the error drops quickly as the peers meet other peers. Already at 1500 meetings the footrule distance drops below 0.4 for the Amazon data and below 0.2 for the Web Crawl. At this point, each of the 100 peers, on average, has met and exchanged its graph with 15 other peers. The linear score error shows that the JXP scores converge to the global PageRank values. The other measures also confirm the convergence behavior, since both L1-norm and cosine measures converge to one. These observations demonstrate the practical viability of the JXP method. Moreover, the L1-norm also shows that initially the scores are underestimated, since the only transfer of scores mass from the world node to the local pages is by random jumps. As the meetings are performed,



(a) Spearmans's footrule Distance
(b) Linear score error
(c) L1-norm
(d) Cosine

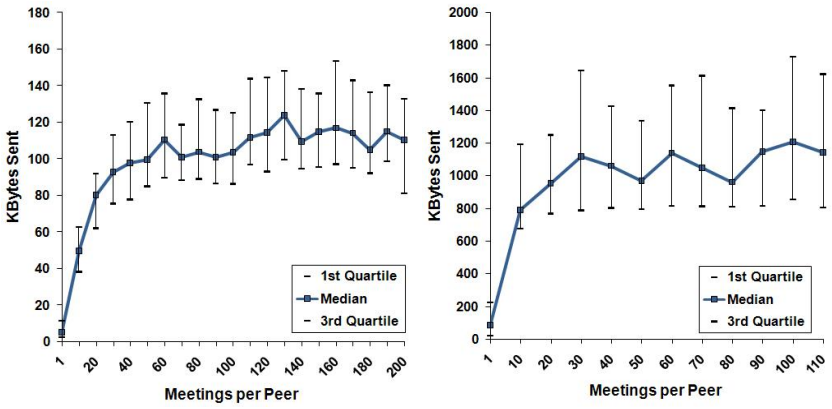
Figure 4.7: JXP Performance at Web Crawl Dataset

more and more authority mass is transfered from the world nodes to the local graphs, and hence the sum of JXP scores increases.

Message Costs

Even though a good approximation of the true PageRank scores can be obtained with a few iterations, convergence requires a considerable number of meetings. However, the size of the transmitted messages is small, since for the JXP computation, no page content is required. We measured, for the same setup presented before, the message size of a peer at each meeting. Figure 4.8 shows the median, the first quartile and the third quartile (in KBytes) for the values at all peers, after each meeting they have performed, for both Amazon and Web Crawl collections.

The results show that JXP consumes rather little network bandwidth, as the message sizes are small. The rapid growth in the first meetings is due to the phase where more pages are added to the world node (and consequently added to the messages transmitted). However as soon as all peers have learned about all their incoming links, message sizes should remain constant. Also recall that meetings are asynchronous, and the time interval between two successive meetings can be adapted to the available bandwidth.



(a) Amazon Dataset (b) Web Crawl Dataset

Figure 4.8: Message size (in kB) for the Amazon and Web Crawl datasets.

Effects of Misestimating the Global Number of Pages

As said before, JXP assumes that the total number of pages in the global graph (N) is known or can be estimated with decent accuracy, which is not a critical assumption, given that there are efficient techniques for distributed counting with duplicate elimination. However, in cases where neither the exact value nor a good approximation are available, any other value for the total number of pages only causes a rescaling on the JXP scores, while the ranking order of the pages is preserved. The experiments in this section confirm this statement. We have replaced N by the variable X, as shown in Equation 4.3, in our experiments and we have experimented with different values for X. Figures 4.9 and 4.10 show the results for values of X equal to N, $10N$, $5N$, and $0.5N$.

We can see that Spearman's footrule distance and the cosine measure are not affected by the different choices of X, which is an indication that the JXP scores are affected only by a rescaling factor, ad that the ranking order is not altered. The other two plots show the rescaling factor: for $X > N$, the L1-norm shows that the JXP scores will converge to values that are smaller than the true PageRank scores. As a consequence, the linear error score, which compares the JXP scores against the global PageRank scores, remains high throughout the computation. When $X < N$, the scores achieve values that are higher than the true PageRank scores, and their sum converges to a value that is higher than one. The behavior of the linear score error curve can be explained as follows. The scores are initially smaller than the true PageRank scores, and as the scores increase during the execution of the algorithm their values approach the PageRank scores, so there is an initial drop in the curve. However, scores are no longer bounded by the global PageRank values, so as the scores keep increasing beyond the PageRank values, the error curve increases.

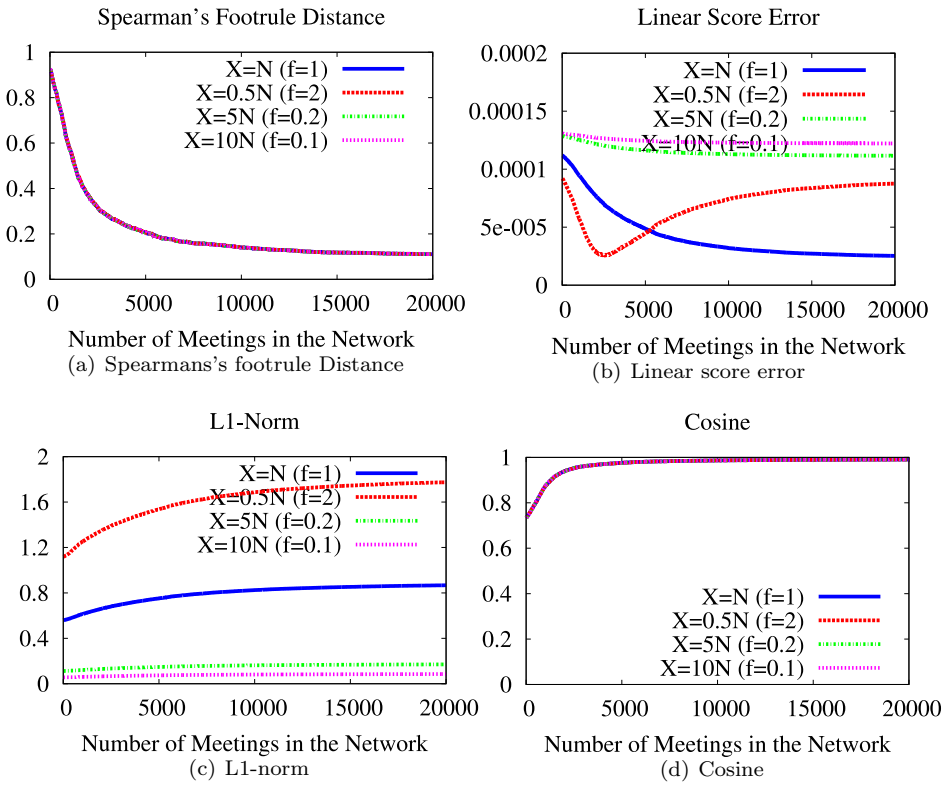


Figure 4.9: Experimental results for X equal to N, $0.5N$, $5N$ and $10N$ for the Amazon dataset.

Scalability

The size of a P2P network, i.e. the number of peers, typically grows and P2P applications have to scale to adapt to changes in the system. We have studied the scalability of the JXP algorithm, by varying the number of peers in the network. We have tested networks with 100, 200, and 500 peers. Results are shown in Figures 4.11 and 4.12. For better comparison of the results, the x-axis shows the average number of meetings per peer, i.e., total number of meetings divided by the number of peers in the network.

The results show that, as the number of peers in the network increases, even though the total number of meetings increases, the average number of meetings a peer has to perform for the same approximation quality does not vary that much, so the computation gracefully scales with the size of the network.

4.6 Applications of JXP Scores

Authority scores have proved to be useful in many centralized applications, in particular in search result ranking. Decentralized authority scores computation allows distributed applications to also benefit from authority scores in a simi-

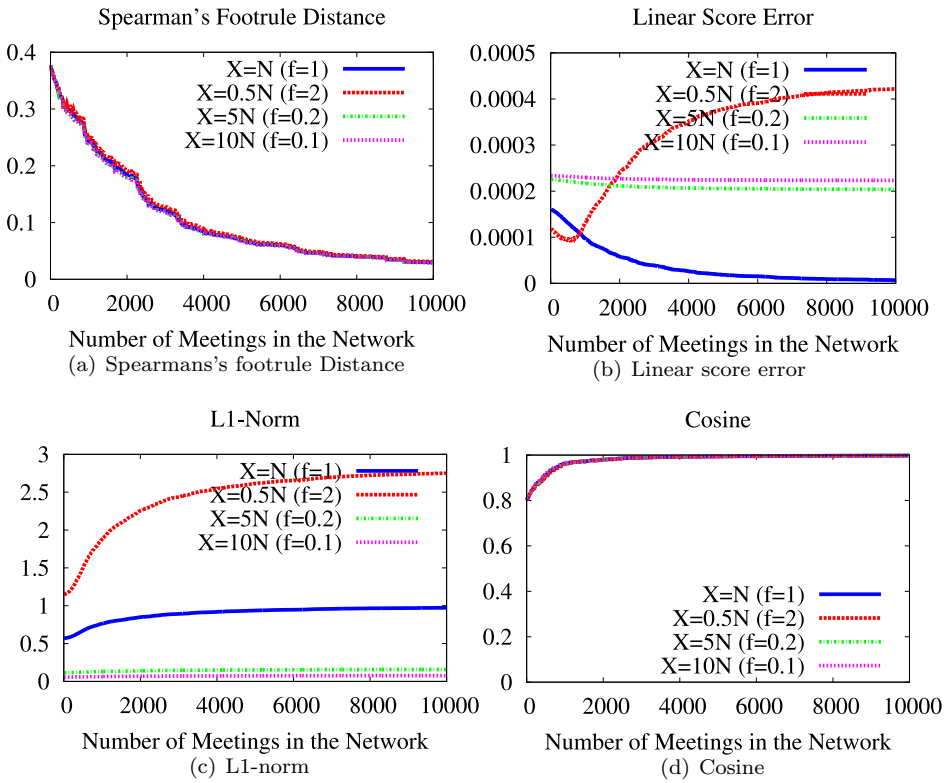


Figure 4.10: Experimental results for X equal to N, $0.5N$, $5N$ and $10N$ for the Web Crawl dataset.

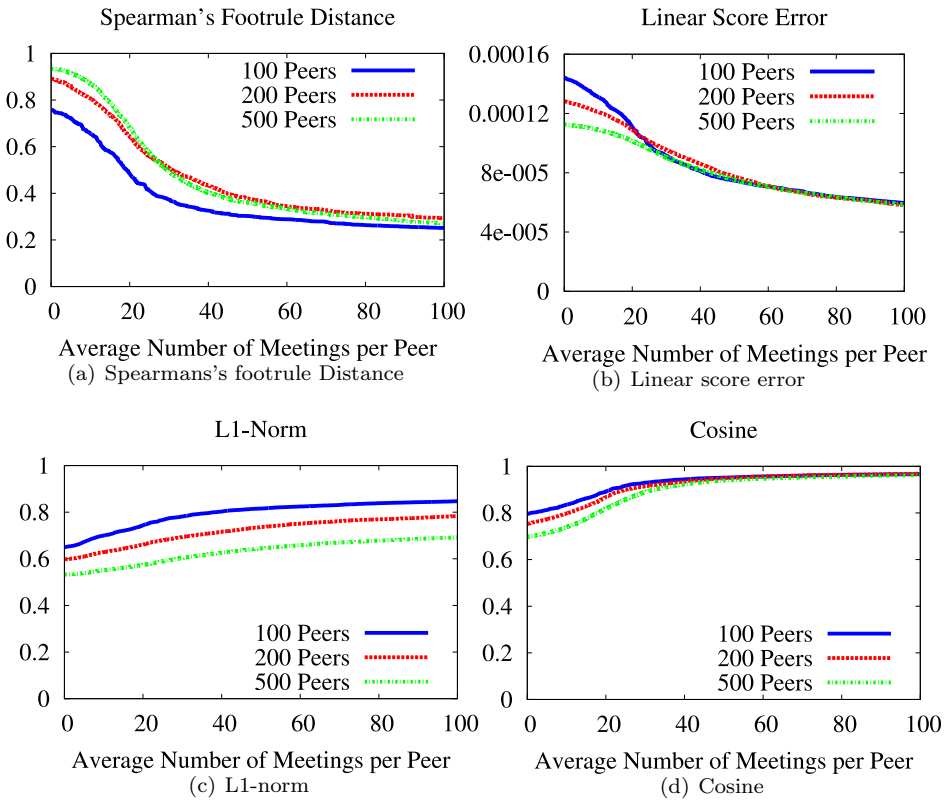
lar way that centralized ones do, and also in applications that are specific for distributed scenarios. We have applied our JXP algorithm in the area of P2P information retrieval and we have chosen the Minerva [BMT+05b, BMWZ05, BMPC07] system as our testbed P2P application.

4.6.1 Minerva

The experiments were performed using Minerva[2], a fully operational distributed search engine [BMT+05b, BMWZ05, BMPC07]. It assumes a P2P collaboration in which every peer is autonomous and has a local index that can be built from the peer's own crawls or imported from external sources and tailored to the user's thematic interest profile. The index contains inverted lists with URLs for Web pages that contain specific keywords.

A conceptually global but physically distributed directory, which is layered on top of a Distributed Hash Table (DHT) (such as CHORD [SMK+01] or Pastry [RD01]), holds compact, aggregated information about the peers' local indexes and only to the extent that the individual peers are willing to disclose.

[2]Project homepage available at http://www.mpi-inf.mpg.de/departments/d5/software/minerva/



(a) Spearmans's footrule Distance
(b) Linear score error
(c) L1-norm
(d) Cosine

Figure 4.11: Performance with different numbers of peers for the Amazon dataset.

Minerva only uses the most basic DHT functionality, *lookup(key)*, that returns the peer currently responsible for *key*. Doing so, the term space is partitioned, such that every peer is responsible for a randomized subset of terms within the global directory. For failure resilience and availability, the entry for a term may be replicated across multiple peers.

Directory maintenance, query routing, and query processing work as follows (see Figure 4.13). In a preliminary step (step 0), every peer publishes a summary (*Post*) about every term in its local index to the directory. A hash function is applied to the term in order to determine the peer currently responsible for this term. This peer maintains a *PeerList* of all postings for this term from peers across the network. Posts contain contact information about the peer who posted this summary together with statistics to calculate IR-style measures for a term (e.g., the size of the inverted list for the term, the maximum average score among the term's inverted list entries, or some other statistical measure). These statistics are used to support the query routing process. The query routing step yields a number of promising peers for the complete query. Subsequently, the query is forwarded to these peers and executed based on their local indexes (*query execution*; step 2). Note that this communication is done

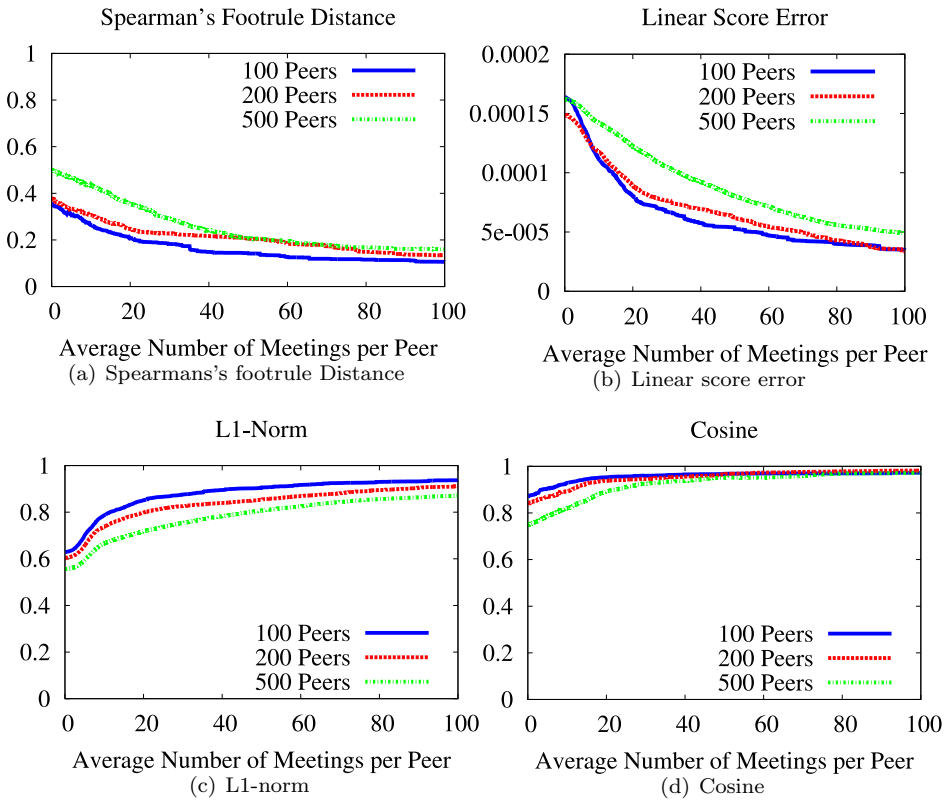


Figure 4.12: Performance with different numbers of peers for the Web Crawl dataset.

in a pairwise point-to-point manner between the peers, allowing efficient communication and limiting the load on the global directory. Finally, the results from the various peers are combined at the querying peer into a single result list. Due to efficiency reasons, the query initiating peer does not have to retrieve the complete PeerLists. Instead, it can run a distributed top-k algorithm to efficiently figure out the k most promising peers.

4.6.2 Improving Results Quality

Here we tested whether the JXP scores can help improve search results quality, as PageRank scores do in centralized approaches. We have performed a simple experiment: we again used the Web Crawl dataset from Section 4.5.1, which contain pages from 10 different topics. We have created 40 peers out of the 10 topics by splitting each topic into 4 fragments. Each of the 40 peers hosts 3 out of 4 fragments from the same topic, thus forming high overlap among same-topic peers. Then we ran 15 queries that are typical for popular Web search requests [BRRT05], using the query routing mechanism of Minerva. The merged results were ranked in two ways: 1) by a standard IR model based on

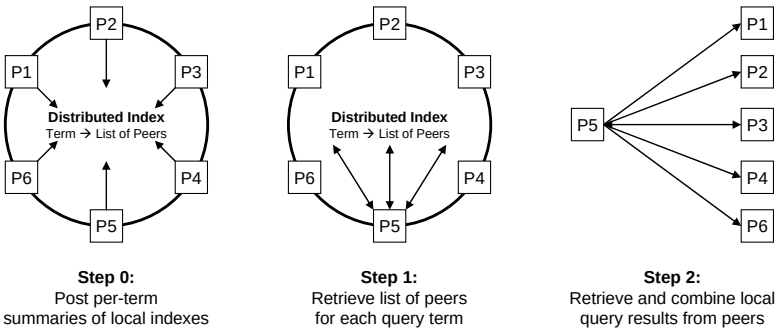


Figure 4.13: Minerva System Architecture.

term frequency (tf) and inverse document frequency (idf) [BYRN99], and 2) by a weighted sum of the $tf*idf$ score and the JXP score (with weight 0.6 of the first component and weight 0.4 of the second component). The queries were taken from [BRRT05] and have been intensively used in prior literature on link analysis. We manually assessed the relevance of the top-10 results under the two different rankings. Given the small size of the collection, we considered pages with links to relevant pages not reached by the crawler also as relevant pages. The results for precision at top-10 are given in Table 4.3. The best results are shown in boldface. On average, the standard $tf*idf$ ranking achieved a precision of 40%, whereas the combined $tf*idf$/JXP ranking was able to increase precision to 57%.

Table 4.3: Precision at top-10 for the Web Crawl Dataset

Query	$tf*idf$	(0.6 $tf*idf$ + 0.4 JXP)
affirmative action	40%	40%
amusement parks	60%	60%
armstrong	20%	**80%**
basketball	20%	**60%**
blues	20%	20%
censorship	**30%**	20%
cheese	40%	**60%**
iraq war	**50%**	30%
jordan	40%	40%
moon landing	**90%**	70%
movies	30%	**100%**
roswell	30%	**70%**
search engines	20%	**60%**
shakespeare	60%	**80%**
table tennis	50%	**70%**
Average	40%	**57%**

4.6.3 Query Routing Strategy Using JXP Scores

The query routing in P2P networks is a well studied problem [CLC95, GGMT99, BMT+05a]. Popular techniques for query routing, such as CORI [CLC95, Cal00], tend to prefer larger peers (i.e., peers containing more pages) over smaller peers, as larger peers are expected to have a higher probability of containing high-quality query results. In the section we present a different approach that, instead of looking the the collections' sizes, looks at the authority scores of pages in the collections. Each peer can be seen as one large page, i.e., an union of all its local pages, similar to what query routing strategies based on statistical language models [SJCO02] do, the query routing process is then equivalent of finding the the top-k "large pages" in the network. Since authority scores are known to greatly improve this process, it seems a natural idea do explore them for query routing as well.

Our idea for improving the query routing process is to prefer peers that have high authority mass, where the authority scores are computed using our JXP algorithm. We have identified two ways of exploring authority scores for query routing purposes: by using the total JXP mass or by a term-specific JXP mass approach.

Total JXP Mass

The total JXP score mass of a peer corresponds to the sum over all JXP scores of local pages. Given a local JXP computation continuously running on a local peer P, the total JXP mass of a collection s_P is calculated as follows:

$$s_P = \sum_{i \in G_P} \alpha_i,$$

where α_i is the JXP score of page i, and G_P is the local collection of peer P.

The total JXP mass however might not be a good indicator of a peer's authority for a particular query; instead, the set of peers with high total JXP mass would always be chosen *regardless of the actual query.* For example, consider the Web page of a researcher that has crawled the publications of leading university departments. While his local Web graph might have a high total JXP mass, the peer is ill-suited to evaluate queries about travel, movies, or music. So the total JXP mass of a collection is not an appropriate measure for the judging the result quality for a particular query. We need a way of using authority scores that is query-dependent.

Term-specific JXP Mass

We can aggregate the JXP scores in a term-based manner, by considering only the scores of pages that contain the term, i.e.,

$$s_P^t = \sum_{\substack{i \in G_P \\ t \in i}} \alpha_i.$$

Query routing based on such term-specific JXP score masses is straightforward: we sum up the term-specific JXP mass for every term in the query,

$$s_P^Q = \sum_{t \in Q} s_P^t,$$

where Q is the list of query terms. Note that term-specific JXP mass does not require a separate JXP computation for every query term, but simply sums up the the regular JXP values for the pages that contain the term at query time. The JXP score of a page will be accounted as many times as the number of query terms the page has.

While the existence of query-specific quality estimators allows for a better query routing approach by summing up only potentially relevant portions of the JXP mass, it assumes term independence, as high score masses regarding terms a and b alone do not guarantee a single high authority page for the combined query (a,b).

Query routing approaches driven by authority scores could also be combined with existing techniques, in the hope to achieve an even better performance. We have chosen CORI, one of the most popular query routing strategies, to devise a hybrid approach.

CORI

CORI is a peer selection strategy proposed by Callan et al. [CLC95, Cal00]. It computes the collection score s_P of the peer P with regard to a query $Q = \{t_1, t_2, ..., t_n\}$ as

$$CORI_P = \sum_{t \in Q} \frac{CORI_{P,\,t}}{|Q|},$$

where

$$CORI_{P,\,t} = \gamma + (1-\gamma) \cdot T_{P,\,t} \cdot I_{P,\,t}.$$

The computations of $T_{P,\,t}$ and $I_{P,\,t}$ use the *number of peers* in the system, denoted np, and the *document frequency* (cdf) of term t in collection G_P for any term t in collection G_P:

$$T_{P,\,t} = \frac{cdf_{G_P,\,t}}{cdf_{G_P,\,t} + 50 + 150 \cdot \frac{|V_P|}{|V^{avg}|}}$$

$$I_{P,\,t} = \frac{log(\frac{np+0.5}{cf_t})}{log(np+1)}$$

where the *collection frequency* cf_t is the number of peers that contain the term t. The value γ is chosen as $\gamma = 0.4$ [CLC95].

CORI considers the size $|V_P|$ of the term space of a peer (i.e., the total number of distinct terms that the peer holds in its local collection) and the average term space size $|V^{avg}|$ over all peers that contain term t. Note that, in the absence of

global knowledge, $|V^{agv}|$ is replaced by the average term space size over all peers that contain term t (see [Cal00]).

Hybrid Approach

CORI mainly focuses on the document frequency of the query terms to select the most promising peers for a query, but it does not take into account the quality of these documents. To overcome this problem, we present a hybrid approach to combine CORI-style quality measure with PageRank-style authority scores for query routing.

We focus on query-specific JXP authority score masses. We suggest the following linear combination to compute s_P^{hyb}, the hybrid collection score of the peer P:

$$s_P^{hyb} = \sum_{t \in Q} \beta * CORI_{P,\,t} + (1-\beta) * s_P^t$$

where $CORI_{P,\,t}$ and s_P^t are the CORI score and the term-specific JXP mass of peer P for term t, respectively.

As extreme cases, $\beta = 1$ results in standard CORI-based query routing, while $\beta = 0$ results in query routing based on term-specific JXP score masses only.

In order to account for the different absolute score values yielded by CORI and JXP, we previously apply the following normalization to all values of $CORI_{P,t}$ and s_P^t, generalized to *score*:

$$\frac{score - min_t(score)}{max_t(score) - min_t(score)}$$

where $min_t(score)$ and $max_t score$ refer to all applicable *score* values regarding term t in the network.

Experiments

We tested our query routing strategies also in the Web Crawl dataset. Given a query, identifying peers belonging to the query topic is a relative easy task, so we focused on a more challenging task, which is to find the best order *within* the peers of the query's topic. For this purpose, we have restricted ourselves to exactly one topic, namely "movies", and distributed only the documents related to movies over a total of 10 peers. With a number of queries related to movies, we proceed with the evaluation of the different strategies regarding the task to discriminate peers that share the same topic. The queries were taken from Google's Zeitgeist archive[3] that match the topic movies, at the time of and slightly prior to acquisition of the dataset. Table 4.4 shows those queries.

For the different number of peers selected, we measure the *relative recall*: given the global ranking formed by the union of all peers' local collections, we

[3] http://www.google.com/press/zeitgeist.html

superbowl commercials	earthquake
harry potter	christopher reeve
julia roberts	angelina jolie
desperate housewives	golden globes
jennifer aniston	academy awards
blockbuster	

Table 4.4: Queries

compute the portion of pages from the top-k positions that were retrieved. More formally,

$$RelativeRecal_{topk} = \frac{|Retrieved_{topk}|}{|Global_{topk}|}, \tag{4.4}$$

where $Retrieved_{topk}$ is the set of pages from the top-k ranking that were retrieved, and $Global_{topk}$ is the set containing the top-k pages from the global ranking. In the experiments k was set to 20. The selected peers locally deploy the same document scoring model that was used on the reference collection, based on standard $tf*idf$ document scores.

We compare the following instances of our hybrid framework:

- $\beta = 1$: standard CORI
- $\beta = 0.5$, $\beta = 0.1$: hybrid strategies
- $\beta = 0$: term-specific JXP masses only

Figure 4.14 plots the relative recall for an increasing number of peers selected by the different query routing strategies. The *optimal* curve shows a theoretical result where, for each query, we precomputed the relevant pages in each collection and query routing was based on an ascending order of relevant pages. Both the hybrid strategy and our strategy based on term-specific JXP score masses outperform the baseline, CORI, in terms of relative recall, in particular for a small number of peers. This is crucial, because the ultimate goal of query routing is to achieve good recall with a very small number of peers. The fact that quality-unaware query routing based on PageRank authority scores only performs as good as our hybrid strategy is an artifact of our small-scale experimental setup. Even though this is a small-scale experiment, this gives first evidence of proof for our hypothesis that authority score masses can be a helpful ingredient in discriminating peers for query routing.

4.7 Discussion

JXP is an algorithm that computes an approximation of PageRank scores of pages distributed in a P2P network in an efficient and scalable manner, while preserving the autonomy of peers. It runs at every peer, and works by combining locally computed PageRank scores with meetings among the peers in the

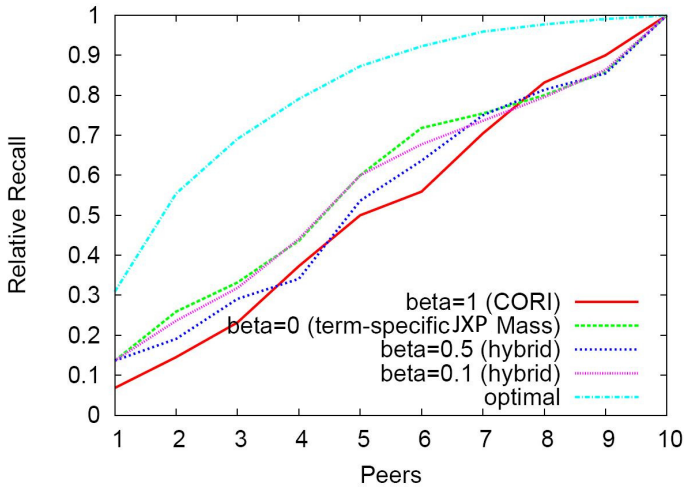


Figure 4.14: Relative Recall Performance

network. Meeting are asynchronous and the local data collections can overlap. Through experiments as well as theoretical arguments we showed that the JXP scores converge to the true PageRank scores that one would obtain by a centralized computation. The algorithm is versatile and could be easily adapted to compute other kinds of authority and trust measures that are based on principal Eigenvectors in some form of social network.

A salient property of JXP is its scalability: regardless of how large the network becomes, the storage and computational costs per peer are limited by the (order of the) resource commitments that the peer has made for hosting its local data collection and graph fragment anyway. Also, the messaging costs for peer meetings are very small. Experimental results, with two different datasets and systematic variation of setups, have confirmed the anticipated properties of JXP: convergence to global PageRank values and low computational costs. In addition, we have shown the benefits of the JXP scores in the Minerva distributed Web search engine, in the tasks of query routing and results ranking.

Other aspects of P2P networks, like dynamics and susceptibility to malicious behavior will be addressed in subsequent chapters.

Chapter 5

TrustJXP: JXP in Untrustful Networks

The open and anonymous nature of P2P networks, which is one of the main advantages over client-server approaches, is also one of the main issues faced when designing P2P applications since it opens the door to abuses of these networks by malicious peers.

According to [MGM06], the two primary types of adversaries in P2P networks are *selfish* peers and *malicious* peers. These two behaviors differ mainly by their goal in the system. Selfish peers want to use the network services without contributing resources (or only minimal contribution). A well-known example of selfish peers is the so-called "free-riders" in file sharing networks [AH00], like Kazaa and Gnutella, that refuse to host files to reduce their cost in bandwidth and CPU utilization.

Malicious peers, on the other hand, aim at causing harm to some network members or to the whole network, and are willing to spend time and resources to achieve their goal. An example of malicious behavior is the distribution of corrupted or virus-infected files to discourage piracy or to gain notoriety in the network. In the context of page authority computation, malicious peers would try to distort the correctness of the computation, by providing different (usually higher) scores for their local pages. Having pages with high authority scores can bring many benefits for the peer: with its pages appearing at the top positions in the ranking for answering queries posted to the network, the probability that a user clicks on one of them is higher, which may translate, for instance, in revenue for that peer.

In general, P2P networks are vulnerable to selfish/malicious behaviors and need *reputation systems* [MGM06] in place to be able to operate properly. Since peers can behave badly in many different ways, the usual approach when designing reputation systems is to consider each type of adversary at a time.

5.1 The TrustJXP algorithm

TrustJXP integrates the JXP algorithm for decentralized authority scoring with an equally decentralized reputation system, for computing more reliable authority scores. The approach is based on anomaly detection techniques, that allow the detection of a suspicious peer based on the deviation of its behavior from some common features that constitute the usual peer profile. It combines an analysis of the authority score distribution and a comparison of rankings for a small set of pages.

The algorithm is completely decentralized, does not require storing any additional information about other peers, can operate anonymously, and involves only local computations. Also, TrustJXP does not require any form of cooperation among peers, and the system works as long as the fraction of well behaving peers is significantly larger than the fraction of cheating peers.

Next we describe in detail what types of adversarial behaviors are considered and how they are addressed, and also how the TrustJXP scores are computed.

5.1.1 Adversarial Behaviors

There are many possible forms of attacks or manipulations in a P2P network. In this work we focus on the group of attacks where peers want to distort the authority scores being computed by JXP, by reporting false scores for a set of pages at the meeting phase. We have modeled two general types of attack:

1. A cheating peer can report a higher score for a subset of its local pages, in an attempt to get its pages into high positions in the global ranking that JXP peers may perceive. In this form of manipulation, the peer would boost pages at the "expense" of reducing the total weight of its world node (giving lower score mass to all non-local pages).

2. A cheating peer can manipulate the scores of its local pages by modifying the scores, not necessarily increasing them. This way, some pages are boosted while others are downgraded. The score mass of the world node would stay unchanged. If the cheating peer wants to maintain the statistical distribution of the scores among local pages, it can just permute the scores of its local pages.

How to detect and eliminate or compensate the effects of these two forms of attack, or even from combined attacks that use both techniques, is explained next.

Malicious Increase of Scores

To combat this kind of manipulation we use the scores distribution of the pages in a peer's local graph. After a few iterations, the local distribution should resemble the global distribution. The justification for this hypothesis stems from the way the local graph fragments are built. In our P2P model, each peer

gathers its data by performing Web crawls, starting from particular seeds and possibly using a thematically focused crawler in order to harvest pages that fit with the interest profile of the corresponding user (or user group). Given that the Web graph is self-similar [DKM+02, BCDF06], the properties of the small graph fragment that a peer eventually compiles should be statistically indistinguishable from the properties of the full Web graph as seen by a centralized crawler. [DKM+02] observed these properties also across different partitions of the Web graph, including the case where pages were separated by their content, which corresponds to using a focused crawler.

Storing the Typical Profile

A representation of the distribution of the scores in the network is kept in histograms. Pages are assigned to histogram buckets according to their JXP scores. Since scores are expected to follow a power-law distribution, we make the boundaries of the buckets also exponential, similar to what is used in [BCSU05]. More precisely, the bucket number i will have the boundaries

$$bucket(i) = [a \cdot b^{i-1}, a \cdot b^{i}).$$

The precise values for a and b will depend on the distribution of PageRank values in the observed sample, which in turn depends basically on the number of pages in the entire network and the dampening factor for PageRank. The dampening factor for the computation is shared among all the nodes. The number of pages (at least its order of magnitude) can be initialized with an estimation that can be improved after a few meetings. The choice of the buckets is not relevant, as long as not all the pages fall in the same bucket. It is not necessary that all peers use the same buckets, and in the worst case, a peer can re-initialize its histograms with new parameters at any time (at the cost of slowing down its convergence).

We create, at each peer, a histogram which is initially filled with the initial JXP scores of local pages. After each meeting, the distribution of the local scores of the other peers is added to the histogram. We introduce a *novelty factor* to account for the dynamics of the scores across the meetings. Given the histogram at meeting t, H^t, and the score distribution from the other peer D, the histogram at meeting $(t+1)$ is updated as follows:

$$H^{(t+1)} = (1-\rho)H^t + \rho D,$$

where the parameter ρ represents how much importance we give to the new values, and the precise choice only affects how fast the peer learns the global distribution of scores, which in turn changes the convergence speed for the scores in that particular peer.

Since we rely on the assumption that the number of honest peers is significantly bigger than the number of dishonest ones, we expect that the histogram always reflects the true distribution of the honest peers. If dishonest peers are

reporting higher scores for some of their local pages, the distribution of their local scores would no longer resemble the distribution expected over all peers. Therefore, a comparison against the accumulated local histogram should give an indication of this deviation from normal behavior.

Comparing Histograms

Given the accumulated histogram of a peer i, H_i, and the histogram containing the scores distribution of another peer j, D_j, we want to compute how much D_j deviates from H_i. Since the distributions are expected to be similar [DKM+02], we believe that the distributions of honest peers should be very close to each other, and if D_j differs from H_i by a large margin, it is an indication that the peer is cheating about its local scores. For comparing the two distributions we have chosen the *Hellinger Distance*, which is defined as [Cam86]:

$$HD_{i,j} = \frac{1}{\sqrt{2}}[\sum_k (\sqrt{H_i(k)} - \sqrt{D_j(k)})^2]^{\frac{1}{2}},$$

where k is the total number of buckets, and $H_i(k)$ and $D_j(k)$ are the number of elements at bucket k at the two distributions, both normalized by the total number of elements at each distribution. The factor $1/\sqrt{2}$ is introduced to normalize the range of possible values.

As an alternative to the Hellinger Distance, we could also use the χ^2 goodness-of-fit test or information-theoretic measures such as Kullback-Leibler divergence. Our choice for the Hellinger Distance was mainly due to the fact that, since it is a metric, the Hellinger Distance has nice properties, besides the fact that values can be normalized, which makes it easier to be combined with other measures.

Malicious Permutation of Scores

The histograms comparison is inherently unable to detect a cheating peer that reports a permutation of the current scores of its local pages, since both distributions would be statistically indistinguishable. For detecting this type of attack we use a different technique. In our experimental studies of the JXP algorithm, we have observed that, after a few meetings, although the local JXP scores do not correspond yet to the global authority scores, the relative rank orderings of their local pages are already very close to the final ordering. This is also exploited in [XL04b] for a different task (testing if a feedback given by a peer makes sense).

We compare the rankings given by the two peers in a meeting for those pages that fall into the overlap of both local graphs, and we measure what we refer to as the *Tolerant Kendall's Tau Distance* between those rankings. We use a relaxation of Kendall's Tau since we need to tolerate small fluctuations in the scores of pages with almost identical global authority. To this end, we discount page pairs that have different relative orders in the two rankings if their score

differences are below a tunable threshold Δ. In this case, we consider the page pair as incomparable and their rank order as arbitrary.

Our *Tolerant Kendall's Tau Distance* is therefore defined as:

$$K'_{i,j} = |(a,b) : (a < b) \wedge (|score_i(a) - score_i(b)| \geq \Delta \vee |score_j(a) - score_j(b)| \geq \Delta) \wedge ((\tau_i(a) < \tau_i(b) \wedge \tau_j(a) > \tau_j(b)) \vee (\tau_i(a) > \tau_i(b) \wedge \tau_j(a) < \tau_j(b)))|,$$

where $score_i(a)$ and $score_i(b)$ are the scores of pages a and b at peer i, $a < b$ refers to the lexicographical order of page URLs (to avoid double-counting), τ_i and τ_j are the rankings of pages in the overlapping set at peers i and j, and Δ is our tolerance threshold. A good choice of Δ can be derived from the dampening factor of the underlying PageRank model as follows. We consider as our threshold the minimum amount of authority mass one page can have, which is the score mass earned from the random jumps. Therefore, at each peer, Δ is set to

$$\Delta = \frac{(1-\varepsilon)}{N},$$

where ε is usually set to 0.85 and N is the total number of pages in the network.

This approach assumes that whenever two peers meet, there is a sufficient overlap between their locally known pages to make this comparison statistically meaningful. In an application where such overlaps cannot be guaranteed with high probability, we would have to add artificial overlaps as "honesty witnesses". One way of designing such an additional set of witness pages would be to randomly draw a set of sample URLs and disseminate them in the network by an epidemic protocol or using the overlay network of the P2P system. This set of witnesses should be changed periodically to counter adaptation strategies of malicious peers.

5.1.2 Assigning Trust Scores to Peers

We now use our reputation system to assign trust scores to peers. The method is totally decentralized: each peer is responsible for assigning (its perception of) trust scores to other peers, based on interactions with them. During a meeting, peers exchange the scores of their local pages. These scores are used for computing both histograms divergence and the rank divergence for the overlapping pages. These two measures determine the level of trust that should be given to the peer. A new trust score is assigned to a peer at every meeting, as scores are changing.

For combining histograms divergence and rank divergence into one single trust score, we take a conservative choice: we always take the lower level of trust among the two measures. Thus, we define the trust score that a peer i gives to a peer j as

$$\theta_{i,j} = min(1 - HD_{i,j}, 1 - K'_{i,j}).$$

This is the trust score that will be used in the TrustJXP algorithm for computing more reliable authority scores.

5.1.3 TrustJXP Authority Scores Computation

The idea of TrustJXP is to incorporate the trust measure θ into the JXP algorithm for computing more reliable and robust authority scores by using the trust measure at peer meetings when adding the information in the world node. When updating the world node, in the original JXP algorithm, if a page is already represented, its score will be set to the maximum between the current score and the score received by the other peer (see Section 4.1.2). As state earlier

$$\alpha_j^t = max(\alpha_j^{t-1}, \alpha_j^{msg}),$$

where α_j^t and α_j^{t-1} are the scores of page j (stored in the world node) at the current meeting and at the previous meeting, respectively, and α_j^{msg} is the score of j in the message received during the meeting. α_j^{t-1} is zero if the world node does not contain the page.

For the TrustJXP algorithm, the contribution of the scores from the other peer are weighted based on how much that peer is considered to be trustworthy. The score of a page j in the world node is now defined as

$$\alpha_j^t = max(\alpha_j^{t-1}, \theta * \alpha_j^{msg}).$$

After updating the world node, the TrustJXP algorithm proceeds as in the JXP algorithm: the transition probabilities from the world node are updated, and a PageRank computation is performed, leading to new authority scores.

5.2 Experimental Evaluation

5.2.1 Setup

The experiments were conducted on the same Web Crawl collection used in Chapter 4. We created a set of 100 peers that used the same crawling strategy described in the previous chapter. In our setup, these 100 peers will correspond to the trustful peers and each one will hold its full graph fragment that was assigned to it. Thus, in the absence of malicious peers, our authority scores can converge to the global PageRank scores of the complete graph.

The different fractions of malicious peers were introduced into the system. Their sets of local pages are subsets of the collections held by honest peers. Malicious peers perform meetings and local PageRank computations like any normal peer. The difference is that, when asked by another peers for a scores vector, a malicious peers will lie about the scores of its local pages, according to one of the possible cheating behaviors.

The boundaries of the histograms' buckets were defined as

$$bucket(i) = [0.005 \cdot 0.3^{i-1}, 0.005 \cdot 0.3^{i}),$$

and when updating histograms the novelty factor (ρ) was set to 0.6.

5.2.2 Cheating Behaviors

Each of the malicious peers picks one of the following attacks:

- Report local authority scores that are higher than the true values for all of their local pages. The exact value is set at each experiment.
- Report these falsely boosted scores for only half of their local pages (drawn randomly but used consistently throughout all meetings).
- Report a permuted scores list (with a consistent permutation, otherwise it could be easily detected by two successive meetings).

In the experiments, peers do not change their behavior during the TrustJXP computation; for example, if a peer chooses to permute its scores for the first meeting, it will do so for all subsequent meetings and it will apply always the same permutation.

5.2.3 Performance Metrics

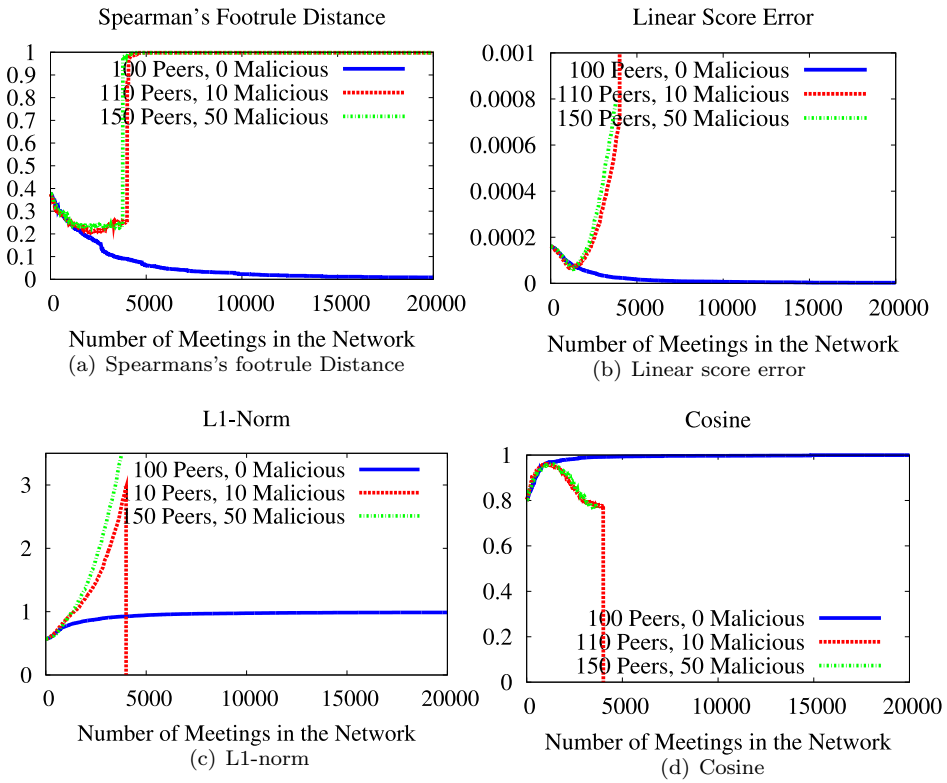
We have used the same four metrics defined in Chapter 4: Spearman's footrule distance and Linear score error over the top-1000 pages, plus L1 norm for the TrustJXP ranking vector and the cosine similarity between the vectors with TrustJXP and global PageRank scores. In addition, for some experiments, we also report the values for the Hellinger Distance and the Tolerant Kendall's Tau.

5.2.4 Results

Effect of malicious peers in JXP

We have first analyzed the impact of cheating peers in the JXP algorithm. Starting with the 100 honest peers, we first introduced 10 cheating peers. Each of these 10 peers uses one of the possible attacks by uniformly random choice (i.e., with each one of the three types of adversarial behavior having probability 1/3 to be chosen by a dishonest peer). The values of the increased scores are twice as high as the true values. Keeping this setup of mixing behavior, we then increased the number of dishonest peers form 10 to 50. The results of are shown in Figure 5.1.

We clearly see that, with the introduction of malicious peers and without any defense mechanism, the JXP scores do no longer converge to the true global PageRank values. The mathematical analysis of the JXP algorithm given in 4.2 proved that the JXP scores are upper-bounded by the true PageRank scores. With malicious peers reporting scores that are higher than the true ones, there is no bound for the scores. This effect can escalate: it distorts the world node



(a) Spearmans's footrule Distance (b) Linear score error (c) L1-norm (d) Cosine

Figure 5.1: Impact of malicious peers in JXP.

score and the transition probabilities from the world node to the local pages, and can even lead to a negative transition probability for the word node's self loop. At this point, scores start becoming undefined. At this point, the linear score error, cosine, and L1-norm curves start behaving oddly, until they eventually became undefined.

Effect of malicious peers in TrustJXP

We proceeded by testing our trust model, measuring both histograms divergence and rank divergence for the overlapping pages. We again introduced 50 cheating peers, but now all peers performed the same type of attack. Figure 5.2 shows the Hellinger Distance and the Tolerant Kendall's Tau for the case where cheating peers report scores five times higher than the true ones, and for the case where peers permute their scores, respectively.

The results confirm our hypothesis that comparing histograms can be an effective indicator of cheating behavior with increased scores. We can also see that, when scores are permuted, the histogram approach does no longer work, and the rank divergence provides a better indication of such malicious behavior.

We then repeated the experiment with 50 malicious peers and the random choice of attack types (again, peers report five times higher values than the true

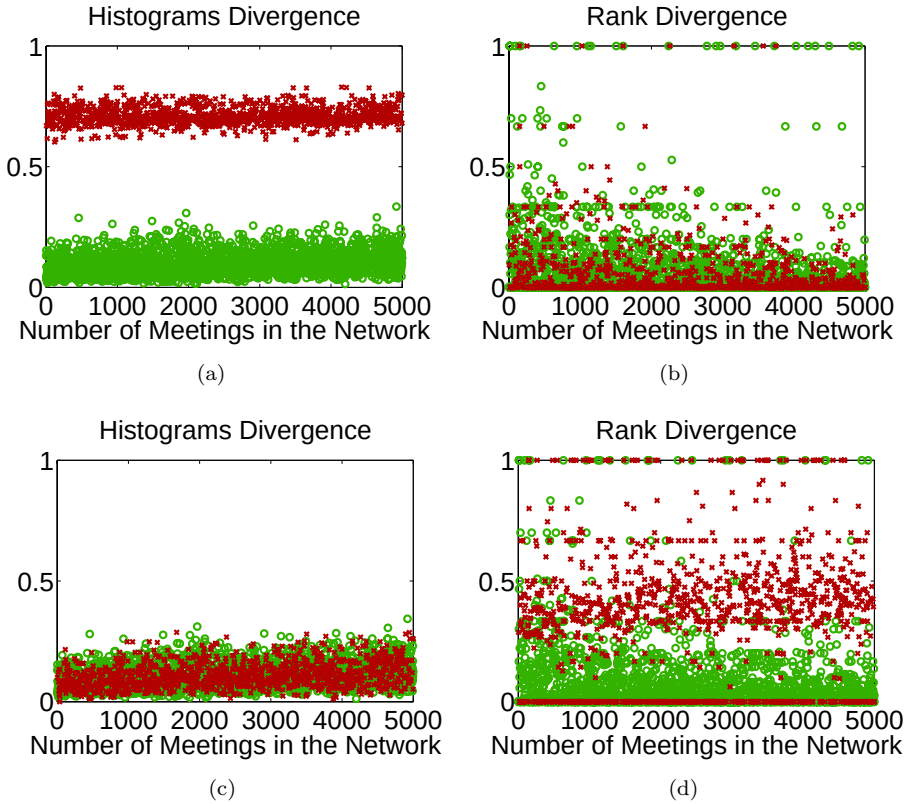


Figure 5.2: Increased-scores attack: (a) histogram divergence (b) rank divergence. Permuted-scores attack: (c) histogram divergence (d) rank divergence. A circle (∘) represents a meeting between two honest peers, and a cross (×) a meeting between an honest and a dishonest peers. Meetings between two dishonest peers are not shown for clarity.

ones for the increased scores attack), and used our new TrustJXP method for computing local scores. The histograms and rank divergence, as well as the final TrustJXP scores are shown in Figure 5.3.

We can see that the histogram divergence is already able to detect many dishonest peers (the ones with higher values), but there are still some peers whose malicious behavior can not be detected. The same happens with the ranking divergence. However, both measures combined leads to a much better malicious behavior detection, as the majority of peers with high trust scores are indeed trustful.

For comparison on how effective a trust model could be, we also simulated a best case, with an oracle-based defense mechanism that knows the class of each peer (honest vs. cheating) beforehand. The results for TrustJXP versus JXP and the oracle-based system are shown in Figure 5.4.

For most of the metrics, our TrustJXP method is fairly close to the ideal case

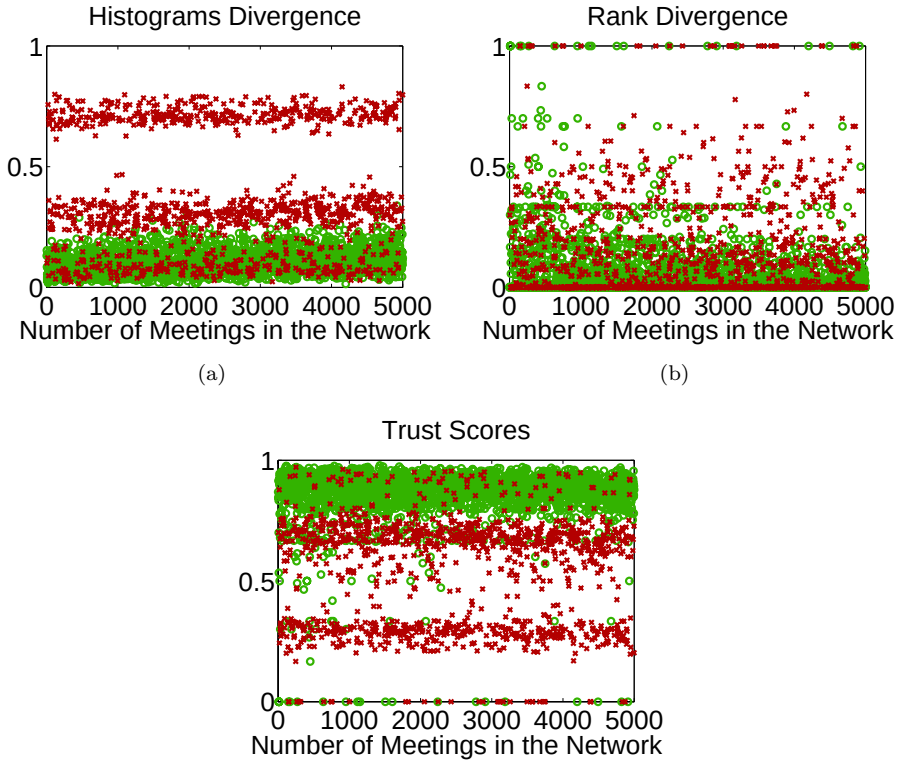


Figure 5.3: Random choice of forms of attack: (a) histograms divergence (b) rank divergence (c) trust scores.

in terms of detecting and compensating malicious peers. In the figure we can also see that the original JXP algorithm cannot counter the effect of malicious peers and it quickly degrades.

In addition, for the trust scores shown in Figure 5.3 (d), we measured, for a given threshold θ, the percentage of honest and dishonest peers that have received a trust scores higher and equal to θ, i.e., if there are 100 honest peers and 90 of them received a trust score of at least 0.5, then the percentage of honest peers for $\theta = 0.5$ is 90%. Analogously, if there are 20 malicious peers and 5 of them have a trust score higher or equal to 0.5, then the percentage of dishonest peers for $\theta = 0.5$ is 25%.

In Table 5.1, we show the percentage of honest and dishonest peers for three different values of θ.

These values could be used in a (hypothetical) system in which the trust score is measured and a meeting is rejected whenever the other peer's trust value is below the threshold. We can see that for $\theta = 0.8$, the percentage of honest and dishonest peers are 86.9% and 12.1%, respectively. This means that, by using this threshold, it would be possible to recognize and and discard 87.9% of

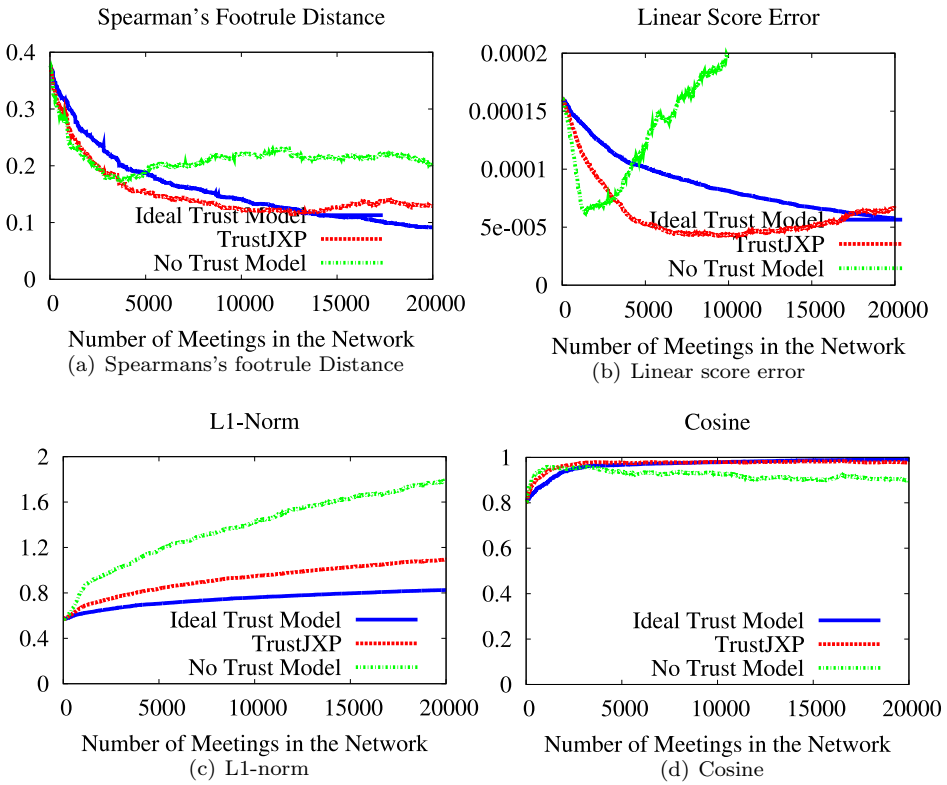


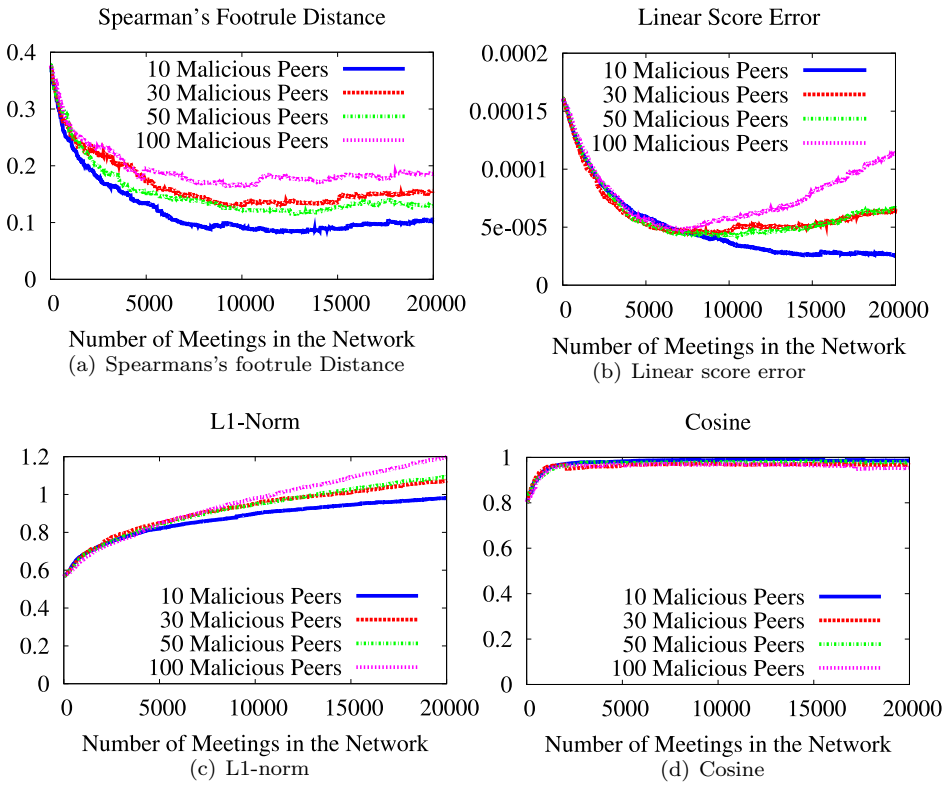
Figure 5.4: Impact of malicious peers with TrustJXP.

θ	% of honest	% of dishonest
0.9	37.4%	4.7%
0.8	86.9%	12.1%
0.6	98.0%	54.5%

Table 5.1: Percentage of honest and dishonest peers for different values of θ.

malicious peers, while loosing the information from only 13.1% of honest peers.

Even though we have shown that TrustJXP is a good contribution for the problem of detecting malicious behavior, it is by no means sufficient in this task, since the contributions from malicious peers are still accounted for, although with lower weight. Figure 5.5 shows the algorithm's performance for different numbers of bad peers in the network. The number of good peers is fixed and equals 100. We can see that, as the number of bad peers increases, TrustJXP becomes less effective in detecting all malicious behaviors. However, even with a high number of malicious peers, the algorithm is able to slow down the effects of the attacks.



(a) Spearmans's footrule Distance
(b) Linear score error
(c) L1-norm
(d) Cosine

Figure 5.5: Impact of malicious peers with TrustJXP.

5.3 Discussion

Having pages with high authority scores can bring many benefits for a peer: with its pages appearing at the first positions in the ranking for answering queries posted on the network, the probability that a user clicks on one of them is higher, which may translate, for instance, in revenue for that peer. Therefore some peers might try to manipulate the scores computation, in order to get higher scores to their local pages.

The TrustJXP algorithm is an attempt to reduce the impact of malicious peers in our JXP algorithm for decentralized authority computation. It integrates the JXP algorithm with a reputation systems designed specifically for detecting such types of adversarial behavior. The reputation system combines an analysis of the authority score distribution and a comparison of rankings from a small set of pages. It relies on the assumptions that score distributions at all honest peers should look similar, given that the Web graph is self-similar, and that there is sufficient overlap among peers' local graphs. In cases where these assumptions do not hold, honest peers might be punished, slowing down the scores convergence but staying conservative.

Experiments have demonstrated the viability and robustness of our method.

For example, we showed the normal JXP system can withstand a population of 10% of malicious peers using the described attack models, but not a population of 33%. We have seen that TrustJXP can work with such a high number of malicious peers. For bigger populations, we showed that the algorithm becomes less effective, but it is still able to slow down malicious effects.

Chapter 6

JXP under P2P Dynamics

One of the main characteristics of P2P networks is their dynamic nature. Peers are constantly joining and leaving the network, meaning that the fully content is not always available. Moreover, peers might change what they store, for instance a user can become interested in a different topic and start to store information about this new topic instead. We can distinguish dynamics into two types: network dynamics and content dynamics. Network dynamics refers to changes on the peer population since peers are continuously joining and leaving the system. Content dynamics refers to changes on what is stored by the peers.

In previous chapters we have presented the JXP algorithm for decentralized computation of global PageRank scores in a P2P network. JXP has potential limitations, namely, it assumes that (i) the global size of the graph is known, and (ii) peers and their contents are static throughout the entire computation. In Chapter 4 we have addressed (i), showing that a wrong estimation of global size causes only a rescaling of the JXP scores, while the ranking order is preserved. For convergence to the true PageRank scores however, the correct graph size is needed.

In case of peer dynamics only, i.e., the Web graph is fixed and peers are constantly leaving and eventually joining the network again, the convergence guarantees given in Chapter 4 still hold, with the difference that the convergence is slowed down, given that some peers are not accessible for a certain period. Dealing with content dynamics, i.e., pages being added to the network or becoming unavailable, gives a more realistic model, and is discussed in this chapter.

We present an approach on how to estimate the total number of distinct pages in the network. Then we proceed on explaining how JXP can be adapted to handle dynamics.

6.1 Estimating the Global Number of Pages

As mentioned earlier, convergence to the true PageRank values requires the knowledge of the total number of pages in the network. In this section we

propose a method for computing this value in a dynamic P2P network.

Our approach works as follows: instead of a single value, peers initialize a hash sketch [FM85] that represents the set of local pages. During a meeting, peers exchange the hash sketches and the local copy is updated by taking the union of both sketches (local and from the peer met). What we aim at is to have the hash sketches at all peers to be the same and equal to the sketch that represents the union of all local sets. The size of the global graph, can then be estimated at each peer, with error bounds given by the hash sketch construction. This gossiping algorithm can be adapted to content dynamics using a sliding window approach.

6.1.1 Hash Sketches

Hash sketches were first proposed by Flajolet and Martin in [FM85] to probabilistically estimate the cardinality of a multiset S. Hash sketches rely on the existence of a pseudo-uniform hash function $h() : S \rightarrow [0,1,\ldots,2^L)$. Durand and Flajolet presented a similar algorithm in [DF03] (*super-LogLog counting*) which reduced the space complexity and relaxed the required statistical properties of the hash function.

Hash sketches work as follows: let $\rho(y) : [0,2^L) \rightarrow [0,L)$ be the position of the least significant (leftmost) 1-bit in the binary representation of y, that is,

$$\rho(y) = \{\min_{k \geq 0} bit(y,k) \neq 0\},\ y > 0,$$

and $\rho(0) = L$. $bit(y,k)$ denotes the k-th bit in the binary representation of y (bit-position 0 corresponds to the least significant bit). In order to estimate the number N of distinct elements in a multiset S we apply $\rho(h(s))$ to all $s \in S$ and record the least-significant 1-bit in a bitmap vector $B[0\ldots L-1]$. Since $h()$ distributes values uniformly over $[0,2^L)$, it follows that

$$P(\rho(h(s)) = k) = 2^{-k-1}.$$

Thus, when counting elements in an N-item multi set, $B[0]$ will be set to 1 approximately $\frac{N}{2}$ times, $B[1]$ approximately $\frac{N}{4}$ times, etc. Then, the quantity

$$R(S) = max_{s \in S} \rho(h(s))$$

provides an estimation of the value of $\log_2 N$. An example showing how to compute the bitmap vector B and how to use B to estimate the number of elements is shown in Figure 6.1.

The estimator above has an additive bias of 1.33 and a standard deviation of 1.87. To improve it the authors in [FM85, DF03] present techniques that use multiple bitmap vectors (B), instead of only one. In more detail, they use a set of $m = 2^c$ bitmap vectors. Then for each element in the set S, the first c bits of $h(s)$ are used to select each vector the element will be inserted into, and the remaining bits if $h(s)$ are used to update the selected vector. The set of

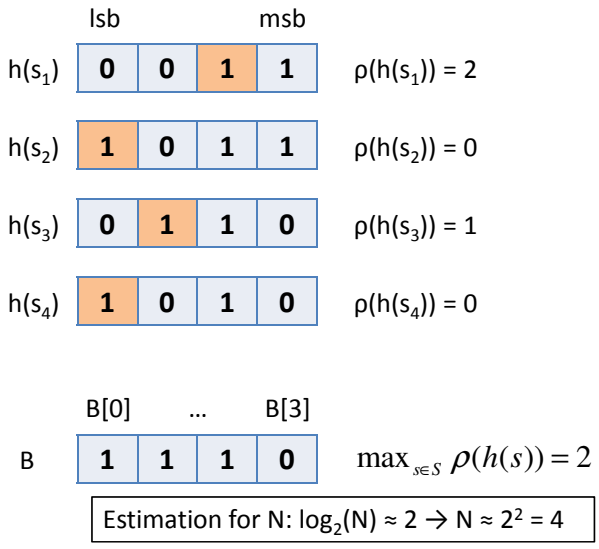


Figure 6.1: Example of a Hash Sketch.

bitmap vectors is used in different ways by the authors in [FM85, DF03]. We have chosen the estimator by [FM85], which is given by

$$E(N) = \frac{1}{0.77351}\beta 2^{\frac{1}{\beta}\Sigma_0^{(\beta-1)}M^i},$$

where M^i is the position of the leftmost 0-bits in the i^{th} bitmap. The bias and standard error of this estimator are closely approximated by $1+0.31/\beta$ and $0.78/\sqrt{\beta}$, respectively [FM85].

One of the main advantages of using hash sketches is that they offer duplicate elimination "for free", or in other words, they allow counting distinct elements in multi sets. Estimating the number of distinct elements (e.g., pages) of the *union* of an arbitrary number of multi sets (e.g., distributed and autonomous collections) — each represented by a hash sketch synopsis — is easy by design: a simple bit-wise *OR*-operation over all synopses yields a hash sketch for the combined collection that instantly allows us to estimate the number of distinct elements of the combined collection.

6.1.2 Estimating Global Counts Using Hash Sketches

In the task of estimating global counts using hash sketches, peers can benefit from the counts of all the other peers, due to the duplicate aware counting. To make the analysis tractable, lets assume for now that all peers perform their meetings in a synchronized way, i.e., after some amount of time, all peers have performed the same number of meetings. Consider one particular peer that is about to perform its m^{th} meeting. Therefore, it has already performed $m-1$ meetings in the past, and the peer it will meet is also in its m^{th} meeting. By

transitive effects (the met peer having met other peers earlier), both peers now double the amount of meetings they are aware of (recorded in hash sketches). We denote by $C(m)$ the number of meetings a peer is aware of after the m^{th} meeting. In the synchronized case we can also write $C(m) = 2^{(m-1)}$, i.e., the number of meetings a peer is aware of grows exponentially with the number of meetings the peer has performed. From a single peer's point of view, after having performed m meetings the situation is identical with having had $C(m)$ meetings where peers do not share information about their previous meetings.

Charikar et al. [CCMN00] consider the problem of estimating the number of distinct values in a column of a table. The difference to our scenario is that in a database table, the number of tuples is known, whereas in a truly distributed large scale system, the total number of peers is unknown. In addition, we know only how many peers or pages we have seen so far, and not the frequency of observation. In practice, all we have is an estimate of distinct values given the sampling using meetings and the exchanged hash sketches, thus we cannot directly apply the estimators from [CCMN00]. However, the estimation of the number of distinct items in a multi set is a well studied problem (cf., e.g., [LP56]). In [LP56] the authors show that, for a set that contains N distinct elements, if a sample of size x is taken from the set, the expected number of distinct elements k, $k \leq N$, observed in the sample is given by

$$E[k] \cong N(1 - e^{-x/N}).$$

In our scenario, the sample size is the number of pages seen after m meetings, $C(m)$, therefore we can write

$$\frac{C(m)}{N} = ln(\frac{N}{N - E[k]}),$$

which can be used to get an estimator $\hat{N}$ of the total number of distinct elements N. The variance of the estimator can be obtained from the fact that the probability of seeing exactly k distinct elements in the sample is a likelihood function ([LP56]), and it is given by

$$\sigma^2_{\hat{N}} = \frac{N}{e^{C(m)/N} - (1 + \frac{C(m)}{N})}.$$

Hence, to reach negligible error even for big values of N, we need only few rounds of peer meeting since $C(m)$ grows exponentially.

In practice we do not know the value of $C(m)$ since peers meet asynchronously and the online time of peers largely varies. In addition, we are not aware of N, the total distinct number of pages in the system. We have only an estimate given by the hash sketch based sampling. The reasoning presented above shows, however, that few iterations are needed to get to a meaningful hash sketch. That does not include any reasoning about the quality of hash sketches which is given in the original work by Flajolet et al. [FM85] and is thus orthogonal to our goals.

The approach for estimating the number of pages needs to be adapted for dealing with dynamics in the system, with pages being inserted or removed from

the network. The former case is handled by the estimator introduced above. The latter case requires some further improvements. Since one can easily add items to a hash sketch but one cannot remove items from such a sketch, we employ the usage of a time sliding window over multiple hash sketches. We let each peer keep an array of k hash sketches, ordered by time, the k^{th} hash sketch is considered to be the "oldest" one. After τ time steps we remove the oldest sketch and insert an empty one at array position 1. Newly observed pages will always be inserted into the sketch at position 1. At any time, the current estimate of distinct pages is the estimate derived from the hash sketch created by forming the union of all k sketches.

6.2 Adapting JXP for Dynamics

Recalling the previous JXP meeting procedure, a peer selects another peer for a meeting and contacts this peer. The contacted peer then returns the information that is relevant to the peer initiator. Due to possible overlaps and the asynchronous nature of the algorithm, different peers might provide different score values for the same page. In these cases, the highest score is kept, since the correctness proof of the algorithm shows that scores are, at any time during the computation, upper-bounded by the true PageRank scores, i.e., the scores to which the JXP scores converge to. Therefore, keeping the highest values provides a speedup in convergence. In addition, local pages with links to pages outside the local graph do not need to know the exact location of those, since links to non-local pages are represented as links to the world node. With content dynamics, however, three new events come into play, and the algorithm needs to detect them: pages can be added, modified, or deleted.

6.2.1 The New World Node

So far we actually did not consider the problem of invalid information kept in the world node in case of peers leaving the system (taking their pages with them). One idea would be to keep for each page in the world node that points to a local page a list of peers that had reported a score for that particular page. The number of data to keep track of (bookkeeping) should be constant or growing sublinearly. Keeping track of all peers that store a particular page is infeasible, since it would require massive amounts of storage caused by overly popular pages, i.e., pages likely to be stored at many peers, like for instance, `google.com` or `cnn.com`.

Instead of remembering all peers that have reported scores for a particular page, we opt for storing the last χ peers that reported a score, i.e., we store for each page a list of pairs (peerId, score) for the last χ scores seen for the page, along with the corresponding peer. The parameter χ can depend on the storage capacity of each peer, but we envision χ to be in order of $O(log N)$, where N is the number of peers in the network. This limitation to a certain length is reasonable, since the probability that the list for a page becomes empty, while

there is still some peer in the network that hosts the page, is small. Even if it happens, the page will be rediscovered, due to the basic JXP performance. Hence, the actual choice of χ is not crucial for the performance of JXP.

In addition to remembering external pages with links to the local graph, the world node now also needs to keep track of external pages that are *pointed to by* local pages. This way we can correctly reconstruct both links from and to the world node. Here we also apply the approach of keeping a list of limited size containing the last χ peers met that contained the page, but no score is needed, since they do not directly influence the local scores.

6.2.2 JXP Meetings Adapted

In the JXP algorithm, the meetings are of fundamental importance for the effectiveness and correctness of the algorithm. With dynamics, their role becomes even more crucial: it is through the meetings that peers will be able to detect the changes in the network. As stated before, a change can be of one of the three types: pages can be added, modified, or deleted.

Page addition is a trivial problem, since the algorithm is already designed to discover non-local pages. Recall that, in the algorithm, a peer sends information about both local pages and pages currently in its world node. With the world node now storing scores lists instead of single scores, a decision has to be made about what to send for those pages in the world node. For keeping message cost small, our solution is to send a single (peerId, score) pair per page, where the score is computed by averaging all scores currently known for the page. With the limit on the size of the lists, and a fair amount of meetings performed, old scores will gradually be replaced by updated, better scores, and the average is then expected to converge to the correct score of the page. For the peerId, we can simply choose the most recent peer met for that page, since chances are higher that this peer will remain for a longer period in the network.

Page deletion might occur when peers that reported information for the page have left the network or have changed their contents. Whenever one of the two happens, the reference for that peer is removed from the world node. If the list of peers for a page becomes empty, it is assumed that the page no longer exists, and therefore must be removed from the world node.

It could also happen that a page had its contents modified, so it could still be reached but the new information given for that page contradicts previous information. Since the content of a page itself is not needed for the JXP computation, the only two possible changes in a page are changes in the score and changes in the outgoing links. Changes on the score are not considered, since peers are constantly updating this information, so for detecting that a page has been modified we check whether the outgoing edges have been modified. If so, the page is initially removed from the world node and re-added with the new information. Remember that the world node keeps the information about the outgoing links for every page it stores, so a simple comparison of the current link information stored at the world node with the one being sent by the other

peer can determine if a page has been modified or not.

What is left to describe is how to detect when a peer has left the system. In P2P networks, it is very common that peers temporally leave the network and return to it a short later. In such situations, we would rather leave the world node unchanged and wait until the peer returns. Therefore, a single failed attempt to contact a peer sometimes might not be a good indication that the peer has left the network indefinitely. Instead, we keep a counter of consecutive failed attempts made to contact a peer, and only if this number is above a certain threshold, that can be tuned according to the network behavior, we assume that the peer is no longer alive, and its references should be removed. Upon a successful attempt this counter is reset.

6.2.3 Storage and Network Bandwidth Costs

For the JXP algorithm to work under peer and content dynamics a few modifications had to be done that have affected the storage and network bandwidth costs of the algorithm. However, we show that even though the requirements have slightly increased, the costs are still within an acceptable limit.

Again, the storage cost can be divided into the cost of storing the local graph and the cost of storing the world node, i.e.,

$$SC_P = SC_G + SC_w$$

where SC_P is the storage cost at peer P, SC_G and SC_w, are the local graph and world node costs, respectively.

For every local page in the local graph, besides storing the ID of the page, the list of the outgoing links, and current JXP score, we now need to add, for each outgoing link, a list of size χ containing the identifier of the last χ peers met that store the page to which the link points to. So the new cost become

$$SC_G = \sum_{i \in G} (SC_{ID} + SC_{score} + out(i) \cdot (SC_{ID} + \chi \cdot SC_{ID_P}))$$

where G is the local graph, $out(i)$ is the outdegree of page i, and SC_{ID}, SC_{score}, SC_{out} are the costs of storing the identity of a page, the score of a page and the identify of a peer, respectively.

A similar list with peers' ID is kept for every page stored at the world node, besides the ID of the page, its score and outdegree. In addition, we have to remember peers which were failed to contact. So the total cost of storing the new world node is given by

$$SC_w = \sum_{i \in G} \sum_{\substack{j \mid j \in W \\ j \to i}} (SC_{ID} + SC_{score} + SC_{out} + \chi \cdot SC_{ID_P}) + \sum_{p \in failed} SC_{ID_P} + SC_{count}$$

W is the set of pages represented at the world node, and $failed$ is the set of peers that were failed to be contacted. $SC_i count$ is the cost of storing a counter for the number of failed attempts. When there is a successful attempt to contact a peer, that peer is removed from the $failed$ set (in case it is in the set).

The storage cost of the local graph is still linear in the number of incoming and outgoing links of local pages, and on average each page has only a handful of in-links and out-links. Therefore storage cost is $O(n)$, where n is the size of the local graph. The first part of the world node storage cost is also $O(n)$, but the number of peers in the $failed$ does not depend on the number of local pages. In the worst case, the $failed$ set could contain every other peer in the network, but in practice we expected a much smaller number.

When sending a message to another peer, neither the lists with peers' id, nor the set of failed to connect peers are needed. Therefore, network bandwidth costs remains $O(n)$.

6.3 Experimental Evaluation

6.3.1 Setup

For the experiments we used a slightly larger dataset than the ones from previous chapters. The new dataset was obtained in 2005 by crawling parts of the .eu domain, and contains 862,664 pages with 19,235,140 links. It is available under `http://law.dsi.unimi.it/`, and accessible using the WebGraph framework [BV04], available under `http://webgraph.dsi.unimi.it/`. For a meeting, a peer contacts a randomly chosen peer in the network, and asks for its current local knowledge.

Dynamic Model

To model peer behavior, we use previous works [LNBK02, PRU01] that have derived mathematical models that closely represent the dynamics observed in P2P networks. More specifically, peer joins are expected to follow a Poisson distribution, i.e., the probability that n peers join the network on the next time interval can be written as

$$P_\lambda(n) = \frac{\lambda^n}{n!} e^{-\lambda},$$

where λ is the average number of peers joining the network per time interval. Peer leaves, in turn, follow an exponential distribution: given the average number of drop outs in one time interval (μ), the probability that a peer leaves the network after x time intervals is $F(x) = 1 - e^{-\mu x}$. Note that both distributions are equivalent, since the interval between two consecutive events of the Poisson distribution follows the exponential distribution. In the following experiments, we used these models to generate peer dynamics. For the content dynamics, we randomly choose a percentage of the peers and replace their local graphs by performing new crawls.

6.3.2 Performance Metrics

Like in the previous chapters, we construct a total ranking from the distributed scores and we compare this JXP ranking against the true global PageRank ranking. But since we are trying to evaluate the performance of JXP under network churn, the evaluation becomes more complicated, once the baseline, i.e., the PageRank scores of all pages currently available in the system, is not static anymore. Hence, for every change in the network, we consider the union of all pages currently in the system, and compute the baseline scores. Then, for some points in time the JXP scores are compared to the baseline at that time point.

For comparing the ranking given by the JXP algorithm and the ranking given by traditional, centralized PageRank we again use the Spearman's footrule distance and the linear score error for the top-k pages (see Chapter 4), as well as the cosine similarity between the two vectors and the L1-norm of the vector containing the JXP scores.

6.3.3 Results

The experimental evaluation consists of two parts. First we report on the performance of the estimator presented in Section 6.1. Then, we present results on the performance of JXP under network and content dynamics.

Figure 6.2 (left) shows the quality of the estimator compared to the exact values, i.e., the number of pages currently in the system. For this experiment, we simulated random peer meetings within a system of 50 peers. Each peer randomly draws from a pool of $150,000$ pages between 250 and 1000 distinct pages. Peers are either active or inactive, according to the exponential distributions that models the peer behavior. Each peer maintains only 4 hash sketches with 2^{10} bitmaps each, resulting in a negligible storage consumption of $32KBytes$. After 2 meetings, each peer shifts the sliding window over the hash sketches by one position, i.e., each hash sketch is valid only for 2 meetings. As shown in Figure 6.2 (left), the estimation accurately follows the exact values, with major drastic fluctuations being smoothed out. To get a deeper insight about the usability of our estimator inside JXP, we also report on the distribution of count estimates, as presented in Figure 6.2 (right). The variation between the first and the third quartile is remarkably small, indicating that peers nearly agree on one particular value, which is important for the performance of JXP. Note that both figures shows one particular, representative run, and that it is not smoothed over multiple runs or multiple parameter choices.

For the experiments with the adapted JXP we increased the size of the network to 1000 peers. Overlaps among local graphs are allowed, and the collection of all peers holds in total around $100,000$ documents. Peer and content dynamics are introduced in the system always after a certain number of meetings has occurred in the network. We considered both successful and unsuccessful meetings for the counter. We then varied the parameters of the peer churn and content dynamics models, to simulate different degrees of dynamics.

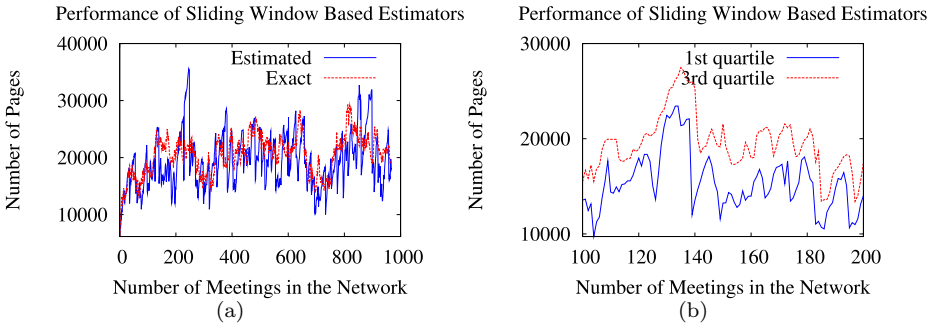


Figure 6.2: Hash sketch based estimation of the number of documents under network churn.

We present results for two scenarios: *Moderate Churn*, with join and leave rates of 100/0.1, and a change of the contents of 1% of peers; and *Heavy Churn*, with join and leave rates of 200/0.1, and a change of the contents of 5% of peers. For a better understanding of the impact of dynamics the following results were obtained without the use of our distinct page count estimator, and peers were artificially told about the correct size of the global graph. Figure 6.3 shows the results obtained, where the baseline simulates the case without dynamics. Note that the actual values of the linear score error are in general not meaningful: since scores correspond to stationary probabilities, they are expected to sum up to one, so if there is an increase of the number of pages in the network, the scores drop, which explain the behavior of the curve. However, the key insight obtained here is that the error decreases even under dynamics. The other three accuracy measures show very good performance of JXP under churn, in particular the L1-norm nicely follows the baseline, even though the underlying global graph is not stable.

6.4 Discussion

One of the main characteristics of a P2P network is its dynamic nature, with peers constantly joining and leaving the systems. We have adapted our JXP algorithms to enable the distributed computation of authority scores in the presence of network churn. We have identified potential shortcomings of our JXP method, and presented means to extend the algorithm to cope with network dynamics, while keeping storage requirements and message costs low. We have also presented an estimator based on hash sketches and sliding windows to count the number of distinct pages in a dynamic network, which is one of the basic input parameters of JXP. Our experiments have shown that our estimator is effective when computing an approximation for the total number of pages in the network, and that the modified version of the JXP algorithm is able to adapt to the changes in the network.

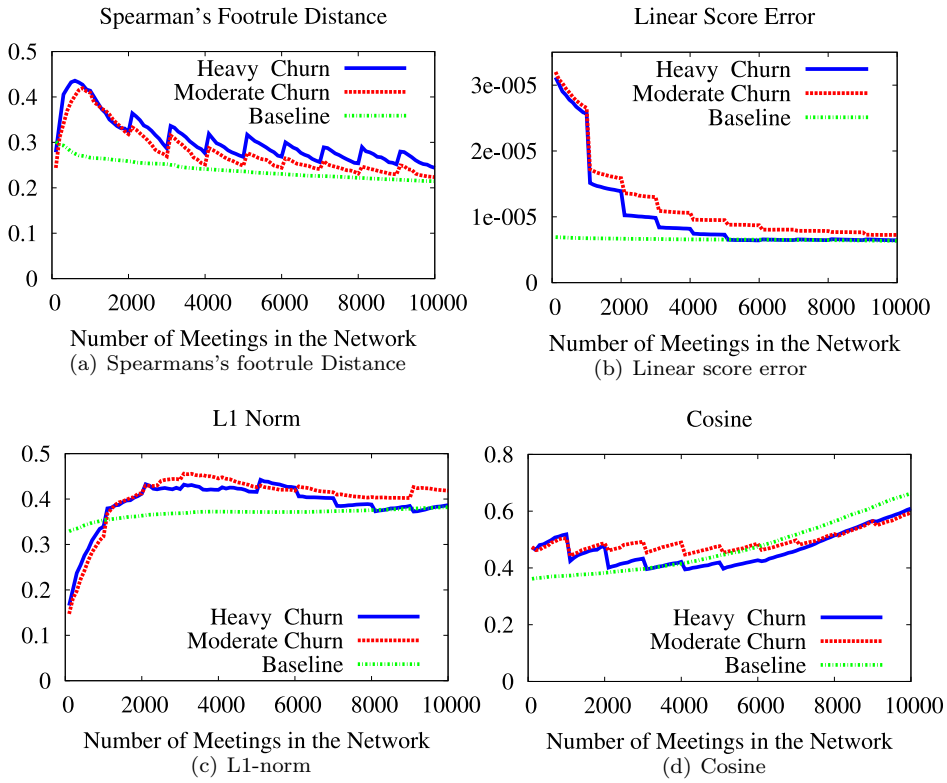


Figure 6.3: Performance of JXP under P2P Dynamics.

Chapter 7

p2pDating — Creation and Maintenance of SONs

Semantic Overlay Networks (SONs) [ACMHP04, BMR03, CGM04, TXKN03] are a network organization that improves query performance while maintaining a high degree of peer autonomy. Peers with semantically similar content are connected through an overlay network, and a peer can belong to multiple overlay networks (e.g., if its contents is diverse). Queries are routed only to the appropriate semantic overlay networks, increasing the chances that matching information (e.g. files, documents) will be found quickly, and reducing the load on peers having unrelated content.

In this chapter we introduce an algorithm, coined *p2pDating*, that allows autonomous peers to form context-rich SONs, and we show how these SONs can be utilized during query routing in P2P web search engines and also for improving the performance of our JXP algorithm.

There are many challenges when building SONs, regarding how peers are assigned to SONs and to which SONs a query should be sent. According to the initial idea, peers should be evenly distributed among SONs, so queries can be answered fast, as fewer peers have to be asked; and each peer should belong to a small number of SONs, so that each peer has to handle only a few number of connections. However, in the real world, the distribution of peers over semantic classes is expected to be very skewed and dynamic as many peers will belong to some very popular topics that are constantly changing, whereas some uncommon classes will be less populated. Moreover, in most of early approaches, an algorithm that classifies the peers' contents into one or more predefined classes was used. Each of these classes defines a SON. This leads to a fixed configuration of the SONs, so that the performance is highly dependable on a good choice of the classification algorithm and the classes, and it also requires that all peers use these same algorithm and classes, which is undesirable. To overcome the restriction that the peers are classified into these strict topic schemes, our proposed p2pDating algorithm gives more autonomy to the peers when deciding which SONs they should join. It works by rearranging the connections between

peers, according to the peers' criteria of a "good" neighbor (i.e., a "friend"), and using caching to remember the peers that were defined as friends. Possible measures for deciding if a peer should be considered a friend or not could be, for instance, the overlap between pages held by the peer and pages held by the candidate for being a friend, the similarity between their pages, history of the peer, level of trust, etc. A peer also has the option to delete an already established link with a friend, if it has either changed its selection criteria or found more interesting peers.

We proceed by explaining in detail the p2pDating algorithm and also examples of different criteria for finding friends in the network. The usefulness of the algorithm is tested in experiments that show how p2pDating can improve the performance of the JXP algorithm, and how it can be used to efficiently and effectively find promising peers during query routing.

7.1 The p2pDating Algorithm

The idea of p2pDating is to create SONs in a P2P environment, where a peer has autonomy when deciding which SONs it wants to join. The approach works by having peers meet other peers that they still do not know (like "blind dates"). If a peer "likes" another peer, i.e, if this other peer has information that is interesting for the peer, it might want to remember this peer, and insert it into the *friend list*. On the other hand, if the peer decided that the other peer is not interesting, it is most likely that it might not want to remember this peer, so no link is created or if there is already a link between them, it might be dropped. We advocate that caching (i.e. remembering) of high quality peers is the natural way to create SONs.

The process starts with a randomly connected network and runs infinitely since peers are constantly joining and leaving the network. SONs will dynamically evolve from this process, as semantic links are more and more refined. In a dynamic P2P network, we expect that the SONs are continuously changing to adapt to the changes in the network, i.e., changes in the peers' behavior, peers' contents, etc. The semantic links are represented by entries in the friends lists. It is important to emphasize that no physical links are created. Semantic links can be seen as abstract links. When a peer joins the network its friends list is empty and will be filled over time. Figure 7.1 illustrates three dynamically evolving SONs, each one represented by a different color, where we can see that besides the semantic links there are also additional random links. The random links are physical links needed to keep the whole network together and are dictated by the underlying P2P network protocol. For instance, in the Chord protocol the random links correspond to the entries in the finger tables [SMK+01].

7.1.1 The Semantic Routing Table

The friends are annotated with statistics to form a *semantic routing table* (SRT) in which the peers are ordered according to their usefulness. Table 7.1 shows

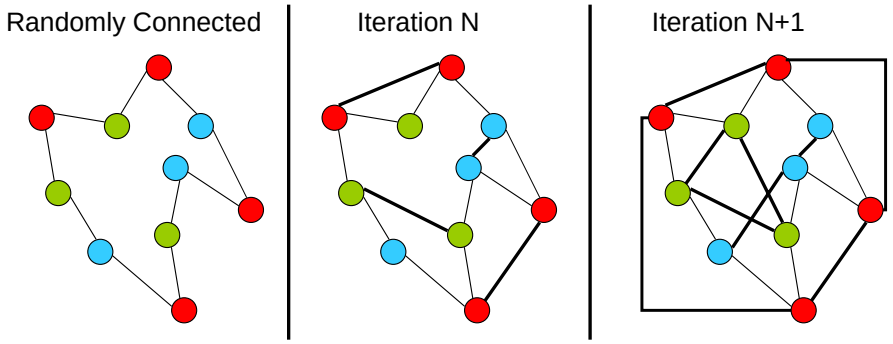


Figure 7.1: Dynamically Evolving Semantic Overlay Networks. Links inside a SON are represented by thicker lines. The thinner lines correspond to the random links. Peers with the same color belong to the same SON.

	IP	**Overlap**	**Similarity**	**Credits**	**Last Used**	**Usage Frequency**
Peer A	192.168.1.3	4%	70%	434	two days	34
Peer B	192.168.1.5	1%	30%	344	yesterday	12
Peer C	192.168.1.2	7%	50%	121	today	4

Table 7.1: Example of a semantic routing table containing statistics about known friends.

an example of the semantic routing table held by some peer, which shows five possible measures for assessing the usefulness of a peer: percentage of overlap between the peers collection, similarity between the collections, credit points (for instance for good cooperation in the past), last time the peer was used for a particular task (e.g., query routing), and the usage frequency.

There are many ways to define a friend, as it will be discussed later in this chapter. The measures and values displayed in the example are just an illustration.

When a new friend is found, an entry containing information about this peer is added to the table. Friends lists have a fixed length, which means that current friends might need to be dropped (according to some criteria) from the table, so that new friends can be added. Dropping a friend corresponds to remove an abstract link in the network. Each peer creates its friends list independently of other peers. In particular, "friendship" is not generally symmetric. If peer *A* adds peer *B* into its friend list, *A* is not automatically inserted into *B*'s list. It is up to *B* to decide whether to add *A* or not. This means that the links created by p2pDating form a directed graph.

7.1.2 Finding New Friends

Friends lists, besides defining the links in the SONs, can also be used to find new friends in an intuitive manner, by looking at the SRTs of other peers: if peer A finds peer B interesting, it is very likely that the friends of B will be interesting to A as well. Therefore, besides adding B into A's friend list, we also add B's friends into a so-called *candidate list*. Then, for the next meeting, a peer can choose to meet a friend of one of its friends, instead of picking a peer at random since being a friend of a friend is a stronger recommendation. Alternatively, a criterion other than the one used to define a friend can be used to decide whether to add new candidates to the list or not. In this case, candidates are not necessarily friends of one of the peer's friends. Candidates lists, like friends lists, also have fixed length, which means that if the maximum number is reached, candidates have to be dropped.

7.1.3 p2pDating Algorithm

Algorithm 7.1 shows the procedure of picking a peer for the next meeting. It draws a random number between zero and one, and according to some predefined probabilities, it chooses a peer from the candidate list or a peer from the friend list, or a random peer in the network.

Algorithm 7.1 The *choosePeerToMeet*() procedure

1: $r \leftarrow random(0,1)$
2: **if** $(r \leq \alpha)$ **then**
3: $P \leftarrow$ a peer from the candidate list
4: **elsif** $(r \leq (\alpha+\beta))$ **then**
5: $P \leftarrow$ a peer from the friend list
6: **else**
7: $P \leftarrow randomPeer()$
8: **end**
9: return P

We can think of scores for peers in the candidate list, based on the scores of the peers where they were defined as friends. Thus, we can select the peer with the highest score when choosing a peer from the candidate list. It is important that peers have an updated view of the network, as peers can change their contents or eventually leave the network. Therefore, peers have to revisit their friends from time to time. For search engine applications, friends will be visited during query execution, so these updates can be integrated into the standard querying process. Another possibility is to assign a *time to live* (TTL) to every friend so that peers are automatically dropped, or revisited to re-assess their usefulness. In addition, the probability of picking a peer at random should not equal zero, to assure that every peer in the network can be reached.

Algorithm 7.2 shows the pseudo-code for the p2pDating algorithm. A peer chooses another peer for the next meeting and contacts it. Then it decides

whether the peer is a friend or not, based on the peer's content. If so, the peer is added to the friend list. It then decides if the friends of the peer are good candidates, adding them to the candidates list in case of a positive answer. As said before, we can simply choose to follow the chain of friends, by making *hasGoodFriends*(P) return true whenever P is a friend, or we can use any other criterion to implement this function. The process of adding peers to the friends or candidate list checks if the maximum number of the peers on the list has been reached, and removes peers, if necessary.

Algorithm 7.2 p2pDating Algorithm

```
1: repeat
2: P ← choosePeerToMeet()
3: contact P
4: if isFriend(P) then
5:    add(P, friend list)
6: end if
7: if hasGoodFriends(P) then
8:    C ← friends of P
9:    add(C, candidate list)
10: end if
```

Figure 7.2 illustrates a meeting between two peers, showing their friends and candidates lists before and after the meeting. We can see two possible cases: 1) when a peer decides that the other peer is a friend and has good friends, the friend and candidate lists are updated, and 2) when a peer decides that the other peer is not a friend and also does not have good friends, the lists remain the same. The other two possible cases are when a peer decides to update only its friends or its candidates list.

Choosing a peer from the friends list means revisiting an already known peer. In a highly dynamic P2P network this is very important in order to have an updated view of the network, as peers can change their contents or eventually leave the network.

To avoid that the friends/candidates lists grow forever, we need a replacement strategy that keeps track about peers that are no longer interesting and thus replaced by other peers or just dropped from the cache, e.g., if it turns out that these peers have left the network. As we limited the size of a friend list, we use a ranking of friends so that if the size of the list reaches its maximum, the lowest-ranked friend is dropped. The friends list's order can be defined by a combination of measures that will be presented in the next section.

7.2 Defining Good Friends

As explained earlier, when peer A meets peer B in the network, it accesses B's content, i.e., the information that peer B has made visible for the others, and decides whether to establish a link to B or not, based on a measure of the

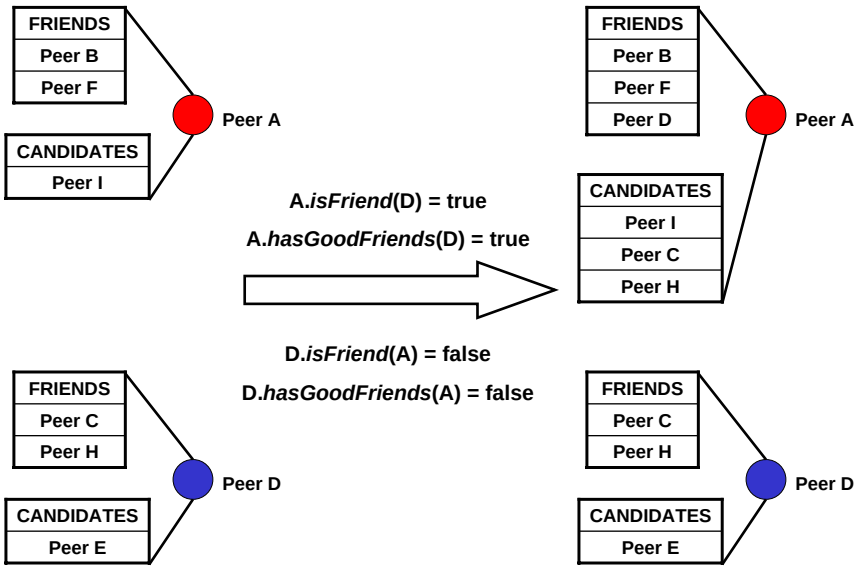


Figure 7.2: Example of friends and candidates lists after a peer meeting.

quality/usefulness of peer B. It also decides, based on the same or, alternatively, another measure, if B's friends should be also visited. There are many different measures that can be used, depending on the purpose of the SON that is being formed, like good behavior in the past, collection similarity, overlap between the collection, authority scores, etc.

Figure 7.1 shows some measures that can be used to find the most promising peer for a particular need (for instance, query routing). We can see that sometimes a single measure might not give enough information to decide which peer to choose. For instance, peer A is the best choice if we consider the number of credit points, whereas peer B seems to be most promising if we take a look at the overlap. So, obviously, there is great need for an aggregation function that combines the single measures in a meaningful way, since selecting a peer based only on a particular measure can be misleading. For instance, it might be the case where a high quality peer has many pages that we already know, and a peer that offers a lot of new information has a lower quality measure.

Aggregation functions can be of any kind, but usually they are expected to be simple, for instance, a linear combination of two or more quality measures, since the definition of a good friend and/or candidate is most of the times very intuitive.

As a peer's content can be very broad and diverse, applying quality measures to its complete collection can be inaccurate. In such cases, a peer might consider to split its page set into topic-specific subsets. Each peer would then maintain more than one semantic routing table, more precisely, one for each topic it

is interested in, and usefulness assessment would be made for each particular topic. Although this creates additional cost, it might increase the accuracy of the quality assessments, since comparing the semantic similarity of two collections might be misleading in the case where collections are related to more than one topic. It is important to note that it is up to the peers to decide how they classify their content, and a globally given classifier is not required.

In addition, peers can organize the different topics in a hierarchy (see Figure 7.3 for an example) with edge weights corresponding to topic-subtopic similarities. The edge weights can be interpreted as some kind of confidence measure that gives weight to the semantic query routing. Thus, peers can leverage the semantic routing table even in the case where the query does not correspond directly to a topic, by considering SRTs from related topics. Moreover, when the query fits to a specific topic, SRTs from sub-topics or from more general topics can be incorporated into query routing using a weighted quality assessment.

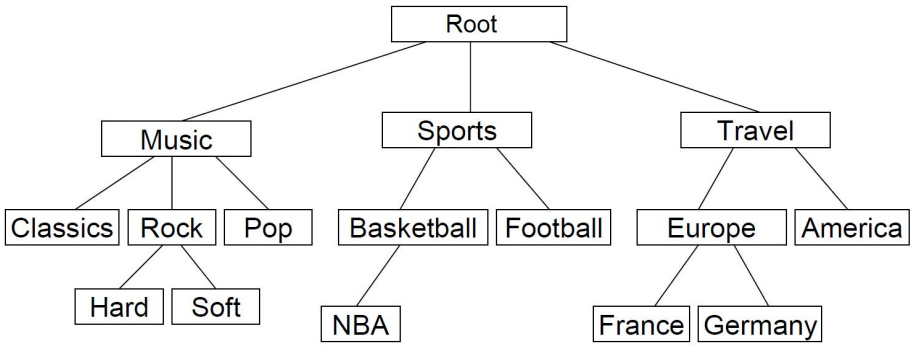


Figure 7.3: Example of a hierarchical semantic classification scheme.

We now proceed by describing some of the possible measures that can be used to identify good friends in the network.

7.2.1 Quality/Usefulness Measures

In the p2pDating algorithm, each peer is free to decide which criteria to use when choosing a peer to its friends list. There are many ways to access the quality/usefulness of a peer. Below we present some of the measures that could be used, and how they could be used.

History

In any application that requires collaboration among peers, the presence of malicious peers can pose a problem. Creating a SON is not an exception. By sending false content, a peer's quality is measured wrongly, and peers might be induced to choose peers which in reality have poor content, instead of real good ones.

As recently proposed by [TSW04], remembering excellent behavior in the past is a natural way to find friends. We can, for instance, give credit points to peers for good cooperation. The history of a peer is very useful when combined with other measures. We could for instance, weigh other measures, based on the past cooperation of a peer. This would decrease the impact of malicious peers and can be seen as an incentive mechanism as it can be used to prioritize incoming queries from friends.

What a peer stores and how it behaves can vary over time. For instance, it can find and store new pages about its topics of interest or change its preferences and start storing pages about a different topic. Furthermore, non-collaborative peers might become more collaborative in order to have access to network resources. The number of credit points should reflect these changes. One solution is to reset the credit points counter, at regular intervals. The time period between two resets can be tuned by observing the frequency at which peers change. Another alternative is to specify a time window, such that credit-points are given based only on the observations made inside this time window. The size of a time window can be defined also based on the peers' dynamics. Keeping track over the behavior in the past can be seen as a utility to predict the usefulness of a peer in the future.

Overlap

Avoiding the retrieval of duplicate pages is a crucial issue in large scale distributed information system. Recall that we consider peers to be autonomous and have their own local collection, generated, for instance, by focused Web crawls. The problem inherently associated with this scenario is that collections can have a high mutual overlap; thus, it is likely that the query initiating peer will retrieve pages that it already knows from its local collection. High quality pages are useless if they are already known: there is no need in querying a peer when it is known before-hand that this peer has an extremely high overlap with regard to the own collection. The mutual overlap between peers has to be taken into account while selecting promising peers for a particular query. Overlap-aware techniques [BMT+05b, MBTW06] avoid retrieving redundant information so that a certain level of recall can be reached by querying fewer peers, compared to the non-overlap-aware approach. For instance, if the most promising peers have exactly (or nearly) the same collections only the first peer can deliver valuable results whereas the following peers will not contribute with any new pages.

In the task of measuring overlaps of collections stored at different sites the main issue that arises is that sending the whole collections across the different peers is prohibitive, since it would incur in a large bandwidth consumption. Instead, techniques that are based on *statistical synopses* are largely used.

Statistical Synopses

Fundamentals of statistical synopses of sets and multisets have a rich literature, including work on Bloom filters [Blo70, FCAB00], hash sketches [FM85], and min-wise permutations [BCFM98, BCFM00]. Hash sketches were already introduced in Section 6.1.1.

A Bloom filter (BF) [Blo70] is a simple data structure that represents a set as a bit vector in order to efficiently (in time and space) support membership queries. With bit vectors being a very compact representation of a set, Bloom filters are an ideal representation in an environment where storage and bandwidth consumption are an issue. For a particular set, a Bloom filter is a bit map of length m and is created by applying k hash functions on each member element, each yielding a bit location in the vector. Exactly these element positions of the Bloom filter will be set to 1. To check if a given element is in the set, the element is hashed using the same hash function and the corresponding k bits of the Bloom filter are examined. If there is at least one of these bits that is *not* set to 1, the element is definitely not in the set; otherwise it is conjectured that it is in the set. There is a non-zero probability that the examined k bit positions were set by other documents, thus, creating a *false positive*. The probability of a false positive can be calculated by

$$pfp \approx (1 - e^{-kn/m})^k,$$

where n is the number of items in the original set [FCAB00].

Min-Wise Independent Permutations, or MIPs for short, have been introduced in [BCFM98, BCFM00]. This technique assumes that the set elements can be ordered and computes N random permutations of the elements. Each permutation uses a linear hash function of the form

$$h_i(x) := a_i * x + b_i \; mod \; U,$$

where U is a big prime number and a_i, b_i are fixed random numbers. By ordering the resulting hash values, we obtain a random permutation. For each of the N permutations, the MIPs technique determines the minimum hash value, and stores it in an N-dimensional vector, thus capturing the minimum set element under each of these random permutations. The technique is illustrated with an example in Figure 7.4. Its fundamental rationale is that each element has the same probability of becoming the minimum element under a random permutation. By using sufficiently many different permutations, we can approximate the set cardinality.

An unbiased estimate of the resemblance between two sets, S_A and S_B, i.e.,

$$Resemblance(S_A, S_B) = \frac{|S_A \cap S_B|}{|S_A \cup S_B|},$$

is obtained by counting the number of positions in which the two vectors have the same number and dividing this by the number of permutations N [BCMR04]. In the example of Figure 7.4 the two MIPs vectors have two matching positions, out of the six permutations. Essentially, this holds as the matched numbers are guaranteed to belong to the intersection of the sets.

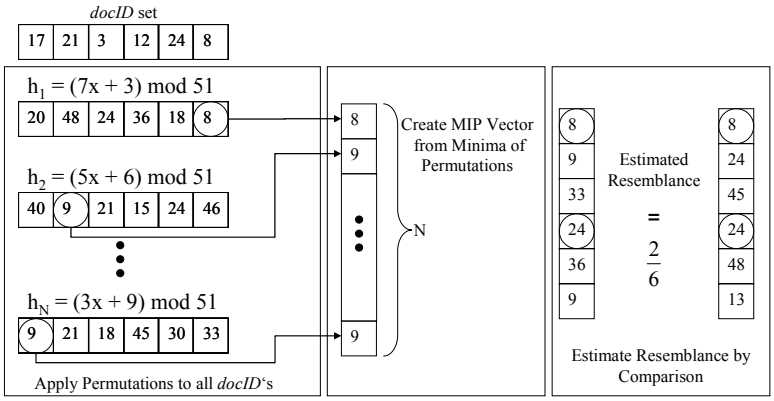


Figure 7.4: Example of min-wise independent permutations estimation of resemblance.

The overlap between the two sets S_A and S_B, i.e.,

$$Overlap(S_A, S_B) = |S_A \cap S_B|,$$

can be derived from the resemblance measure given that

$$|S_A \cup S_B| = |S_A| + |S_B| - |S_A \cap S_B|.$$

By diving both sides of the equation above by $|S_A \cup S_B|$, and rearranging the terms we have

$$\begin{aligned} Resemblance(S_A, S_B) + 1 &= \frac{(|S_A| + |S_B|)}{|S_A \cup S_B|} \\ &= \frac{(|S_A| + |S_B|)}{|S_A \cup S_B|} \frac{|S_A \cap S_B|}{|S_A \cap S_B|} \\ &= \frac{Resemblance(S_A, S_B) \cdot (|S_A| + |S_B|)}{|S_A \cap S_B|}. \end{aligned}$$

Therefore we can write

$$Overlap(S_A, S_B) = \frac{Resemblance(S_A, S_B) \cdot (|S_A| + |S_B|)}{Resemblance(S_A, S_B) + 1}.$$

Another measure to predict collection overlap is to compare the bookmark collections. If the local page collections have been generated by web crawls we can treat bookmarks as crawl seeds. Thus, assuming roughly the same crawling strategy, comparing two sets of bookmarks can provide a good approximation of the collections' content overlap. As an alternative to overlap and resemblance, we can also consider the notion of *Containment* an appropriate measure of mutual set correlation [Bro97].

$$Containment(S_A, S_B) = \frac{|S_A \cap S_B|}{|S_B|} = \frac{Resemblance(S_A, S_B) \cdot (|S_A| + |S_B|)}{(Resemblance(S_A, S_B) + 1)|S_B|}.$$

Semantic Similarity

We can measure the semantic similarity between two peers by comparing their bookmarks or their complete collections. More specifically, we can compare the peers in three aspects: i) regarding their URL sets, ii) regarding the term frequency distributions in the pages referenced by the bookmark lists, or iii) regarding the term frequency distributions in their complete collections.

As for term distributions, we could use the relative entropy, also called the Kullback-Leibler distance [Kul59], as a measure of information inequality. It is defined by

$$KL(f, g) := \sum_x f(x) log \frac{f(x)}{g(x)},$$

where f and g are discrete probability distributions. The relative entropy has important mathematical properties; for example, it is non negative and equals zero if and only if $f = g$. If a peer wants to find semantically related peers, the benefit of each candidate peer can be seen as inversely proportional to $KL(f, g)$. f and g denote the term frequency distributions in all pages in a collection or only the pages referenced by the bookmark lists.

Link Distribution inside Semantic Communities

[FLGC02] shows that the Web is self-organizing in the sense that Web communities are formed automatically. These communities can be easily identified by considering the link distribution within these pages. It is shown that Web pages have more links to other pages inside their community than to pages that are outside their community, so the analysis of the link structure can also be used to find semantically related peers.

To give an example, we used the Web Crawl dataset described in Section 4.5.1, and analyzed the topic distribution of outgoing links from pages under the category "sports". The result is shown in Figure 7.5. As we can see, sports pages mainly point to other sports pages, an observation that one can support by personal experiences when surfing through the Web.

In the following two sections we describe two applications that can benefit from the SONs created using the p2pDating algorithm.

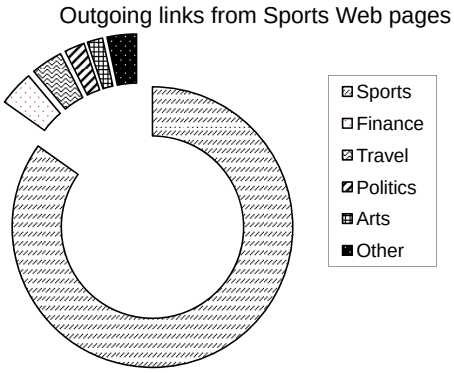


Figure 7.5: Outgoing links from sports pages mainly point to other sports pages.

7.3 SONs for the JXP Authority Scores Computation

It is clear that, in the JXP algorithm, peers do not contribute to each other in an uniform way. How much a peer A will benefit from peer B for improving its local authority scores heavily depends on the degree of connectivity between their two local graphs and the level of overlap between them. Peers will not gain much from meetings when there are only a few links between the local graphs or if the local graphs are almost identical.

The performance of the algorithm could be improved in the presence of an overlay network where peers are clustered together according to their degree of connectivity (which is an indicator of semantic similarity as well). For creating the semantic overlay network we first need to compute at each peer two sets: one containing its local pages' IDs and another containing the IDs of all the successors targets (outgoing links) of all local pages. The network bandwidth consumption is kept small by using MIPs vectors for representing these two sets. This makes the messages for transmitting this information small, such that they can be piggybacked onto established communication. For a given peer A we call these vectors $local(A)$ and $successors(A)$.

Assuming that peer A has received information from peer B, the p2pDating algorithm decides whether peer B should be considered a friend by computing $Containment(successors(B), local(A))$, i.e., the fraction of local pages in peer A that have incoming links from local pages in peer B. If the value is above some pre-defined threshold, peer A adds peer B to its friend list.

For finding potential candidates for being a friend, the algorithm computes the overlap between the local page sets of A and B, i.e. $Overlap(local(B), local(A))$. The idea here is that, given three peers, A, B and C, if peer C has many links to peer B, and the overlap between A and B is relatively high, it is very likely that C will have many links pointing to A as well. If the overlap is relatively high, friends of B will be inserted into A's candidate list.

With the semantic overlay network, the JXP algorithm can identify the best peers to exchange information, instead of choosing peers at random, which leads to fewer meetings to reach a good approximation of the global authority scores.

7.3.1 Experiments

To show the benefits of having SONs for guiding the choice of peers to meet, we have implemented the p2pDating algorithm to use the criteria just described when adding peers to the friends and candidates lists. More specifically, the functions *isFriend*() and *hasGoodFriends*() from Algorithm 7.2 were defined as follows

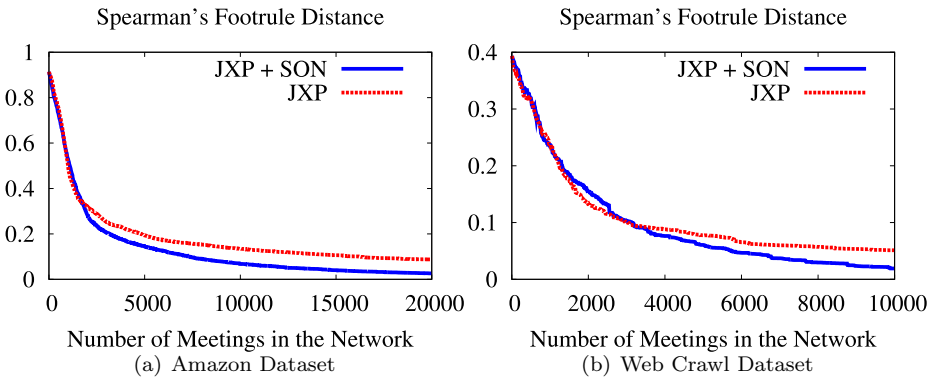
$$P_A.isFriend(P_B) = \begin{cases} \textbf{true}, & \text{if } Containment(successors(B), local(A)) \geq \gamma_F \\ \textbf{false}, & \text{otherwise} \end{cases}$$

$$P_A.hasGoodFriends(P_B) = \begin{cases} \textbf{true}, & \text{if } Overlap(local(B), local(A)) \geq \gamma_C \\ \textbf{false}, & \text{otherwise} \end{cases}$$

where γ_F and γ_C are pre-defined thresholds for the containment and overlap measures. The JXP and p2pDating algorithms were then integrated as follows. Running at peer A, the JXP algorithm chooses peer B to meet according to the algorithm described in Algorithm 7.1. peer B then sends, besides the standard JXP message described in Chapter 4, the two MIPs vectors containing the successors and local sets, and the friends list. The JXP meeting is carried out as usual, with the relevant information from peer B being added to the world node of peer A, and updated JXP scores of pages in A being computed. In parallel, the p2pDating algorithm, also running at peer A, checks if peer B qualifies as a friend, and if this is the case, it adds peer B to peer A's friends list. It then proceeds by checking if peer B has potentially good friends. If this is also the case, the peers in the friends list of peer B are added to peer A's candidates list, but initially with no score, since we can not assess their quality from the current meeting. This is done in subsequent small p2pDating meetings, where peer A asks these peers for their MIP vector containing the successors set, which only adds little cost to the bandwidth consumption.

For the experiments we have used the same datasets and settings described in Chapter 4. We compare the performance of the original JXP algorithm without the p2pDating algorithm (where peers are chosen randomly) against the extended JXP with the SON created by p2pDating. Figure 7.6 shows the Spearman's footrule distance for the top-1000 pages for the Amazon and Web Crawl datasets.

We can see that during the first meetings both approaches perform similarly but, as the semantic overlay networks are being formed, the JXP algorithm is able to find the most promising peers, reducing the number of meetings needed for a good approximation to the global PageRank scores. For instance, in the

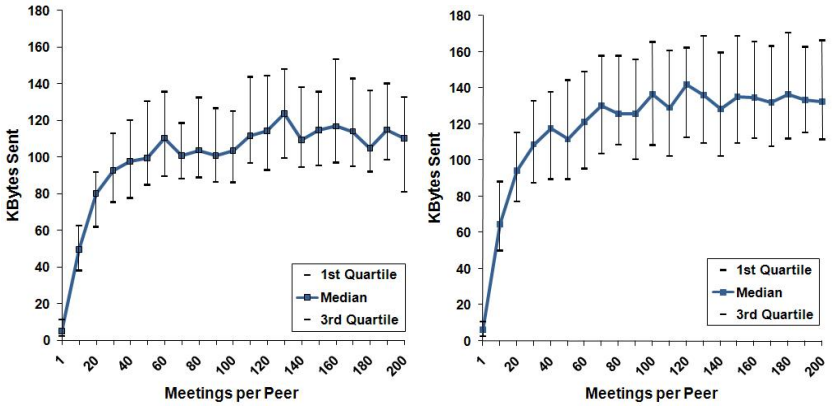


(a) Amazon Dataset (b) Web Crawl Dataset

Figure 7.6: Results showing the benefit of the p2pDating algorithm for the Amazon and Web Crawl datasets.

Amazon dataset, to make the footrule distance drop below 0.1 we needed a total of 16190 meetings without the SON. With the SON this number was reduced to 7340. For the Web crawl dataset, for a footrule distance of 0.05, 5730 meetings are enough with the SON, whereas without the SON not even after 10000 meetings this threshold was reached.

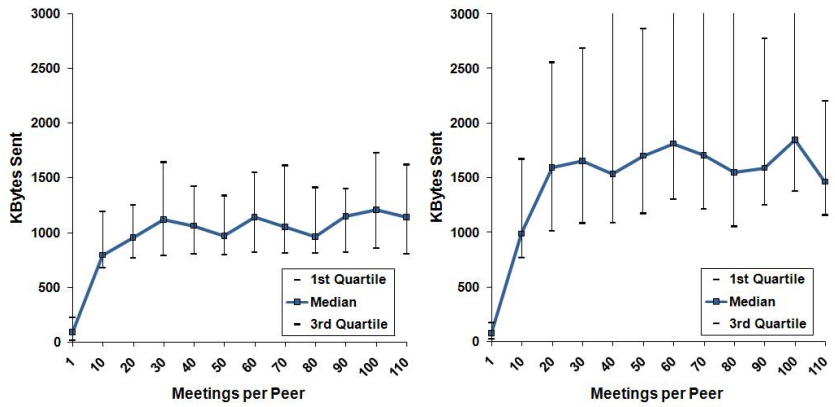
We also measured the additional bandwidth consumption due to the p2pDating meetings. We compare the message sizes of the standard JXP algorithm shown in Figure 4.8, with the message costs of the JXP and p2pDating algorithms combined. Results are shown in Figures 7.7 and 7.8, which shows that there is only a slightly increase in the costs per meeting, which actually pays off in the overall execution, since less meetings are needed.



(a) JXP (b) JXP + p2pDating

Figure 7.7: Message size (in kB) for the JXP algorithm alone and combined with the p2pDating algorithm for the Amazon dataset.

The advantage of having a SON to improve the convergence speed of the JXP algorithm is clear. By finding the most promising peers, many meetings



(a) JXP (b) JXP + p2pDating

Figure 7.8: Message size (in kB) for the JXP algorithm alone and combined with the p2pDating algorithm for the Web Crawl dataset.

with peers that would contribute only little useful information are avoided.

7.4 SONs for Query Routing

The main advantage of SONs is that peers are grouped together in a way that allows for efficient query execution (routing) with solely local knowledge. In contrast, many other approaches make use of some kind of global index that helps with the query routing decision. The CORI scoring approach is one example. As described in Section 4.6.3 CORI ranks peers based on per-term and per-collection information published for instance in a Minerva style index (directory) [BMT+05b, BMWZ05, BMPC07] (see Section 4.6.1). Compared to SONs, such a distributed directory creates a higher network load at query time as meta information have to be retrieved to select the most promising peers. The scoring used in CORI is solely focused on predicting high quality peers, disregarding potential overlap among peers. Bender et al. [BMT+05b] have enriched the Minerva directory with synopses to estimate overlap among peers, and integrated the overlap information in the scoring of CORI.

In the experiments that follow we have built different overlay networks, according to different criteria, and measure their performance both in terms of results quality and bandwidth consumption.

7.4.1 Experiments

For the experiments we have used the same dataset and setup described in Section 4.6.2. We have used 30 popular Google queries taken from Zeitgeist[1] in 2005; they are shown in Table 7.2.

[1] www.google.com/press/zeitgeist.html

Query	Topic	Query	Topic
andy roddick	sports	star wars	movie
american music awards	music	iraq	politics
oscars	movie	sports illustrated	sports
thailand	travel	hurricane charley	politics
music	music	klingerman virus	natural science
mel gibson	movie	diane lane	movie
nfl	sports	gregory hines	movie
berlin marathon	sports	salt lake city	travel
columbus day	politics	fathers day	politics
chicago marathon	sports	fifa 2003	sports
marilyn monroe	movie	matrix reloaded	movie
emmy awards	music	baseball	music
haiti	travel	lebron james	sports
solar eclipse	natural science	real madrid	sports
world series of poker	sports	carmen electra	movie

Table 7.2: Queries

To assess the query routing performances using SONs, we have built different SONs where the quality of peers is assessed in different ways: an overlay measure, a similarity measure, and hybrid measures. We compared the following approaches:

- **CORI**: This is the collection selection strategy as proposed in [CLC95, Cal00], implemented on top of Minerva.
- **Overlap-aware CORI**: This strategy uses the technique presented in [BMT+05b]. We use a combination of a quality based query routing strategy with an overlap prediction method to form a selection strategy that reflects the relative usefulness of a peer with respect to the query initiating peer. The strategy that we employ here is based on (i) the collection selection strategy CORI, and (ii) an overlap estimator based on min-wise independent permutations (MIPs).
- **Overlap Only**: Here we consider only the overlap between peers.

$$Usefulness(A,B) = 1 - \frac{|A \cap B|}{|B|},$$

  where A is the query originator.
- **Similarity Only**: Here we consider only the similarity between peers. This measure uses the Kullback-Leibler distance [Kul59] to assess the semantic similarity, based on the term occurrence distributions.

$$Usefulness(A,B) = \frac{1}{1+KL(A,B)},$$

where A is the query originator.

- **Overlap * Similarity**: This is a combination of overlap and similarity.

$$Usefulness(A,B) = \frac{1}{1+KL(A,B)} * (1-\frac{|A \cap B|}{|B|}),$$

where A is the query originator.

- **Weighted Sum**: This is a weighted sum of the overlap and the similarity measure.

$$Usefulness(A,B) = (\alpha)\frac{1}{1+KL(A,B)} + (1-\alpha)(1-\frac{|A \cap B|}{|B|}),$$

where A is the query originator.

Experimental Results

Figure 7.9 shows the relative recall (see Equation 4.4) considering the top-20 pages from the global ranking, for the above mentioned routing strategies. We also present a baseline, where queries are forwarded to randomly chosen peers. We can see that although CORI and CORI overlap-aware provide the best results, the SONs that have been created using a similarity measure provide very good result quality. We can also see that the overlap-only approach is not enough to deliver good results. We have also conducted experiments for different values of the coefficient α in the weighted sum of similarity and overlap; these showed comparable behavior and are thus omitted.

The good performance of CORI and overlap-aware CORI comes with a high bandwidth costs. Table 7.3 shows the total number of messages, and transferred bytes at query routing time for the complete benchmark of 30 Zeitgeist queries. We consider the average URL length to be 70 bytes. The number of messages consists of the messages to retrieve the data statistics (if needed), and the messages to actually execute the query by sending it to the selected peers. As the SON based routing does not use any statistics, there are only 150 messages required, as we consider the top-5 peers for 30 queries, whereas the CORI and overlap-aware CORI strategies involve 57 additional messages to retrieve the published per-term peer lists. Network traffic consists of the number of bytes for sending the query (on average 2 terms each of length 10 bytes), and retrieving the top-20 result URLs plus scores (floating point numbers) from the queried peers. This is the communication cost for the query execution. In addition, the directory peers cause network traffic when retrieving the statistics about peers' collections. Whereas CORI uses only the standard statistical information along with peer information (IP address and port), the overlap-aware CORI strategy

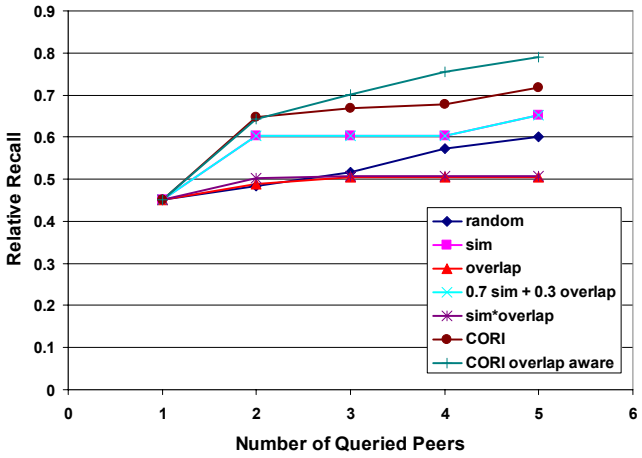


Figure 7.9: Relative Recall

Strategy	Number of Messages	Traffic (in Bytes)
p2pDating strategies	150	225 000
CORI	207	1 205 400
Overlap-Aware CORI	207	1 789 080

Table 7.3: Total Bandwidth Savings

additionally requires the MIP vectors to assess the overlap. In the experiments we use 64 permutations which already a reasonably good approximation of the desired overlap measures.

In summary, we have observed that query routing based on SONs offers a good compromise between result quality and communication overhead.

7.5 Discussion

We have presented an approach to create and maintain semantic overlay networks based on the notion of p2pDating, where peers maintain information about their friends to form a semantic network. Friends are chosen based on a variety of usefulness estimators, like overlap and semantic similarity. We have shown how we can leverage these friends networks in two different applications: in JXP, friends can be used to guide the selections of peers for the next meeting, by preferring peers with higher chances of providing useful information, over randomly chosen peers, therefore reducing the number of meetings in the network. In query routing, we show that, in comparison to state-of-art approaches, the p2pDating algorithm offers a good compromise between result quality and communication overhead.

Chapter 8

Conclusion and Outlook

We have presented the JXP algorithm for dynamically computing, in a decentralized P2P manner, authority scores when the graph is spread across many autonomous peers with arbitrarily overlapping graph fragments and the peers are a priori unaware of other peers' fragments. The algorithm works by combining local PageRank computations at each peer's local collection, with meetings among peers to exchange information. Throughout the meetings, each peer increases its knowledge about the rest of the network, which is then used in the subsequent PageRank computations, to improve the scores. We have shown through theoretical and experimental analyses that scores computed by JXP converge to the true PageRank scores that one would obtain by a centralized computation, and that after an acceptable number of meetings the JXP scores are a good approximation of the correct values. In addition, we have demonstrated how the JXP scores can be used by different P2P applications in the tasks of query routing and results ranking.

Having local pages with high authority scores can bring benefits (e.g., revenue) for a peer. Therefore, it is expected that malicious peers would try to distort the results of the JXP algorithm, by providing different (usually higher) scores for some of their local pages. To overcome such dishonest behavior we have proposed an extension of the JXP algorithm, coined *TrustJXP*, that integrates a decentralized reputation system into the JXP algorithm. Our reputation system is based on anomaly detection techniques that allow us to identify a suspicious peer based on the deviation of its behavior from some common features that constitute the usual peer profile.

We have also addressed the dynamic characteristic of P2P networks, also known as P2P churn, where peers are not static, but are rather joining and leaving the network. P2P churn has a big impact on the authority computation, since the content available in the network varies over time. We have proposed methods to adapt the JXP algorithm to work under dynamics, and a method for estimating the number of pages currently available in the network that combines multiple hash sketches in a sliding window manner.

In addition, we have also considered the problem of building Semantic Over-

lay Networks (SONs), a way of organizing networks where peers with semantically similar content are connected through an overlay network. We have proposed a new method, named *p2pDating*, for creating dynamically evolving SONs that gives more autonomy to the peers when deciding which SONs they should join. The method works by rearranging the links on the overlay networks, according to the peers' criteria of a "good" neighbor or "friend"), and using caching to remember the peers that were defined as friends. We have shown how the SONs created by our algorithm can be used in the query routing task and also by the JXP algorithm for choosing peers for a meeting.

Authority analysis can also be explored in the context of social communities. Users in a community interact in a way that results in different social graphs, and authority computation appears as a natural choice for analyzing these emerging graphs, given its success in Web graph analysis. The nature of social networks suggests a decentralized P2P setting; for example, tagged photo collections would ideally reside on the owner's computer and shared with the community by a P2P-style network, and the same holds for lists of friends and private interactions. P2P implementations of social network applications are not only well conceivable, but would actually have advantages in terms of lower vulnerability to performance bottlenecks, privacy breaches, and other forms of attacks, censorship, or manipulation. A decentralized authority computation in social networks could be easily carried out with our JXP algorithm, and it is left for future work.

List of Figures

List of Algorithms

List of Tables

Bibliography

[Abe01] Karl Aberer. P-Grid: A self-organizing access structure for P2P information systems. In Carlo Batini, Fausto Giunchiglia, Paolo Giorgini, and Massimo Mecella, editors, *Cooperative Information Systems, 9th International Conference, CoopIS 2001, Trento, Italy, September 5-7, 2001, Proceedings*, volume 2172 of *Lecture Notes in Computer Science*, pages 179–194. Springer, 2001.

[ACMHP04] Karl Aberer, Philippe Cudré-Mauroux, Manfred Hauswirth, and Tim Van Pelt. GridVine: Building internet-scale semantic overlay networks. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *The Semantic Web - ISWC 2004: Third International Semantic Web Conference,Hiroshima, Japan, November 7-11, 2004. Proceedings*, volume 3298 of *Lecture Notes in Computer Science*, pages 107–121. Springer, 2004.

[AH00] Eytan Adar and Bernardo A. Huberman. Free riding on Gnutella. *First Monday*, 5(10), 2000.

[APC03] Serge Abiteboul, Mihai Preda, and Gregory Cobena. Adaptive online page importance computation. In *Proceedings of the 12th international conference on World Wide Web*, pages 280–290. ACM Press, 2003.

[AW03] Karl Aberer and Jie Wu. A framework for decentralized ranking in web information retrieval. In Xiaofang Zhou, Yanchun Zhang, and Maria E. Orlowska, editors, *Web Technologies and Applications, 5th Asian-Pacific Web Conference, APWeb 2003, Xian, China, April 23-25, 2002, Proceedings*, volume 2642 of *Lecture Notes in Computer Science*, pages 213–226. Springer, 2003.

[AYBB07] Sihem Amer-Yahia, Michael Benedikt, and Philip Bohannon. Challenges in searching online communities. *IEEE Data Engineering Bulletin*, 30(2):23–31, 2007.

[BA99] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286:509–512, October 1999.

[BCDF06] Luca Becchetti, Carlos Castillo, Debora Donato, and Adriano Fazzone. A comparison of sampling techniques for Web characterization. In *Workshop on Link Analysis: Dynamics and Static of Large Networks (LinkKDD)*, Philadelphia, USA, August 2006.

[BCFM98] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. Min-Wise independent permutations (extended abstract). In *Symposium on Theory of Computing (STOC)*, pages 327–336, 1998.

[BCFM00] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. Min-Wise independent permutations. *Journal of Computer and System Sciences*, 60(3):630–659, 2000.

[BCK+07] Matthias Bender, Tom Crecelius, Mouna Kacimi, Sebastian Michel, Josiane Xavier Parreira, and Gerhard Weikum. Peer-to-peer information search: Semantic, social, or spiritual? *IEEE Data Engineering Bulletin*, 30(2):51–60, 2007.

[BCK+08] Matthias Bender, Tom Crecelius, Mouna Kacimi, Sebastian Michel, Thomas Neumann, Josiane Xavier Parreira, Ralf Schenkel, and Gerhard Weikum. Exploiting social relations for query expansion and result ranking. In *Proceedings of the 24th International Conference on Data Engineering Workshops, ICDE 2008, April 7-12, 2008, Cancún, México*, pages 501–506. IEEE Computer Society, 2008.

[BCMR04] John W. Byers, Jeffrey Considine, Michael Mitzenmacher, and Stanislav Rost. Informed content delivery across adaptive overlay networks. *IEEE/ACM Transactions on Networking*, 12(5):767–780, 2004.

[BCSU05] András A. Benczúr, Károly Csalogány, Tamás Sarlós, and Máté Uher. SpamRank – Fully automatic link spam detection. In *AIRWeb 2005, First International Workshop on Adversarial Information Retrieval on the Web, co-located with the WWW conference, Chiba, Japan, May 2005*, pages 25–38, 2005.

[Ber05] Pavel Berkhin. Survey: A survey on PageRank computing. *Internet Mathematics*, 2(1), 2005.

[BKK+03] Hari Balakrishnan, M. Frans Kaashoek, David R. Karger, Robert Morris, and Ion Stoica. Looking up data in P2P systems. *Communications of the ACM (CACM)*, 46(2):43–48, 2003.

[BKM+00] Andrei Z. Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet L. Wiener. Graph structure in the web. *Computer Networks*, 33(1-6):309–320, 2000.

[BLMP06] Andrei Z. Broder, Ronny Lempel, Farzin Maghoul, and Jan O. Pedersen. Efficient PageRank approximation via graph aggregation. *Information Retrieval*, 9(2):123–138, 2006.

[Blo70] Burton H. Bloom. Space/Time trade-offs in hash coding with allowable errors. *Communications ACM*, 13(7):422–426, 1970.

[BMCMA09] Adriana Budura, Sebastian Michel, Philippe Cudré-Mauroux, and Karl Aberer. Neighborhood-based tag prediction. In *Proceedings of the 6th Annual European Semantic Web Conference, ESWC 2009, Heraklion, Greece, 31 May - 4 June, 2009*, 2009.

[BMPC07] Matthias Bender, Sebastian Michel, Josiane Xavier Parreira, and Tom Crecelius. P2P web search: Make it light, make it fly (demo). In *CIDR 2007, Third Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 7-10, 2007, Online Proceedings*, pages 164–168. www.crdrdb.org, 2007.

[BMR03] Mayank Bawa, Gurmeet Singh Manku, and Prabhakar Raghavan. SETS: Search enhanced by topic segmentation. In *SIGIR 2003: Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, July 28 - August 1, 2003, Toronto, Canada*, pages 306–313. ACM, 2003.

[BMT+05a] Matthias Bender, Sebastian Michel, Peter Triantafillou, Gerhard Weikum, and Christian Zimmer. Improving collection selection with overlap awareness in P2P search engines. In Ricardo A. Baeza-Yates, Nivio Ziviani, Gary Marchionini, Alistair Moffat, and John Tait, editors, *SIGIR 2005: Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Salvador, Brazil, August 15-19, 2005*, pages 67–74. ACM, 2005.

[BMT+05b] Matthias Bender, Sebastian Michel, Peter Triantafillou, Gerhard Weikum, and Christian Zimmer. MINERVA: Collaborative P2P search. In Klemens Böhm, Christian S. Jensen, Laura M. Haas, Martin L. Kersten, Per-Åke Larson, and Beng Chin Ooi, editors, *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*, pages 1263–1266. ACM, 2005.

[BMTW06] Matthias Bender, Sebastian Michel, Peter Triantafillou, and Gerhard Weikum. Global document frequency estimation in peer-to-peer web search. In *Ninth International Workshop on the Web and Databases, WebDB 2006, Chicago, Illinois, USA, June 30, 2006*, 2006.

[BMWZ05] Matthias Bender, Sebastian Michel, Gerhard Weikum, and Christian Zimmer. The MINERVA project: Database selection in the context of P2P search. In Gottfried Vossen, Frank Leymann, Peter C. Lockemann, and Wolffried Stucky, editors, *Datenbanksysteme in Business, Technologie und Web, 11. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" (DBIS), Karlsruhe, 2.-4. März 2005*, volume 65 of *LNI*, pages 125–144. GI, 2005.

[BP98] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.

[Bro97] Andrei Z. Broder. On the resemblance and containment of documents. In *SEQUENCES: Proceedings of the Compression and Complexity of Sequences*, page 21, Washington, DC, USA, 1997. IEEE Computer Society.

[BRRT05] Allan Borodin, Gareth O. Roberts, Jeffrey S. Rosenthal, and Panayiotis Tsaparas. Link analysis ranking: Algorithms, theory, and experiments. *ACM Transactions on Internet Technology (TOIT)*, 5(1):231–297, 2005.

[BV04] Paolo Boldi and Sebastiano Vigna. The webgraph framework i: Compression techniques. In Feldman et al. [FUNW04], pages 595–602.

[BXW+07] Shenghua Bao, Gui-Rong Xue, Xiaoyuan Wu, Yong Yu, Ben Fei, and Zhong Su. Optimizing Web search using social annotations. In Williamson et al. [WZPSS07], pages 501–510.

[BYRN99] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, Boston, MA, May 1999.

[Cal00] James P. Callan. Distributed information retrieval. In *Advances in Information Retrieval*, pages 127–150. Kluwer Academic Publishers, 2000.

[Cam86] Lucien M Le Cam. *Asymptotic methods in statistical theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1986.

[CAPMN03] Francisco Matias Cuenca-Acuna, Christopher Peery, Richard P. Martin, and Thu D. Nguyen. PlanetP: Using gossiping to build content addressable peer-to-peer information sharing communities. In *12th International Symposium on High-Performance Distributed Computing (HPDC-12 2003), 22-24 June 2003, Seattle, WA, USA* [DBL03], pages 236–249.

[CCH92] James P. Callan, W. Bruce Croft, and Stephen M. Harding. The INQUERY retrieval system. In A. Min Tjoa and Isidro Ramos, editors, *Database and Expert Systems Applications, Proceedings of the International Conference in Valencia, Spain, 1992*, pages 78–83. Springer, 1992.

[CCMN00] Moses Charikar, Surajit Chaudhuri, Rajeev Motwani, and Vivek R. Narasayya. Towards estimation error guarantees for distinct values. In *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, May 15-17, 2000, Dallas, Texas, USA*, pages 268–279. ACM, 2000.

[CDK+03] Steve Chien, Cynthia Dwork, Ravi Kumar, Daniel R. Simon, and D. Sivakumar. Link evolution: Analysis and algorithms. *Internet Mathematics*, 1(3), 2003.

[CEMJ05] Geoffrey Canright, Kenth Engo-Monsen, and Márk Jelasity. Efficient and robust fully distributed power method with an application to link analysis. Technical Report UBLCS-2005-17, University of Bologna, Department of Computer Science, Bologna, Italy, 2005.

[CGM04] Arturo Crespo and Hector Garcia-Molina. Semantic overlay networks for P2P systems. In Gianluca Moro, Sonia Bergamaschi, and Karl Aberer, editors, *Agents and Peer-to-Peer Computing, Third International Workshop, AP2PC 2004, New York, NY, USA, July 19, 2004, Revised and Invited Papers*, volume 3601 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2004.

[CGS04] Yen-Yu Chen, Qingqing Gan, and Torsten Suel. Local methods for estimating PageRank values. In David A. Grossman, Luis Gravano, ChengXiang Zhai, Otthein Herzog, and David A. Evans, editors, *Proceedings of the 2004 ACM CIKM International Conference on Information and Knowledge Management, Washington, DC, USA, November 8-13, 2004*, pages 381–389. ACM, 2004.

[Cha02] Soumen Chakrabarti. *Mining the Web: Discovering Knowledge from Hypertext Data*. Morgan-Kauffman, 2002.

[CKM+08a] Tom Crecelius, Mouna Kacimi, Sebastian Michel, Thomas Neumann, Josiane Xavier Parreira, Ralf Schenkel, and Gerhard Weikum. Making SENSE: Socially enhanced search and exploration. *PVLDB*, 1(2):1480–1483, 2008.

[CKM+08b] Tom Crecelius, Mouna Kacimi, Sebastian Michel, Thomas Neumann, Josiane Xavier Parreira, Ralf Schenkel, and Gerhard Weikum. Social recommendations at work. In Myaeng et al. [MOS+08], page 884.

[CLC95] James P. Callan, Zhihong Lu, and W. Bruce Croft. Searching distributed collections with inference networks. In Edward A. Fox, Peter Ingwersen, and Raya Fidel, editors, *SIGIR'95, Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. Seattle, Washington, USA, July 9-13, 1995 (Special Issue of the SIGIR Forum)*, pages 21–28. ACM Press, 1995.

[CM00] Grace E. Cho and Carl D. Meyer. Markov chain sensitivity measured by mean first passage times. *Linear Algebra and its Applications*, 316(1–3):21–28, 2000.

[CMH+02] Ian Clarke, Scott G. Miller, Theodore W. Hong, Oskar Sandberg, and Brandon Wiley. Protecting free expression online with Freenet. *IEEE Internet Computing*, 6(1):40–49, 2002.

[Cou77] P. J. Courtois. *Decomposability: Queueing and Computer System Applications*. Academic Press, N.Y., USA, 1977.

[DBL03] *12th International Symposium on High-Performance Distributed Computing (HPDC-12 2003), 22-24 June 2003, Seattle, WA, USA*. IEEE Computer Society, 2003.

[DF03] Marianne Durand and Philippe Flajolet. Loglog counting of large cardinalities (extended abstract). In Giuseppe Di Battista and Uri Zwick, editors, *Algorithms - ESA 2003, 11th Annual European Symposium, Budapest, Hungary, September 16-19, 2003, Proceedings*, volume 2832 of *Lecture Notes in Computer Science*, pages 605–617. Springer, 2003.

[DKM+02] Stephen Dill, Ravi Kumar, Kevin S. McCurley, Sridhar Rajagopalan, D. Sivakumar, and Andrew Tomkins. Self-similarity in the web. *ACM Transactions on Internet Technology (TOIT)*, 2(3):205–223, 2002.

[DKM+06] Micah Dubinko, Ravi Kumar, Joseph Magnani, Jasmine Novak, Prabhakar Raghavan, and Andrew Tomkins. Visualizing tags over time. In Les Carr, David De Roure, Arun Iyengar, Carole A. Goble, and Michael Dahlin, editors, *Proceedings of the 15th international conference on World Wide Web, WWW 2006, Edinburgh, Scotland, UK, May 23-26, 2006*, pages 193–202. ACM, 2006.

[DKNS01] Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. Rank aggregation methods for the Web. In *Proceedings of the Tenth International World Wide Web Conference, WWW 10, Hong Kong, China, May 1-5, 2001*, pages 613–622, 2001.

[DNP05] Andrei Damian, Wolfgang Nejdl, and Raluca Paiu. Peer-sensitive ObjectRank - Valuing contextual information in social networks. In Anne H. H. Ngu, Masaru Kitsuregawa, Erich J. Neuhold, Jen-Yao Chung, and Quan Z. Sheng, editors, *Web Information Systems Engineering - WISE 2005, 6th International Conference on Web Information Systems Engineering, New York, NY, USA, November 20-22, 2005, Proceedings*, volume 3806 of *Lecture Notes in Computer Science*, pages 512–519. Springer, 2005.

[DR01] Peter Druschel and Antony I. T. Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. In *Proceedings of HotOS-VIII: 8th Workshop on Hot Topics in Operating Systems, May 20-23, 2001, Elmau/Oberbayern, Germany*, pages 75–80. IEEE Computer Society, 2001.

[FCAB00] Li Fan, Pei Cao, Jussara M. Almeida, and Andrei Z. Broder. Summary cache: A scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3):281–293, 2000.

[FKS03] Ronald Fagin, Ravi Kumar, and D. Sivakumar. Comparing top k lists. *SIAM Journal on Discrete Mathematics (SIDMA)*, 17(1):134–160, 2003.

[FLGC02] Gary William Flake, Steve Lawrence, C. Lee Giles, and Frans Coetzee. Self-organization and identification of Web communities. *IEEE Computer*, 35(3):66–71, 2002.

[FM85] Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31(2):182–209, 1985.

[FUNW04] Stuart I. Feldman, Mike Uretsky, Marc Najork, and Craig E. Wills, editors. *Proceedings of the 13th international conference on World Wide Web, WWW 2004, New York, NY, USA, May 17-20, 2004*. ACM, 2004.

[Gar79] Eugene Garfield. *Citation Indexing — Its Theory and Application in Science, Technology, and Humanities*. ISI Press, Philadelphia, USA, 1979.

[GGMT99] Luis Gravano, Hector Garcia-Molina, and Anthony Tomasic. GlOSS: Text-source discovery over the internet. *ACM Transactions on Database Systems (TODS)*, 24(2):229–264, 1999.

[HJSS06] Andreas Hotho, Robert Jäschke, Christoph Schmitz, and Gerd Stumme. Information retrieval in Folksonomies: Search and ranking. In York Sure and John Domingue, editors, *The Semantic Web: Research and Applications, 3rd European Semantic Web Conference, ESWC 2006, Budva, Montenegro, June 11-14, 2006,*

Proceedings, volume 4011 of *Lecture Notes in Computer Science*, pages 411–426. Springer, 2006.

[HRS07] Harry Halpin, Valentin Robu, and Hana Shepherd. The complex dynamics of collaborative tagging. In Williamson et al. [WZPSS07], pages 211–220.

[ISS+06] Yannis E. Ioannidis, Marc H. Scholl, Joachim W. Schmidt, Florian Matthes, Michael Hatzopoulos, Klemens Böhm, Alfons Kemper, Torsten Grust, and Christian Böhm, editors. *Advances in Database Technology - EDBT 2006, 10th International Conference on Extending Database Technology, Munich, Germany, March 26-31, 2006, Proceedings*, volume 3896 of *Lecture Notes in Computer Science*. Springer, 2006.

[JMB05] Márk Jelasity, Alberto Montresor, and Özalp Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems (TOCS)*, 23(3):219–252, 2005.

[KDG03] David Kempe, Alin Dobra, and Johannes Gehrke. Gossip-based computation of aggregate information. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 482–491. IEEE Computer Society, 2003.

[KHMG03] Sepandar D. Kamvar, Taher H. Haveliwala, Christopher D. Manning, and Gene H. Golub. Exploiting the block structure of the web for computing PageRank. Technical report, Stanford Digital Library Technologies Project, 2003.

[Kle98] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. In *Proceedings of 9th ACM-SIAM Symposium on Discrete Algorithms*, pages 668–677, 1998.

[KM04] David Kempe and Frank McSherry. A decentralized algorithm for spectral analysis. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 561–568. ACM, 2004.

[KNOT06] Panos Kalnis, Wee Siong Ng, Beng Chin Ooi, and Kian-Lee Tan. Answering similarity queries in peer-to-peer networks. *Information Systems*, 31(1):57–72, 2006.

[KS63] John George Kemeny and James Laurie Snell. *Finite Markov Chains*. Princeton, NJ: Van Nostrand, Toronto - New York, 1963.

[KSGM03] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The Eigentrust algorithm for reputation management in P2P networks. In *Proceedings of the 12th international conference*

on World Wide Web, WWW 2003, Budapest, Hungary, May 20-24, 2003, pages 640–651, 2003.

[KST74] John George Kemeny, James Laurie Snell, and Gerald L. Thompson. *Introduction to Finite Mathematics*. Prentice-Hall, Englewood Cliffs, NJ, 1974.

[Kul59] Solomon Kullback. *Information theory and statistics*. John Wiley and Sons., New York, 1959.

[Lam02] Leslie Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[LM03] Amy Nicole Langville and Carl Dean Meyer. Survey: Deeper inside pagerank. *Internet Mathematics*, 1(3), 2003.

[LM06a] Amy N. Langville and Carl D. Meyer. *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton University Press, Princeton, NJ, USA, 2006.

[LM06b] Amy N. Langville and Carl D. Meyer. Updating markov chains with an eye on Google's PageRank. *SIAM Journal on Matrix Analysis and Applications*, 27(4):968–987, 2006.

[LNBK02] David Liben-Nowell, Hari Balakrishnan, and David R. Karger. Analysis of the evolution of peer-to-peer systems. In *21st ACM Symposium on Principles of Distributed Computing (PODC)*, pages 233–242, Monterey, CA, July 2002.

[LP56] Richard C. Lewontin and T. Prout. Estimation of the number of different classes in a population. *Biometrics*, 12(2):211–233, 1956.

[Mar97] Massimo Marchiori. The quest for correct information on the web: Hyper search engines. *Computer Networks*, 29(8-13):1225–1236, 1997.

[MBTW06] Sebastian Michel, Matthias Bender, Peter Triantafillou, and Gerhard Weikum. IQN routing: Integrating quality and novelty in P2P querying and ranking. In Ioannidis et al. [ISS+06], pages 149–166.

[Mey89] Carl D. Meyer. Stochastic complementation, uncoupling markov chains, and the theory of nearly reducible systems. *Society for Industrial and Applied Mathematics (SIAM) Review*, 31(2):240–272, 1989.

[Mey00] Carl D. Meyer. *Matrix Analysis and Applied Linear Algebra*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[MGD06] Alan Mislove, Krishna P. Gummadi, and Peter Druschel. Exploiting social networks for internet search. In *Proceedings of the 5th Workshop on Hot Topics in Networks (HotNets'06)*, November 2006.

[MGM06] Sergio Marti and Hector Garcia-Molina. Taxonomy of trust: Categorizing P2P reputation systems. *Computer Networks*, 50(4):472–484, 2006.

[MOS+08] Sung-Hyon Myaeng, Douglas W. Oard, Fabrizio Sebastiani, Tat-Seng Chua, and Mun-Kew Leong, editors. *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2008, Singapore, July 20-24, 2008*. ACM, 2008.

[NT04] Nikos Ntarmos and Peter Triantafillou. SeAl: Managing accesses and data in peer-to-peer sharing networks. In Germano Caronni, Nathalie Weiler, and Nahid Shahmehri, editors, *4th International Conference on Peer-to-Peer Computing (P2P 2004), 15-17 August 2004, Zurich, Switzerland*, pages 116–123. IEEE Computer Society, 2004.

[PBMW98] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[PCD+08] Josiane Xavier Parreira, Carlos Castillo, Debora Donato, Sebastian Michel, and Gerhard Weikum. The juxtaposed approximate PageRank method for robust PageRank approximation in a peer-to-peer web search network. *VLDB Journal*, 17(2):291–313, 2008.

[PDCW07] Josiane Xavier Parreira, Debora Donato, Carlos Castillo, and Gerhard Weikum. Computing trusted authority scores in peer-to-peer web search networks. In *AIRWeb 2007, Third International Workshop on Adversarial Information Retrieval on the Web, co-located with the WWW conference, Banff, Canada, May 2007*, volume 215 of *ACM International Conference Proceeding Series*, 2007.

[PDMW06] Josiane Xavier Parreira, Debora Donato, Sebastian Michel, and Gerhard Weikum. Efficient and decentralized PageRank approximation in a peer-to-peer web search network. In Umeshwar Dayal, Kyu-Young Whang, David B. Lomet, Gustavo Alonso, Guy M. Lohman, Martin L. Kersten, Sang Kyun Cha, and Young-Kuk Kim, editors, *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*, pages 415–426. ACM, 2006.

[PGW+08] Johan A. Pouwelse, Pawel Garbacki, Jun Wang, A. Bakker, J. Yang, Alexandru Iosup, Dick H. J. Epema, Marcel J. T. Reinders, M. R. van Steen, and Henk J. Sips. TRIBLER: A social-based peer-to-peer system. *Concurrency and Computation: Practice and Experience*, 20(2):127–138, 2008.

[PMB+07] Josiane Xavier Parreira, Sebastian Michel, Matthias Bender, Tom Crecelius, and Gerhard Weikum. P2P authority analysis for social communities. In Christoph Koch, Johannes Gehrke, Minos N. Garofalakis, Divesh Srivastava, Karl Aberer, Anand Deshpande, Daniela Florescu, Chee Yong Chan, Venkatesh Ganti, Carl-Christian Kanne, Wolfgang Klas, and Erich J. Neuhold, editors, *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, pages 1398–1401. ACM, 2007.

[PMW07] Josiane Xavier Parreira, Sebastian Michel, and Gerhard Weikum. p2pDating: Real life inspired semantic overlay networks for web search. *Information Processing and Management*, 43(3):643–664, 2007.

[PMW08] Josiane Xavier Parreira, Sebastian Michel, and Gerhard Weikum. Efficiently handling dynamics in distributed link based authority analysis. In James Bailey, David Maier, Klaus-Dieter Schewe, Bernhard Thalheim, and Xiaoyang Sean Wang, editors, *Web Information Systems Engineering - WISE 2008, 9th International Conference, Auckland, New Zealand, September 1-3, 2008. Proceedings*, volume 5175 of *Lecture Notes in Computer Science*, pages 36–49. Springer, 2008.

[PNT06] Theoni Pitoura, Nikos Ntarmos, and Peter Triantafillou. Replication, load balancing and efficient range query processing in DHTs. In Ioannidis et al. [ISS+06], pages 131–148.

[PRL+07] Ivana Podnar, Martin Rajman, Toan Luu, Fabius Klemm, and Karl Aberer. Scalable peer-to-peer web retrieval with highly discriminative keys. In *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, April 15-20, 2007, The Marmara Hotel, Istanbul, Turkey*, pages 1096–1105. IEEE, 2007.

[PRU01] Gopal Pandurangan, Prabhakar Raghavan, and Eli Upfal. Building low-diameter P2P networks. In *Proceedings of the 42nd Annual IEEE Symposium on the Foundations of Computer Science (FOCS)*, pages 492–499, 2001.

[PW05] Josiane Xavier Parreira and Gerhard Weikum. JXP: Global authority scores in a P2P network. In AnHai Doan, Frank Neven, Robert McCann, and Geert Jan Bex, editors, *Proceedings of the*

Eight International Workshop on the Web & Databases (WebDB 2005), Baltimore, Maryland, USA, Collocated mith ACM SIGMOD/PODS 2005, June 16-17, 2005, pages 31–36, 2005.

[RD01] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In Rachid Guerraoui, editor, *Middleware 2001, IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, Germany, November 12-16, 2001, Proceedings*, volume 2218 of *Lecture Notes in Computer Science*, pages 329–350. Springer, 2001.

[RFH+01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard M. Karp, and Scott Shenker. A scalable content-addressable network. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 161–172, San Diego, USA, August 27-31 2001.

[RGRK04] Sean C. Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. Handling churn in a DHT. In *Proceedings of the General Track: 2004 USENIX Annual Technical Conference, June 27 - July 2, 2004, Boston Marriott Copley Place, Boston, MA, USA*, pages 127–140. USENIX, 2004.

[SBWS05] Natalia Stakhanova, Samik Basu, Johnny Wong, and Oleg Stakhanov. Trust framework for P2P networks using peer-profile based anomaly technique. In *25th International Conference on Distributed Computing Systems Workshops (ICDCS 2005 Workshops), 6-10 June 2005, Columbus, OH, USA*, pages 203–209. IEEE Computer Society, 2005.

[SCK+08a] Ralf Schenkel, Tom Crecelius, Mouna Kacimi, Sebastian Michel, Thomas Neumann, Josiane Xavier Parreira, and Gerhard Weikum. Efficient top-k querying over social-tagging networks. In Myaeng et al. [MOS+08], pages 523–530.

[SCK+08b] Ralf Schenkel, Tom Crecelius, Mouna Kacimi, Thomas Neumann, Josiane Xavier Parreira, Marc Spaniol, and Gerhard Weikum. Social wisdom for search and recommendation. *IEEE Data Engineering Bulletin*, 31(2):40–49, 2008.

[SJCO02] Luo Si, Rong Jin, James P. Callan, and Paul Ogilvie. A language modeling framework for resource selection and results merging. In *Proceedings of the 2002 ACM CIKM International Conference on Information and Knowledge Management, McLean, VA, USA, November 4-9, 2002*, pages 391–397. ACM, 2002.

[SMK+01] Ion Stoica, Robert Morris, David R. Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup

service for internet applications. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 149–160, San Diego, USA, August 27-31 2001.

[SMW+03] Torsten Suel, Chandan Mathur, Jo-Wen Wu, Jiangong Zhang, Alex Delis, Mehdi Kharrazi, Xiaohui Long, and Kulesh Shanmugasundaram. ODISSEA: A peer-to-peer architecture for scalable web search and information retrieval. In Vassilis Christophides and Juliana Freire, editors, *International Workshop on Web and Databases, San Diego, California, June 12-13, 2003*, pages 67–72, 2003.

[SPCW08] Mauro Sozio, Josiane Xavier Parreira, Tom Crecelius, and Gerhard Weikum. Good Guys vs. Bad Guys: Countering cheating in peer-to-peer authority computations over social networks. In *11th International Workshop on the Web and Databases, WebDB 2008, Vancouver, BC, Canada, June 13, 2008*, 2008.

[SSB03] Karthikeyan Sankaralingam, Simha Sethumadhavan, and James C. Browne. Distributed PageRank for P2P systems. In *12th International Symposium on High-Performance Distributed Computing (HPDC-12 2003), 22-24 June 2003, Seattle, WA, USA* [DBL03], pages 58–69.

[Ste94] William J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, Princeton, NJ, 1994.

[STS+03] Sergej Sizov, Martin Theobald, Stefan Siersdorfer, Gerhard Weikum, Jens Graupmann, Michael Biwer, and Patrick Zimmer. The BINGO! system for information portal generation and expert web search. In *CIDR 2003, First Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 5-8, 2003, Online Proceedings*, 2003.

[SW05] Ralf Steinmetz and Klaus Wehrle, editors. *Peer-to-Peer Systems and Applications*, volume 3485 of *Lecture Notes in Computer Science*. Springer, 2005.

[SYYW03] Shuming Shi, Jin Yu, Guangwen Yang, and Dingxing Wang. Distributed page ranking in structured P2P networks. In *32nd International Conference on Parallel Processing (ICPP 2003), 6-9 October 2003, Kaohsiung, Taiwan*, pages 179–186. IEEE Computer Society, 2003.

[TSW04] Christoph Tempich, Steffen Staab, and Adrian Wranik. Remindin': Semantic query routing in peer-to-peer networks based on social metaphors. In Feldman et al. [FUNW04], pages 640–649.

[TXKN03] Peter Triantafillou, Chryssani Xiruhaki, Manolis Koubarakis, and Nikos Ntarmos. Towards high performance peer-to-peer content and resource sharing systems. In *First Biennial Conference on Innovative Data Systems Research (CIDR)*, Asilomar, USA, January 5-8 2003.

[VvS03] Spyros Voulgaris and Maarten van Steen. An epidemic protocol for managing routing tables in very large peer-to-peer networks. In Marcus Brunner and Alexander Keller, editors, *Self-Managing Distributed Systems, 14th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM 2003, Heidelberg, Germany, October 20-22, 2003, Proceedings*, volume 2867 of *Lecture Notes in Computer Science*, pages 41–54. Springer, 2003.

[WA05] Jie Wu and Karl Aberer. Using a layered markov model for distributed web ranking computation. In *25th International Conference on Distributed Computing Systems (ICDCS 2005), 6-10 June 2005, Columbus, OH, USA*, pages 533–542. IEEE Computer Society, 2005.

[WD04] Yuan Wang and David J. DeWitt. Computing PageRank in a distributed internet search engine system. In Mario A. Nascimento, M. Tamer Özsu, Donald Kossmann, Renée J. Miller, José A. Blakeley, and K. Bernhard Schiefer, editors, *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004*, pages 420–431. Morgan Kaufmann, 2004.

[WZPSS07] Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy, editors. *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*. ACM, 2007.

[XBCY07] Shengliang Xu, Shenghua Bao, Yunbo Cao, and Yong Yu. Using social annotations to improve language model for information retrieval. In Mário J. Silva, Alberto H. F. Laender, Ricardo A. Baeza-Yates, Deborah L. McGuinness, Bjørn Olstad, Øystein Haug Olsen, and André O. Falcão, editors, *Proceedings of the Sixteenth ACM Conference on Information and Knowledge Management, CIKM 2007, Lisbon, Portugal, November 6-10, 2007*, pages 1003–1006. ACM, 2007.

[XL04a] Li Xiong and Ling Liu. PeerTrust: Supporting reputation-based trust for peer-to-peer electronic communities. *IEEE Transactions on Knowledge and Data Engineering*, 16(7):843–857, 2004.

[XL04b] Li Xiong and Ling Liu. PeerTrust: Supporting reputation-based trust for peer-to-peer electronic communities. *IEEE Transactions on Knowledge and Data Engineering*, 16(7):843–857, 2004.

[ZAA07] Jun Zhang, Mark S. Ackerman, and Lada A. Adamic. Expertise networks in online communities: Structure and algorithms. In Williamson et al. [WZPSS07], pages 221–230.