Monte Carlo Methods for Web Search

Monte Carlo Methods for Web Search

by Balázs Rácz

Under the supervision of
Dr. András A. Benczúr

Department of Algebra
Budapest University of Technology and Economics

Budapest
2009

Contents

Chapter 1

Introduction

One of the fastest growing sector of the software industry is that of the Internet companies, lead by the major search engines: Google, Yahoo and MSN. The importance of this field is even more emphasized by the plans of almost unprecedented magnitude that the European Union is pursuing to ease their dependence on these US-based technological firms.

The scientific and technological difficulties of this field are dominated by the mere scale: the web is estimated to contain tens to hundreds of billions of pages, with an exponential increase for over a decade and without showing any signs of that growth slowing down. At this scale, even the simplest mathematical constructs, such as a set of linear equations or a matrix inversion are turning out to be infeasible or practically unsolvable.

This thesis and the underlying publications provide solutions to certain of these scalability problems stemming from core web search engine research. The actual problems and their abstract solutions are not ours; they were described in earlier works of seminal authors of the field, generating considerable interest. Nevertheless, it was our work showing the first methods which could really scale to the size of the web without serious limitations.

A particularly important aspect of our solutions is that they are not only theoretically applicable to the web, but also very practical: they follow fairly closely and naturally fit into the architecture of a web search engine; the algorithms are parallelizable or distributed; the computational model we assumed is the one that is present in all current major data centers; and the query serving parts show characteristics very important for industrial applications, such as fault tolerance.

An important price we pay for these benefits is that out methods give approximate solutions to the abstract formulation. However, on one hand we have strict bounds on the approximation quality, on the other hand we formally prove that this is the only way to go: we give lower bounds on the resource usage of any exact method, prohibiting their application on datasets on the Web scale.

1.1 Overview

In the remaining of this chapter we define some terms, describe the architecture and introduce some methods common for the technical chapters. We will also cover related results that are not strictly connected to either problems of the remaining chapters, but rather to the general methodology we use.

In Chapter 2 we consider the problem of personalized web search, also called as personalized ranking. General web search has a static, global ranking function that the engine uses to sort the results according to some notion of relevance that depends on the query but not the user. However, relevance can easily differ from user to user, e.g. a computer geek and a history teacher may find different sites authoritative and interesting for the same query. Personalized web search allows users to specify their preference, and this preference parametrizes the ranking function. As PageRank is the most successful static ranking function, the personalized version, Personalized PageRank is of particular interest. All earlier methods for computing personalized PageRank had severe restrictions on what personalization they allowed. In our work we provided the first Personalized PageRank algorithm allowing arbitrary personalization and still scaling to the full Web. See Section 2.1 for further details and the respective chapter for our results.

In Chapter 3 we consider the problem of similarity search in massive graphs such as the web. Similarity search is not only motivated by advanced data mining algorithms requiring easily computable similarity functions such as clustering algorithms, but also by the 'Related pages' functionality of web search engines, where the user can query by example: supplying the URL of a web page of interest, the search engine replies by good quality pages on a similar topic. Traditional similarity functions stemming in social network analysis such as co-citation express the similarity of two nodes in a graph by using only the neighbors of the nodes in question. However, considering the size and depth (e.g. average diameter) of the web graph, this is just as inadequate as using degree as a ranking function. We consider the similarity function proposed by Jeh and Widom, SimRank, which is a recursive definition similar to that of PageRank. Our methods discussed in Chapter 3 provided the first algorithm that scaled beyond graphs of a few hundred thousand nodes. For further details and our results, see Section 3.1 and the respective chapter.

In the above chapters we follow the same outline: We first give approximation algorithms for the problem, analyzing the approximation quality and convergence speed. Then we claim impossibility results about non-approximation approaches, proving prohibitive space complexity. Finally we validate the methods using experiments on real Web datasets.

In the final chapter, Chapter 4 we pursue further impossibility results on similarity functions of massive graphs. We consider the decision problem: is there a pair of vertices in a graph that share a common neighborhood of a particular size? (This is equivalent to the existence of the complete bipartite graph

$K_{2,c}$ as a subgraph.) We are particularly interested in the space complexity of the problem in the data stream model: an algorithm $\mathcal{A}$ is allowed to read the set of edges of the graph sequentially, and after having one or constant many passes, it has to output the answer to the decision problem. We lower bound the temporary storage use of any such algorithm in the randomized computation model. The relevance of this problem to web search is that an algorithm $\mathcal{A}$ for the decision problem can be emulated by a search engine. During the preprocessing phase the search engine indexer can read the input a few times, producing an index database. Then the search engine query processor can answer queries only the index database, and a proper sequence of queries gives us the answer to the decision problem. Therefore any lower bound we prove on the decision problem applies either to the temporary storage requirements of the indexer, the query engine, or the index database size. A prohibitive (say, quadratic in the input size) lower bound makes it impossible to build a query engine that can feasibly serve similarity queries up to the required precision.

1.2 How to Use this Thesis?

If you are interested in a thorough introduction and motivation for the topics covered, read Chapter 1 up to this section, and Section 1 of each chapter you are interested in.

To get a general notion of the results, read the Abstract and Chapter 1 up to this section, skim through the first section and read the summary at the end of each chapter.

If you are interested in only one area, you can read any individual chapter in itself – this has been one of the main editing concepts behind this thesis. You will be referred back to the methodology sections of the Introduction where required.

To get pointers to related results, read Sections 2.1.1, 3.1.1, 4.1 and 4.2.4.

Each chapter contains bibliographical notes, which detail the original publishing times and places of the results presented in that chapter, and, in accordance with the authorship declaration, indication of authorship of each individual result presented in the chapter in case there were multiple authors. For the sake of completeness and readability we present all results including those that are attributed to co-authors of the original papers.

1.3 Introduction to the Datasets Used and the World Wide Web

The main source of information that web search engines use is naturally the World Wide Web. There are several other datasets involved for example in the computation of quality signals such as manual ratings and collections, implicit or explicit feedback from users such as click logs [76], etc. which are mostly unrelated to this thesis, except that we use data from the Open Directory Project to evaluate the quality of our similarity scores (See Section 3.6.1).

The World Wide Web is a distributed database, where certain computers connected to the Internet are serving requests initiated by clients for content hosted on those servers. The servers running a software conforming to one of the few retrieval protocols are called *web servers*. Clients trying to access a particular content first determine which server is responsible for that content, and then connect to that server directly to fetch the data. The owner of the content is responsible for running the web servers and for registering in the distributed database used for mapping of the resource locators to the actual servers.

Documents on the web are identified by *Universal Resource Locator* strings (in short URLs) such as `http://www.ilab.sztaki.hu/~bracz/index.html`. In this string `http` specifies the protocol to use for retrieving the data, `www.ilab.sztaki.hu` is the key using which the client computer looks up the server address in the Domain Name System database, and `/~bracz/index.html` is the identifier of the requested file on that particular server.

The vast majority of documents on the web use different versions of the *Hypertext Markup Language* (HTML) format. This is a rich document format used to describe formatted text with embedded media objects and cross-references between different portions of the web. The HTML files are viewed on the user's computer using a special software called *web browser*, which provides the entire user experience, from fetching the URL contents, any embedded media objects, formatting them to the screen, and providing navigational features. One of the most important navigational features are *hyperlinks*, which consist of a visual element (typically a piece of text, an image or a section of an image) that is active in the sense that the user can activate that element according to the input method used to communicate with the browser. With the most typical input method being a mouse or similar pointing device, the activation action is usually a *click* on the visual element. When activated, the hyperlink instructs the web browser to load and display another URL to the user. Using these cross-referencing links the user can navigate between different pages or different properties on the web, forming a smooth user experience of information consumption or free-time activity. In the rest of this thesis we may refer to these hyperlinks as *link*s.

One of the major challenge in the usability of the web is the vastly distributed manner it is built. Server owners can decide by themselves what content to publish, and the only way of reaching that content is to either know the exact URL under which it is published, or to accidentally find a link to it. Given that there are tens to hundreds of billions of pages and URLs, finding a particular piece of information is quite hopeless without services specifically designed to facilitate this. In the early years of the web these services were mostly hand-edited collections of links to web pages, also called *directories*. Later the significance of directories diminished in favor of *web search engines*, which allow users to find relevant content on the web by phrasing a *search query*. The search engine then matches the search query against the entire web, and returns the results to the user.

The information access method based on keyword searches in web search engines presents a dual problem. On one hand, the user has to formulate a query that is general enough so that the web page she is looking for matches it, but specific enough so that there won't be a huge amount of matches that are unrelated and irrelevant to her. Restricting the URLs to check for the expected content from 10 billion to one million or even a thousand is a big step, but still does not satisfy the user, as looking through hundreds of pages to find the relevant content is not something web users are happy to do. So the second problem is for the search engine developers: given the current size of the web, any general query will match millions of documents. Given this huge amount of matches, the search engine has to present them in such an order, that the one the user wishes to look at is among the topmost few. Of course this is an extremely underspecified problem (what is the intention of the user when she phrases a particular keyword query, and what webpages are the most corresponding to those intentions), and accordingly, the most successful web search engines have been fine-tuning their ranking algorithms for several years or even a decade. For an introduction to search engine ranking, see Section 1.8.

1.4 The Scale of the Web

Any algorithm or service aiming to process the entire web is facing a significant challenge stemming from the mere size of the web. In this section we try to quantify this size.

Due to the distributed and decentralized nature of the web, it is not easy to answer even the simplest questions about it such as

> 'How many webpages are there in the World Wide Web?'
>
> 'How many hyperlinks are there in the World Wide Web?'

To be more precise, these questions are pretty easy to answer, but the answer quickly reveals that the questions are not formulated well enough so that the answer would matter.

It is easy to see that the number of web pages and hyperlinks on the web is *infinite*. Many web sites are exposing a human-readable form of a structured database, where the HTML representation is generated by the serving machine using arguments retrieved from the URL and the underlying database. Many of these serving programs can accept arguments from an infinite domain and thus generate an infinite number of different pages.

An easy example is a calendar application, that displays some events in a certain time period, say a month. The page would typically have a link 'next month' that will lead to a different page, containing the event list of the next month. Following the 'next month' links one will find an infinite sequence of different pages.

It is just as easy to see that such an infinite sequence does not contain an infinite amount of *useful* pages, since the underlying database of information is finite. In some other cases (for example a calculator that evaluates the formula that the user inputs) it is the query page that is useful, not the individual results of the individual queries.

Therefore we should rephrase the question as

'How many *useful* webpages are there in the World Wide Web?'

and immediately conclude that we cannot give an exact answer due to the mathematically uninterpretable condition *useful*.

Instead, we could turn to a slightly more practical matter, for example the question

'How many webpages does the search engine X process?'

Unfortunately there is fairly little public information that would help us answer this question. The major search engines (Google, Yahoo, MSN) do not publish this number. The recently launched web search startup Cuil [33] claims to be the world's biggest web search engine with having crawled 186 billion pages and serving 124 billion pages ([34], data as of January 2009) in its index. Unfortunately they don't provide any reference or proof to their claims about the comparison to other search engines.

We can choose to rely on independent studies that try to estimate index size of search engines while treating them as blackboxes. A highly cited such study [61] has shown that the union of the major search engines' index exceeds 11.5 billion pages. The study was conducted in 2004, thus this number is severely outdated. A continuously updated study is published on [36], where (as of January 2009) multiple total size estimates are reported (e.g. 26 billion and 63 billion).

The general problem with blackbox-based index size estimates is that they typically need uniform sampling from within the blackbox, or relying on the statistics reported for individual queries about the number of results. Both of these methods usually require a collection of terms to be supplied to the search engine. Creating such collections from web-based datasets typically introduce some skew in the languages covered (e.g. [10] admits that the term collection only covers English). Furthermore, the statistics about the total number of matches for a query are approximations that can be seriously unreliable: For example in the case of a tiered index (for an introduction see [104]) it is quite possible that the larger but more expensive index tiers are not consulted for searches where earlier tiers return results of proper quality.

There is a general lack of recent research in the area since the search technology has long since been focusing on the relevancy of results rather than increasing index size: it is meaningless to return 2 million results to the user instead of 1 million when the user practically never looks beyond the first ten.

We can conclude, that any algorithm not able to process over 10B pages with reasonable machine resources is not acceptable for current leading search engines.

1.5 The Architecture of a Web Search Engine

Here we provide a bird's eye view of how search engines having (or at least aiming at having) entire web repositories work. Although the actual algorithmic and technical details are well-guarded secrets of the major search companies (Google, Yahoo and MSN), the outside frame of their architecture is commonly understood to be the same.

Web search engines are *centralized* from the data store point of view. They download all the content that is searchable and maintain a local copy at the data center of the search engine. The process of downloading all available web content is called *crawling*, where an automated process follows every hyperlink on pages visited so far and downloads their target pages, thereby extracting further hyperlinks, and so on. There are intricate details and non-trivial scientific and technological issues on several parts here [15, 85] which we omit as not being relevant to our subject matter such as the actual management of the URLs waiting for download, parallelizing the crawling process to hundreds of machines, parallelizing hundreds of download threads on each of those machines, deciding whether and when to re-download an already seen page to look for possible changes, etc.

The output of the crawling phase are two datasets: the first one contains all the HTML source of the web pages downloaded, while the second one (also obtainable from the first) is the *web graph*, where each web page is a vertex, and a hyperlink on page v pointing to page w is represented by a $v \to w$ arc.

These two datasets are fundamentally different, pose different problems and require a completely different class of algorithms to process even if the same question has to be solved such as similarity search based on textual data vs. similarity search in massive graphs [5]. The focus of our studies are algorithms and problems formulated over the web graph.

As the crawler progresses by downloading newly appeared pages or refreshing existing pages [26], these datasets are constantly changing. Algorithms for efficiently incorporating these changes into the search engine's current state (instead of re-computing the state from scratch for every little change) are very important and could by themselves easily fill an entire monograph. In most of our studies here we assume to have a snapshot of these datasets, while we consider the incremental update problem of our similarity search solutions in Section 3.2.3.

In a search engine these datasets are preprocessed to form the *index database* [108, 6]. This database by definition contains everything required to compute the results to a query. The index is typically not a database as in the traditional RDBMS sense, but rather a set of highly optimized specialized complex data structures to allow sub-second evaluation of user queries. Furthermore, a very important property is that the index is *distributed*: with its size being measured in tens of terabytes, and query load totaling thousands of queries per second, the only feasible supercomputing architecture for this problem is to employ a large number of parallelly operating cheap, small to medium sized machines for storing the index database and serving the queries.

This architecture is depicted on Figure 1.1.

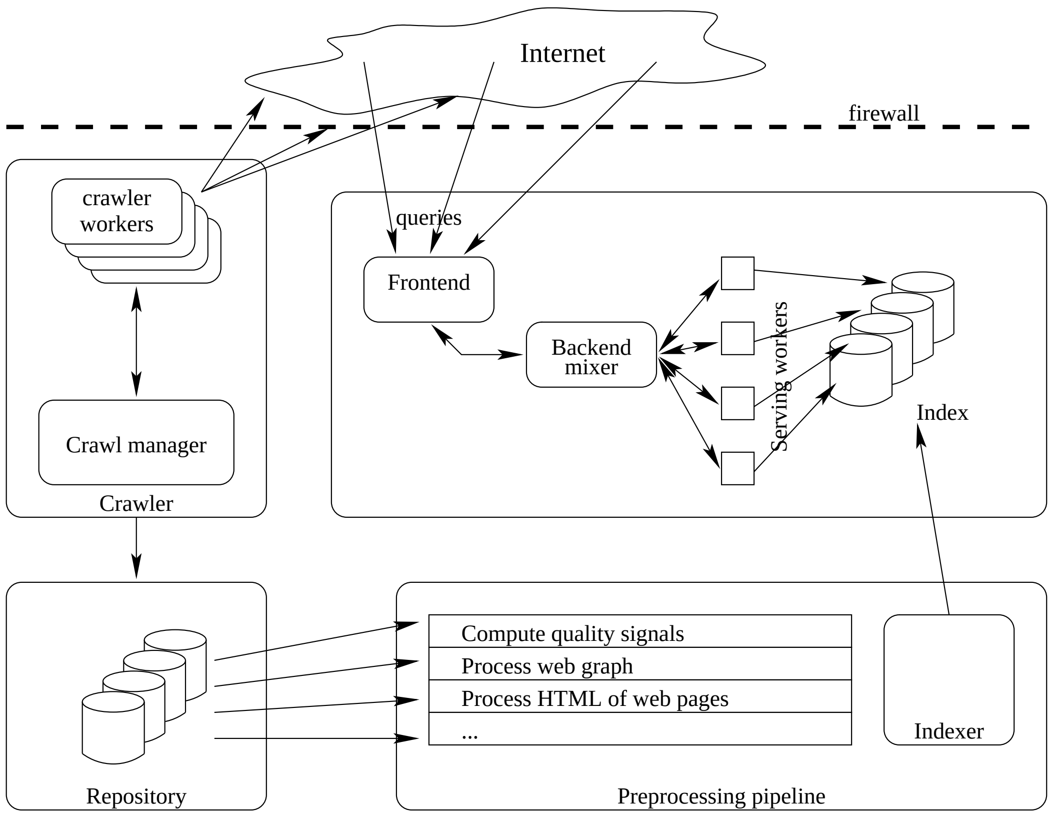


Figure 1.1: Architecture of a web search engine from a bird's eye view

1.6 The Computing Model

In this section we introduce the computing model and environment behind a typical web search engine.

When it comes to computing problems on the scale of the Web, even the best algorithmic solution is going to use supercomputing resources: the simplest task of just reading and parsing the input dataset needs thousands of hours of CPU and disk transfer time.

When it comes to supercomputing, there are in general two approaches: one is to install larger and more powerful computers, the other is to install a large amount of computers. As typically a computer of twice the capacity costs more than twice more, it is easy to see that scaling to very high computing capacity is the most cost effective if we employ a large multitude of small to medium sized computers[20]. This choice was made by Google as detailed in [13].

Reducing the cost for a given computing capacity has always been a high priority for the major web search engines. The exact methods used are well-guarded trade secrets, but there is a rack of a Google datacenter from 1999 on display in the Computer History Museum in Mountain View, California. It is a big mess: the outer frame looks like a trolley in a cafeteria that holds the returned trays of dirty dishes, the computers in there have no case and there are no rigid shelves: bare motherboards are bridging from side to side in the frame. These motherboards are slightly bent from the weight the PCB is supporting. Commodity motherboards, CPUs and disks fill the entire rack, with two or four motherboards back-to-back on the same shelf. The only neat element in the setup is the HP switch installed on the top of the rack.

According to this, the primary model used for designing algorithms we intend to run on the web is high level of parallelization[21, 95, 96]. The input dataset has to be split into chunks, and we shall be able to distribute these chunks of work to different machines, each of the capacity of a commodity PC. These machines are interconnected with some form of network, which is typically also commodity Ethernet. The machines can exchange information over this network, but this exchange is also considered to be a cost, whereas any data available in the local machine is much better accessible. Furthermore, the access to disk is also severely restricted: doing one disk seek (8 ms) sequentially for every web page in a 10 billion-page crawl on a single disk would take 2.5 years (assuming the disk drive can withstand such a utilization), so we would need about 1000 drives to complete the computation within a day. On the other hand, the same 1000-disk farm can transfer 1.73 PetaBytes of data to the CPU in a day when sequentially reading files at 20 MB/sec speed. Furthermore, if we consider the 1000 machine cluster to contain 4 GB of RAM in each computer, then we can distribute 4 TB of data among the cluster such that each computer loads a chunk of it in memory, and is able to serve random lookup queries in nanoseconds instead of in 8 ms from disk as in the previous example.

With these constraints originating from the underlying computing architecture, and the immense scale of the Web come the following strict requirements on the algorithms we are about to develop:

- **Precomputation:** The method consists of two parts: an off-line precomputation phase, which is allowed to run for about a day to precompute an index database, and an on-line query serving part, which can access only the index database, and needs to answer a query within a few hundred milliseconds.

- **Time:** The index database is precomputed within the time of a sorting operation, up to a constant factor. To serve a query the index database can only be accessed a constant number of times.

- **Memory:** The algorithms run in *external memory*: the available main memory is constant, so it can be arbitrarily smaller than the size of the web graph. In some cases we will consider semi-external-memory algorithms [91] with linear memory requirement in the number of vertices in the web graph, with a small constant factor.
- **Parallelization:** Both precomputation and query part can be implemented to utilize the computing power and storage capacity of thousands of servers interconnected with a fast local network.

1.7 Overview of Similarity Search Methods for the Web

The similarity ranking functions can be grouped into three main classes:

Text-based methods treat the web as a set of plain (or formatted) text documents, using classic methods of text database information retrieval [108].

Hybrid methods combine the text of a document with the text of the hyperlinks pointing to that document (the so-called *anchor text*) or even with the text surrounding the anchors. The intuition behind these methods is that the anchor text is typically a very good summary of the document pointed [3], since the reader of the linking page must decide whether to click on the hyperlink purely based on the anchor text and its surrounding context. This intuition was proven by various experiments [44].

The main problem with text-based and hybrid methods is that the web as a textual database is very heterogeneous, at the very least because of the many languages it is written in.

Graph-based methods restrict themselves exclusively on looking at the graph of the hyperlinks to decide the similarity of pages. These avoid the problem of heterogeneity that makes text-based methods so fragile, since the link structure is something that is very uniform across the different parts of the web, independently of the content or the language. The basic intuition behind link-based similarity methods is that a link from page A to page B can be considered a vote of page A for the relevance of page B globally as well as in the context of page A.

Although we typically study these methods isolated, searching for algorithms and evaluating quality, in practice we should always apply a combination of the aforementioned methods, running several of them and combining the scores resulting from them. This is because neither of the methods is clearly superior to all others, and a properly weighted combination has the potential to overcome the individual deficiencies.

In this thesis we primarily focus on graph-based methods, in particular on advanced recursively defined similarity functions, such as SimRank. To paint a complete picture, we quickly introduce similarity functions described in other fields of information retrieval and text database processing.

1.7.1 Text-based methods

defining similarity functions on sets of textual documents and searching for efficient evaluation methods for these is a long-studied part of classic information retrieval [6, 99]. Of the many different solutions we recall a few major approaches here.

Vector-space based document model[14, 99, 108]. Consider all the words appearing in the set of documents, and assign integers to them $1..m$. Then we can treat each document a set (or multi-set) of integers, which can be represented using the characteristic vector of the set over R^m. The individual elements of this vector could be 0 or 1 as in the basic definition, or it could be weighted by the frequency of occurrence of the word in the document, its visual style, and potentially with the selectivity (infrequency) of the word in the entire document set. This is called the TF-IDF weighting (term frequency, inverse document frequency [98]). Then we can define the similarity of individual document by the similarity of their vectors, for example with the scalar product of the vectors. Doing efficient searches in such high dimensional spaces can be achieved with advanced multi-dimensional search trees [57].

Singular decomposition methods [39, 59]. These methods combine the above described vector space model with a well-known statistical method. The main objective is to reduce the dimension of the vector space in order to gain speed and accuracy by removing redundancy from the underlying dataset. We will approximate the document-word incidence matrix, or the matrix of the document vectors with a matrix of low rank. We can achieve this by computing the singular decomposition of the incidence matrix and taking the coordinates represented by the first k singular vectors. We can use similar multi-dimensional search structures as in the pure vector space models. The main advantage of these methods is that the singular decomposition removes redundancy inherent to the language (by e.g. representing synonyms and different cases with vectors very close to each other). The major drawback is that we currently have no practical methods to compute the singular decomposition for billions of documents, therefore these methods are infeasible on the scale of the Web.

Fingerprint-based methods [14, 22]. Here we consider documents as sets of words again, and define the similarity of two documents by the Jaccard-coefficient of the representing sets:

$$\mathsf{sim}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

This in itself does not give a very practical method, but we can give a high-performing approximation algorithm. We'll assign to each document a random *fingerprint* so that the similarity of a pair of fingerprints gives an unbiased estimate of the similarity of their respective documents. The we generate N independent sets of fingerprints, which we then query using traditional indexing methods [108].

To generate a fingerprint let's take a random permutation σ over the integers $1..m$, which correspond to the words in our vector space model. We define the fingerprint of a document to be the the identifier of the word that has the smallest value under this permutation:

$$\mathsf{fp}(A) = \operatorname*{argmin}_{i \in A} \sigma(i)$$

it is easy to see that the fingerprint of two documents A and B will be the same with probability $\mathsf{sim}(A, B)$. This method is called min-hash fingerprinting. Notice that we don't actually require a random permutation, σ can be an arbitrary random hash function where for every set the minimum over that set falls on a uniformly distributed element. Giving small families of functions that satisfy this requirement is an interesting mathematical problem [29].

An interesting further application of this technique is not only to measure the resemblance of documents, but also the containment of them [22].

1.7.2 Hybrid methods

Hybrid methods are text-based methods that treat the text of hyperlinks and potentially the surrounding text specially.

It is typical in text-based search engines to attach the text of anchors to the linked document. A quite remarkable incident due to this method happened a few years ago when a popular search engine presented the home page of a widely used (but not so unanimously popular) software company for the search query "go to hell" as the first result. These methods try to utilize that anchor text gives a good summary of the document the link points to [3], and such summaries are very useful for matching a query text against.

There are many parameters and techniques that we can use to define and refine hybrid methods:

- Do we use the text of the document or exclusively the text of the anchors?
- Do we use the text of the anchors only, or the text surrounding the anchors as well?

- How much of the surrounding text do we use? Shall it be a constant, defined by syntactic boundaries (e.g. visual elements) or semantic boundaries (linguistic methods, sentence boundary, etc.)?
- If we use the surrounding text, how do we weight it?

In addition we have to consider the parameters of the underlying text-based methods as well (e.g. linguistic methods such as stemming, synonyms, etc.).

To select from these multitude of options we can only rely on extensive experimentation. Experimental results can vary highly depending on the underlying dataset, therefore the experimental tuning phase has to be repeated essentially for all applications. A very detailed and thorough experimental evaluation over the above mentioned parameters was performed by Haveliwala et al [65].

1.7.3 Simple graph-based methods

The first set of graph-based similarity search methods stem from sociometry, which analyzes social networks using mathematical methods. The task of sociometry that most resembles the Web is the analysis of scientific publication networks, in particular the references between scientific publications. Often the names of these methods are stemming from these early applications.

An overview of these methods and experimental evaluation is found in [35].

Co-citation [56]. The co-citation of vertices u, v is $|I(u) \cap I(v)|$, i.e. the number of vertices that are linking to both u and v. As necessary co-citation can be normalized into the range $[0, 1]$ by taking the Jaccard-coefficient of the referring sets: $\frac{|I(u) \cap I(v)|}{|I(u) \cup I(v)|}$.

Bibliographic coupling [80] is the dual definition of co-citation, operating on the out-links instead of the in-links. The main drawback of applying bibliographic coupling (or any other out-link-based method) is that the out-links of a page are set by the author of the page, and thus are susceptible to spam.

Amsler [4]. To fully utilize the neighborhoods in the citation graph Amsler considered two papers related in the following conditions: (1) if there is a third paper referring to both of them (co-citation), or (2) if they both refer to a third paper (bibliographic coupling), or (3) if one refers to a third referring to the other. Based on these the formal definition of Amsler similarity is

$$\frac{|(I(u) \cup O(u)) \cap (I(v) \cup O(v))|}{|(I(u) \cup O(u)) \cup (I(v) \cup O(v))|}$$

This coincides with the Jaccard-coefficient-based similarity function on the undirected graph.

The main problem with these purely graph-based methods is that they operate on the neighborhood in the graph up to distance 1 or 2, which is way too little to consider in case of the Web. This is why the advanced iteratively defined graph-based similarity functions are so much important in the case of the Web Search.

However, before going into details about iterative similarity functions it will be useful to first take a look at the basic definitions of two well-known algorithms for Web Search Ranking.

1.8 Introduction to Web Search Ranking

Following the dynamic expansion of the World Wide Web, by the end of the nineties it became a widely believed theory that whatever you're looking for, it is surely available on the Internet. The only problem is how to find it. With the development of web search technology and the accessibility of comprehensive web indexes it became a solved problem to find the set of pages that contain a set of search terms. With the scale of the web however, almost any typical query yields ten thousand to millions of result pages, from which it is impossible for the user to select the pages by hand that contain the searched information. With the dynamic expansion of the Web the main concern of web search engines became relevancy instead of comprehensiveness.

A typical user looks at most at the top five results (the above-the-fold part of the results page) when issuing a search query. If the required information is not found within a click or two, then it constitutes a bad user experience. Therefore it is absolutely crucial for the search engine to sort the result pages and present it in an order to the user that contains the sought target page in the top five results. In order to achieve this a combination of local and global methods are employed.

Local methods come from the text information retrieval studies and try to determine how well the actual query matches the actual document: it looks at where the search terms are found on the page, how far away from each other these hits are, whether they are in highlighted text, the title or URL of the page, or, on the other hand, maybe completely invisible (tiny text, metadata, white text on white background, etc).

The global methods try to establish some notion of global quality or relevance of pages. The global relevance does not depend on the query asked and is typically precomputed and incorporated into the index.

One of the main source of information for the global relevance ranking is the hyperlink structure of the Web. Since our thesis focuses on graph-based methods for Web Information Retrieval, we'll discuss some of them in greater detail here.

The most simple global relevance signal one can extract from the hyperlink graph is the *(in-)degree ranking*, where we rank the pages according to how many hyperlinks point to them. If we assume that incoming hyperlinks are each the opinion (or *vote*) of an independent person or webmaster for the quality of the pointed page, then this should be a fairly good quality and popularity metric.

Unfortunately the above assumption is not correct. Since using web search engines have become the primary way of accessing information on the Web, the wide popularity of this information access method has created a tight bound between the rank of a website on a web search engine and the visitors it will get. If there is any commercial intent of the website (or if it is serving ads), the visitors turn into money, thus there is a strong financial incentive for the website to try to trick the search engine into showing the website higher than its actual relevance and popularity. Therefore any method in web search ranking that can be adversely influenced with little cost to show certain pages higher (or lower) in the ranking is not very useful in practice.

Degree ranking is unfortunately pretty easy to confuse: one has nothing else to do than publish a large number of fake web pages with no actual content, but links pointing to the real target page. Degree ranking will take these pages into consideration and happily boost the rank of the malicious webmaster. Unfortunately this attack can be implemented very cheaply, and thus degree ranking is not usable.

As the search engine spamming became a widely used technique, the developers of search engines and the scientific community turned to creating more sophisticated algorithms, where the rank of a particular webpage depends on a large fraction of the web and thus is not influenceable with isolated sets of spam pages.

1.8.1 The HITS ranking algorithm

Kleinberg [82] in his famous *hub-authority ranking* scheme assigns two numbers to each web page: a *hub score* and an *authority score*.

This scheme tries to grasp the typical web browsing pattern of the nineties: in order to explore a particular topic, one first tried to find a good hub, a link collection, from where one could get to many pages with authoritative information in that particular topic.

From there it comes a natural definition: the more authorities pages a link collection lists, the better that link collection is; on the other hand the more good link collections list a page, the higher quality the information on that page is (i.e., the more authorities that page is).

According to this, the hub score of a page will be the sum of the authority scores of the pages it points to, whereas the authority score of a page will be the sum of the hub scores of the pages that point to them. Of course we will need to normalize the vectors of these scores.

Definition 1. HITS ranking is the limit of the following iteration, starting from the all-1 vectors:

$$\begin{aligned} a_0(v) &= \textstyle\sum_{u\in I(v)} h(u) \\ a(v) &= \tfrac{a_0(v)}{\|a_0\|} \\ h_0(u) &= \textstyle\sum_{v\in O(u)} a(v) \\ h(u) &= \tfrac{h_0(u)}{\|h_0\|} \end{aligned}$$

The hub score of a page v is $h(v)$, the authority score is $a(v)$.

Although the original idea behind HITS is definitely plausible, the mathematical formulation has several deficiencies. It is easy to see that that the hub and authority vectors correspond to the first left and right singular vectors in the singular value decomposition of the adjacency matrix A of the web. If we consider the other singular vector pairs of the adjacency matrix, we'll find a set of orthogonal topics, each fulfilling the HITS equations and thus the original intent. If we rank by the iteration limit, we'll rank according to the single dominant topic, the topic that has the highest eigenvalue in the adjacency matrix. All other topics are ignored, and thus any search query that does not belong to the dominant topic will not benefit from HITS, since there will be no ranking established among the results.

By the same argument injecting a suitably large complete bipartite subgraph in the Web with no out-links will replace the dominant topic and thus attract all the weight in the HITS ranking scheme.

Due to these weaknesses HITS is not used in practice for ranking.

1.8.2 The PageRank algorithm

This ranking algorithm was designed by the founders, initial developers and current presidents of the popular Web search engine Google [60], Larry Page and Sergey Brin [21, 95]. PageRank defines the ranking with similar recursive equations as HITS, but assigning only a single PageRank score to each page. We can think of it as a recursive extension (or refinement) of the in-degree ranking by defining the PageRank of a page to be the normalized sum of the PageRank values of the pages linking to it. This definition does not have a unique solution if the graph is not strongly connected, thus PageRank extends this idea with a correction factor that gives a uniform starting and base weight to each page.

Definition 2 (PageRank vector)**.** The PageRank vector of a directed graph is the solution of the following linear equation system:

$$\mathrm{PR}(v) = c\frac{1}{V} + (1-c)\sum_{u\in I(v)} \frac{\mathrm{PR}(u)}{\deg(u)}$$

where V is the number of nodes of the graph, $c \in (0, 1)$ is a constant, and $I(v)$ is the set of nodes linking to v.

The constant c defines the mixing of the uniform starting point and is typically chosen to be around 0.1–0.2. The PageRank vector can be considered to be the eigenvector of the (slightly changed) adjacency matrix and accordingly, a straightforward computation method is by iteration. The parameter c greatly influences the convergence speed of the iteration, also in theory, but in practice the straightforward iteration converges in 30-50 steps, way faster than one would expect from the chosen value of c.

As an alternative definition of PageRank we'll introduce the *random surfer model*. The random surfer starts from a uniformly selected page. Then when visiting a page v, with probability $1-c$ the random browser follows a uniformly chosen out-link of the page v. Otherwise, with probability c the random surfer gets bored with the current browsing and continues at another uniformly selected page. The PageRank value of a page v is the fraction of the time the random browser is looking at the page v during an infinitely long browsing session. In other words, the normalized adjacency matrix is mixed with the normalized all-1 matrix with weights $1 - c$ and c, and the resulting transition matrix is used to drive a Markov-chain on the web pages. The stationary distribution of this Markov-chain is the PageRank vector.

1.9 Iterative Link-Based Similarity Functions

Similarly to how HITS and mainly PageRank revolutionized Web search ranking quality, we have strong reasons to believe that for the complexity of the Web the ranking power of the similarity search functions inherited from social network analysis (see Section 1.7.3) can be greatly superseded by their counterparts that take into account the deeper structure of the graph.

Two widely studied similarity functions stem from the above discussed ranking methods.

1.9.1 The Companion similarity search algorithm

The Companion similarity function was introduced by Jeff Dean and Monika Henzinger in 1999 [38] based on the ideas from the HITS ranking algorithm. Their method searches for the most similar pages to a query page v and assign similarity scores to them:

1. Using heuristics we identify the subgraph representing the neighborhood of page v. This includes p_i in-neighbors of v; p_{io} out-neighbors of each of them; p_o out-neighbors of v and p_oi in-neighbors of each of them. The four parameters are tuned manually, and wherever the neighbor set exceeds the respective parameter value, we select a uniform random p_{resp} element subset.

2. We take the subgraph spanned by the selected nodes. We merge the nodes that have their out-neighborhood overlapping by 95%.

3. We run the HITS algorithm on the derived graph. We employ a slight modification to the original algorithm that handles multiple edges between nodes with a proper weighting.

4. Finally we use the authority scores of the nodes to rank them.

Apart from the extra cleaning steps this is in practice a local version of the HITS ranking algorithm, computing hub- and authority-scores in the small neighborhood of the query page v. Since the neighborhood is local to the node in question, this method does not suffer from the topical drift of the global HITS ranking algorithm.

Unfortunately this method cannot be applied to the entire web graph based on our computing model (see Section 1.6). The main problem is that in order to select the subgraph spanned by this neighborhood we need to make many random accesses into the database storing the web graph. This is feasible only if we have memory proportional to the size of the complete web graph, which is often prohibitively expensive. Recent results have shown methods that are able to store a significant portion of the web graph in memory using sophisticated compression methods [2, 16], but performing random access on compressed data is also costly.

1.9.2 The SimRank similarity function

The SimRank similarity function, which is one of the central subject of study in Chapter 3, was introduced by Glen Jeh and Jennifer Widom in 2002 [74]. SimRank is the recursive refinement of the co-citation function, similarly as PageRank is the recursive refinement of in-degree ranking.

The key idea behind SimRank is the following:

> The similarity of a pair of web-pages is the average similarity of the pages linking to them.

Definition 3 (SimRank equations)**.**

$$\begin{aligned}
\mathsf{sim}(u,u) &= 1\\
\mathsf{sim}(u,v) &= 0, \text{ if } u \neq v \text{ and } (I(u)=\emptyset \text{ or } I(v)=\emptyset)\\
\mathsf{sim}(u,v) &= \tfrac{c}{|I(u)|\cdot|I(v)|}\textstyle\sum_{u'\in I(u)}\sum_{v'\in I(v)} \mathsf{sim}(u',v'), \text{ otherwise,}
\end{aligned}$$

where $c \in (0,1)$ is a constant, u, v are nodes in the graph, and $I(u)$ is the set of nodes linking to u.

For V nodes this means a linear equation system with V^2 equation and V^2 variables. Since $c < 1$ the norm of the equation matrix is less, than 1 and it is easy to see that the equation system has a unique solution. In theory it is fairly easy to come up with this solution, since from an arbitrary starting point an iteration over the equation system will converge to the solution exponentially.

In practice nevertheless, just in order to be able to do one iteration on the equation system we would need to store the values of all the variables. With a web graph of a mere 1 billion nodes this means 10^{18} values to store, which is a completely unrealistic requirement: we would need billions of hard drives of the highest capacity to date. Even with pruning during the iteration (rounding all values smaller than a threshold to zero [74]) the naive iteration-based method is only applicable to graphs of a few hundred thousand vertices.

Chapter 2

Personalized Web Search

2.1 Introduction

The idea of topic sensitive or personalized ranking appears since the beginning of the success story of Google's PageRank [21, 95] and other hyperlink-based quality measures [82, 19]. Topic sensitivity is either achieved by precomputing modified measures over the entire Web [63] or by ranking the neighborhood of pages containing the query word [82]. These methods however work only for restricted cases or when the entire hyperlink structure fits into the main memory.

In this chapter we address the computational issues [63, 75] of personalized PageRank [95]. Just as all hyperlink based ranking methods, PageRank is based on the assumption that *the existence of a hyperlink* $u \to v$ *implies that page* u *votes for the quality of* v. Personalized PageRank (PPR) enters user preferences by assigning more importance to edges in the neighborhood of certain pages at the user's selection. Unfortunately the naive computation of PPR requires a power iteration algorithm over the entire web graph, making the procedure infeasible for an on-line query response service.

Earlier personalized PageRank (PPR) algorithms restricted personalization to a few topics [63], a subset of popular pages [75] or to hosts [77]; see [66] for an analytical comparison of these methods. The state of the art Hub Decomposition algorithm [75] can answer queries for up to some 100,000 personalization pages, an amount relatively small even compared to the number of categories in the Open Directory Project [94].

In contrast to earlier PPR algorithms, we achieve *full personalization*: our method enables on-line serving of personalization queries for *any* set of pages. We introduce a novel, scalable Monte Carlo algorithm that precomputes a compact database. As described in Section 2.2, the precomputation uses simulated random walks, and stores the ending vertices of the walks in the database. PPR is estimated on-line with a few database accesses.

The price that we pay for full personalization is that our algorithm is randomized and less precise than power-iteration-like methods; the formal analysis of the error probability is discussed in Section 2.3. We theoretically and experimentally show that we give sufficient information for all possible personalization pages while adhering to the strong implementation requirements of a large-scale web search engine.

According to Section 2.4, some approximation seems to be inavoidable since the exact personalization requires a database as large as $\Omega(V^2)$ bits in worst case over graphs with V vertices. Though no worst case theorem applies to the webgraph or one particular graph, the theorems show the nonexistence of a general exact algorithm that computes a linear sized database on any graph. To achieve full personalization in future research, one must hence either exploit special features of the webgraph or relax the exact problem to an approximate one as in our scenario. Of independent interest is another consequence of our lower bounds that there is indeed a large amount of information in personalized PageRank vectors since, unlike uniform PageRank, it can hold information of size quadratic in the number of vertices.

In Section 2.5 we experimentally analyze the precision of approximation on the Stanford WebBase graph and conclude that our randomized approximation method provides sufficiently good approximation for the top personalized PageRank scores.

Though our approach might give results of inadequate precision in certain cases (for example for pages with large neighborhood), the available personalization algorithms can be combined to resolve these issues. For example we can precompute personalization vectors for certain topics by topic-sensitive PR [63], for popular pages with large neighborhoods by the hub skeleton algorithm [75]), and use our method for those millions of pages not covered so far. This combination gives adequate precision for most queries with large flexibility for personalization.

2.1.1 Related Results

We compare our method with known personalized PageRank approaches as listed in Table 2.1 to conclude that our algorithm is the first that can handle on-line personalization on arbitrary pages. Earlier methods in contrast either restrict personalization or perform non-scalable computational steps such as power-iteration in query time or quadratic disk usage during the precomputation phase. The only drawback of our algorithm compared to previous ones is that its approximation ratio is somewhat worse than that of the power iteration methods.

The first known algorithm [95] (Naive in Table 2.1) simply takes the personalization vector as input and performs power iteration at query time. This approach is clearly infeasible for on-line queries. One may precompute the power iterations for a well selected set of personalization vectors as in the Topic Sensitive PageRank [63]; however full personalization in this case requires $t = V$ precomputed vectors yielding a database of size V^2 for V web pages. The current size $V \approx 10^9 - 10^{10}$ hence makes full personalization infeasible.

The third algorithm of Table 2.1, BlockRank [77] restricts personalization to hosts. While the algorithm is attractive in that the choice of personalization is fairly general, a reduced number of power iterations still need to be performed at query time that makes the algorithm infeasible for on-line queries.

The remarkable Hub Decomposition algorithm [75] restricts the choice of personalization to a set H of top ranked pages. Full personalization however requires H to be equal to the set of all pages, thus V^2 space is required again. The algorithm can be extended by the Web Skeleton [75] to lower estimate the personalized PageRank vector of arbitrary pages by taking into account only the paths that goes through the set H. Unfortunately, if H does not overlap the few-step neighborhood of a page, then the lower estimation provides poor approximation for the personalized PageRank scores.

The Dynamic Programming approach [75] provides full personalization by precomputing and storing sparse approximate personalized PageRank vectors. The key idea is that in a k-step approximation only vertices within distance k have nonzero value. However the rapid expansion of the k-neighborhoods increases disk requirement close to V^2 after a few iterations that limits the usability of this approach. Furthermore, a possible external memory implementation would require significant additional disk space. The space requirements of Dynamic Programming for a single vertex is given by the average neighborhood size Neighb(k) within distance k as seen in Fig. 2.1. The average size of the sparse vectors exceeds 1000 after $k \geq 4$ iterations, and on average 24% of all vertices are reached within $k = 15$ steps[1]. For example the disk requirement for $k = 10$ iterations is at least $\mathrm{Neighb}(k) \cdot V = 1,075,740 \cdot 80\mathrm{M} \approx 344$ Terabytes. Note that the best upper bound of the approximation is still $(1-c)^{10} = 0.85^{10} \approx 0.20$ measured by the L_1-norm.

[1]The neighborhood function was computed by combining the size estimation method of [31] with our external memory algorithm discussed in [52].

Method	Personalization	Limits of scalability	Postitive aspects	Negative aspects
Naive [95]	any page	power iteration in query-time		infeasible to serve on-line personalization
Topic-Sensitive PageRank [63]	restricted to linear combination of t topics, e.g. $t = 16$	$t \cdot V$ disk space required	distributed computing	
BlockRank [77]	restricted to personalize on hosts	power iteration in query-time	reduced number of power iterations, distributed computing	infeasible to serve on-line personalization
Hub Decomposition [75]	restricted to personalize on the top H ranked pages, practically $H \leq 100$K	H^2 disk space required, H partial vectors aggregated in query time	compact encoding of H personalized PR vectors	
Basic Dynamic Programming [75]	any page	$V \cdot \text{Neighb}(k)$ disk space required for k iterations, where $\text{Neighb}(k)$ grows fast in k		infeasible to perform more than $k = 3, 4$ iterations within reasonable disk size
Fingerprint (this paper)	any page	no limitation	linear-size $(N \cdot V)$ disk required, distributed computation	lower precision approximation

Table 2.1: Analytical comparison of personalized PageRank algorithms. V denotes the number of all pages.

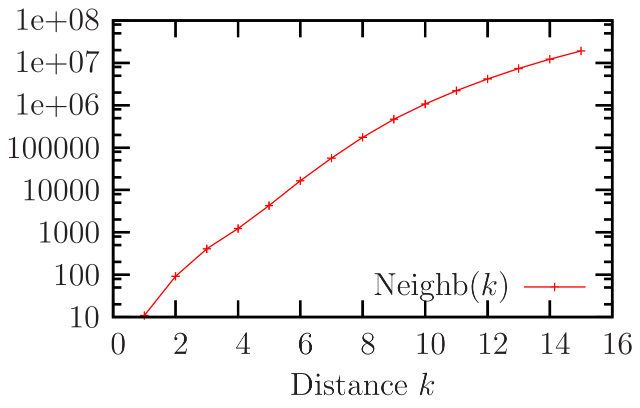


Figure 2.1: The neighborhood function measured on the Stanford WebBase graph of 80M pages.

We believe that the basic Dynamic Programming could be extended with some pruning strategy that eliminates some of the non-zero entries from the approximation vectors. However, it seems difficult to upper bound the error caused by the pruning steps, since the small error caused by a pruning step is distributed to many other approximation vectors in subsequent steps. Another drawback of the pruning strategy is that selecting the top ranks after each iteration requires extra computational efforts such as keeping the intermediate results in priority queues. In contrast, our fingerprint based method tends to eliminate low ranks inherently, and the amount of error caused by the limited storage capactity can be upper bounded formally.

Now, we briefly review some algorithms that solve the scalability issue by fingerprinting or sampling for applications that are different from personalized web search. For example, [96] applies probabilistic counting to estimate the neighborhood function of the Internet graph, [31] estimates the size of transitive closure for massive graphs occurring in databases, and [51, 52] approximates link-based similarity scores by fingerprints. Apart from graph algorithms, [22] estimates the resemblance and containment of textual documents with fingerprinting.

Random walks were used before to compute various web statistics, mostly focused on sampling the web (uniformly or according to static PR) [70, 97, 11, 68], but also for calculating page decay [12] and similarity values [51, 52].

The lower bounds of Section 2.4 show that precise PPR requires significantly larger database than Monte Carlo estimation does. Analogous results with similar communication complexity arguments were proved in [69] for the space complexity of several data stream graph algorithms.

2.1.2 Preliminaries

In this section we introduce notation, recall definitions and basic facts about PageRank. Let $\mathcal{V}$ denote the set of web pages, and $V = |\mathcal{V}|$ the number of pages. The directed graph with vertex set $\mathcal{V}$ and edges corresponding to the hyperlinks will be referred to as the *web graph.* Let A denote the adjacency matrix of the webgraph with normalized rows and $c \in (0, 1)$ the *teleportation probability.* In addition, let $\vec{r}$ be the so called *preference vector* inducing a probability distribution over $\mathcal{V}$. *PageRank* vector $\vec{p}$ is defined as the solution of the following equation [95]

$$\vec{p} = (1 - c) \cdot \vec{p}A + c \cdot \vec{r} \ .$$

If $\vec{r}$ is uniform over $\mathcal{V}$, then $\vec{p}$ is referred to as the *global PageRank vector.* For non-uniform $\vec{r}$ the solution $\vec{p}$ will be referred to as *personalized PageRank vector of* $\vec{r}$ denoted by $\mathrm{PPV}(\vec{r})$. The special case when for some page u the u^{th} coordinate of $\vec{r}$ is 1 and all other coordinates are 0, the PPV will be referred to as the *individual PageRank vector* of u denoted by $\mathrm{PPV}(u)$. We will also refer to this vector as the *personalized PageRank vector of* u. Furthermore the v^{th} coordinate of $\mathrm{PPV}(u)$ will be denoted by $\mathrm{PPV}(u, v)$.

Theorem 4 (Linearity, [63])**.** *For any preference vectors $\vec{r}_1$, $\vec{r}_2$, and positive constants α_1, α_2 with $\alpha_1 + \alpha_2 = 1$ the following equality holds:*

$$\mathrm{PPV}(\alpha_1 \cdot \vec{r}_1 + \alpha_2 \cdot \vec{r}_2) = \alpha_1 \cdot \mathrm{PPV}(\vec{r}_1) + \alpha_2 \cdot \mathrm{PPV}(\vec{r}_2).$$

Linearity is a fundamental tool for scalable on-line personalization, since if PPV is available for some preference vectors, then PPV can be easily computed for any combination of the preference vectors. Particularly, for full personalization it suffices to compute individual $\mathrm{PPV}(u)$ for all $u \in \mathcal{V}$, and the individual PPVs can be combined on-line for any small subset of pages. Therefore in the rest of this chapter we investigate algorithms to make all individual PPVs available on-line.

The following statement will play a central role in our PPV estimations. The theorem provides an alternate probabilistic characterization of individual PageRank scores.[2]

Theorem 5 ([75, 50])**.** *Suppose that a number L is chosen at random with probability $\Pr\{L = i\} = c(1 - c)^i$ for $i = 0, 1, 2, \ldots$ Consider a random walk starting from some page u and taking L steps. Then for the v^{th} coordinate $\mathrm{PPV}(u, v)$ of vector $\mathrm{PPV}(u)$*

$$\mathrm{PPV}(u, v) = \Pr\{\textit{the random walk ends at page } v\}.$$

2.2 Personalized PageRank algorithm

In this section we will present a new Monte-Carlo algorithm to compute approximate values of personalized PageRank utilizing the above probabilistic characterization of PPR. We will compute approximations of each of the

[2]Notice that this characterization slightly differs from the random surfer formulation [95] of PageRank.

PageRank vectors personalized on a single page, therefore by the linearity theorem we achieve full personalization.

Our algorithm utilizes the simulated random walk approach that has been used recently for various web statistics and IR tasks [12, 51, 11, 70, 97].

Definition 6 (Fingerprint path)**.** *A* fingerprint path *of a vertex u is a random walk starting from u; the length of the walk is of geometric distribution of parameter c, i.e., after each step the walk takes a further step with probability $1 - c$ and ends with probability c.*

Definition 7 (Fingerprint)**.** *A* fingerprint *of a vertex u is the ending vertex of a fingerprint path of u.*

By Theorem 5 the fingerprint of page u, as a random variable, has the distribution of the personalized PageRank vector of u. For each page u we will calculate N independent fingerprints by simulating N independent random walks starting from u and approximate $\mathrm{PPV}(u)$ with the empirical distribution of the ending vertices of these random walks. These fingerprints will constitute the *index database*, thus the size of the database is $N \cdot V$. The output ranking will be computed at query time from the fingerprints of pages with positive personalization weights using the linearity theorem.

To increase the precision of the approximation of $\mathrm{PPV}(u)$ we will use the fingerprints that were generated for the neighbors of u, as described in Section 2.2.3.

The challenging problem is how to scale the indexing, i.e., how to generate N independent random walks for each vertex of the web graph. We assume that the edge set can only be accessed as a data stream, sorted by the source pages, and we will count the database scans and total I/O size as the efficiency measure of our algorithms. Though with the latest compression techniques [17] the entire web graph may fit into main memory, we still have a significant computational overhead for decompression in case of random access. Under such assumption it is infeasible to generate the random walks one-by-one, as it would require random access to the edge-structure.

We will consider two computational environments here: a single computer with constant random access memory in case of the *external memory model*, and a *distributed system* with tens to thousands of medium capacity computers [37]. Both algorithms use similar techniques to the respective I/O efficient algorithms computing PageRank [30].

As the task is to generate N independent fingerprints, the single computer solution can be trivially parallelized to make use of a large cluster of machines, too. (Commercial web search engines have up to thousands of machines at their disposal.) Also, the distributed algorithm can be emulated on a single machine, which may be more efficient than the external memory approach depending on the graph structure.

Algorithm 2.2.1 Indexing (external memory method)

N is the required number of fingerprints for each vertex. The array Paths holds pairs of vertices (u, v) for each partial fingerprint in the calculation, interpreted as (PathStart,PathEnd). The teleportation probability of PPR is c. The array Fingerprint$[u]$ stores the fingerprints computed for a vertex u.

```
for each web page $u$ do
  for $i := 1$ to $N$ do
    append the pair $(u, u)$ to array Paths /*start $N$ fingerprint paths from
    node $u$: initially PathStart=PathEnd= $u$*/
  Fingerprint$[u] := \emptyset$
while Paths $\neq \emptyset$ do
  sort Paths by PathEnd /*use an external memory sort*/
  for all $(u, v)$ in Paths do /*simultaneous scan of the edge set and Paths*/
    $w :=$ a random out-neighbor of $v$
    if random()$< c$ then /*with probability $c$ this fingerprint path ends
    here*/
      add $w$ to Fingerprint$[u]$
      delete the current element $(u, v)$ from Paths
    else /*with probability $1 - c$ the path continues*/
      update the current element $(u, v)$ of Paths to $(u, w)$
```

2.2.1 External memory indexing

We will incrementally generate the entire set of random walks simultaneously. Assume that the first k vertices of all the random walks of length at least k are already generated. At any time it is enough to store the starting and the current vertices of the fingerprint path, as we will eventually drop all the nodes on the path except the starting and the ending nodes. Sort these pairs by the ending vertices. Then by simultaneously scanning through the edge set and this sorted set we can have access to the neighborhoods of the current ending vertices. Thus each partial fingerprint path can be extended by a next vertex chosen from the out-neigbors of the ending vertex uniformly at random. For each partial fingerprint path we also toss a biased coin to determine if it has reached its final length with probability c or has to advance to the next round with probability $1 - c$. This algorithm is formalized as Algorithm 2.2.1.

The number of I/O operations the external memory sorting takes is

$$D \log_M D$$

where D is the database size and M is the available main memory. Thus the expected I/O requirement of the sorting parts can be upper bounded by

$$\sum_{k=0}^{\infty}(1-c)^k NV \log_M((1-c)^k NV) = \frac{1}{c} NV \log_M(NV) - \Theta(NV)$$

using the fact that after k rounds the expected size of the Paths array is $(1-c)^k NV$. Recall that V and N denote the numbers of vertices and fingerprints, respectively.

We need a sort on the whole index database to avoid random-access writes to the Fingerprint arrays. Also, upon updating the PathEnd variables we do not write the unsorted Paths array to disk, but pass it directly to the next sorting stage. Thus the total I/O is at most $\frac{1}{c} NV \log_M NV$ plus the necessary edge-scans.

Unfortunately this algorithm apparently requires as many edge-scans as the length of the longest fingerprint path, which can be very large: $\Pr\{\text{the longest fingerprint is shorter, than } L\} = (1-(1-c)^L)^{N \cdot V}$. Thus instead of scanning the edges in the final stages of the algorithm, we will change strategy when the Paths array has become sufficiently small. Assume a partial fingerprint path has its current vertex at v. Then upon this condition the distribution of the end of this path is identical to the distribution of the end of any fingerprint of v. Thus to finish the partial fingerprint we can retrieve an already finished fingerprint of v. Although this decreases the number of available fingerprints for v, this results in only a very slight loss of precision.[3]

Another approach to this problem is to truncate the paths at a given length L and approximate the ending distribution with the static PageRank vector, as described in Section 2.2.3.

2.2.2 Distributed index computing

In the distributed computing model we will invert the previous approach, and instead of sorting the path ends to match the edge set we will partition the edge set of the graph in such a way that each participating computer can hold its part of the edges in main memory. So at any time if a partial fingerprint with current ending vertex v requires a random out-edge of v, it can ask the respective computer to generate one. This will require no disk access, only network transfer.

More precisely, each participating computer will have several queues holding (PathStart, PathEnd) pairs: one large input queue, and for each computer one small output queue preferably with the size of a network packet.

The computation starts with each computer filling their own input queue with N copies of the initial partial fingerprints (u, u), for each vertex u belonging to the respective computer in the vertex partition.

Then in the input queue processing loop a participating computer takes the next input pair, generates a random out-edge from PathEnd, decides whether the fingerprint ends there, and if it does not, then places the pair in the output queue determined by the next vertex just generated. If an output queue

[3]Furthermore, we can be prepared for this event: the distribution of these v vertices will be close to the static PageRank vector, thus we can start with generating somewhat more fingerprints for the vertices with high PR values.

Algorithm 2.2.2 Indexing (distributed computing method)

The algorithm of one participating computer. Each computer is responsible for a part of the vertex set, keeping the out-edges of those vertices in main memory. For a vertex v, part(v) is the index of the computer that has the out-edges of v. The queues hold pairs of vertices (u, v), interpreted as (PathStart, PathEnd).

```
for $u$ with part($u$) = current computer do
  for $i := 1$ to $N$ do
    insert pair $(u, u)$ into InQueue /*start $N$ fingerprint paths from node $u$: initially PathStart=PathEnd= $u$*/
while at least one queue is not empty do /*some of the fingerprints are still being calculated*/
  get an element $(u, v)$ from InQueue/*if empty, wait until an element arrives.*/
  $w$ := a random out-neighbor of $v$/*prolong the path; we have the out-edges of $v$ in memory*/
  if random()< $c$ then /*with probability $c$ this fingerprint path ends here*/
    add $w$ to the fingerprints of $u$
  else /*with probability $1 - c$ the path continues*/
    $o$ := part($w$) /*the index of the computer responsible for continuing the path*/
    insert pair $(u, w)$ into the InQueue of computer $o$
transmit the finished fingerprints to the proper computers for collecting and sorting.
```

reaches the size of a network packet's size, then it is flushed and transferred to the input queue of the destination computer. Notice that either we have to store the partition index for those v vertices that have edges pointing to in the current computer's graph, or part(v) has to be computable from v, for example by renumbering the vertices according to the partition. For sake of simplicity the output queue management is omitted from the pseudo-code shown as Algorithm 2.2.2.

The total size of all the input and output queues equals the size of the Paths array in the previous approach after the respective number of iterations. The expected network transfer can be upper bounded by $\sum_{n=0}^{\infty}(1-c)^n NV = \frac{1}{c}NV$, if every fingerprint path needs to change computer in each step.

In case of the webgraph we can significantly reduce the above amount of network transfer with a suitable partition of the vertices. The key idea is to keep *each domain on a single computer*, since the majority of the links are intra-domain links as reported in [77, 41].

We can further extend the above heuristical partition to balance the computational and network load among the participating computers in the network. One should use a partition of the pages such that the *amount of global*

PageRank is ditributed uniformly across the computers. The reason is that the expected value of the total InQueue hits of a computer is proportional to the total PageRank score of vertices belonging to that computer. Thus when using such a partition, the total switching capacity of the network is challenged, not the capacity of the individual network links.

2.2.3 Query processing

The basic query algorithm is as follows: to calculate PPV(u) we load the ending vertices of the fingerprints for u from the index database, calculate the empirical distribution over the vertices, multiply it with $1-c$, and add c weight to vertex u. This requires one database access (disk seek).

To reach a precision beyond the number of fingerprints saved in the database we can use the *recursive property* of PPV, which is also referred to as the decomposition theorem in [75]:

$$\mathrm{PPV}(u) = c\,\mathbb{1}_u + (1-c)\frac{1}{|O(u)|}\sum_{v\in O(u)} \mathrm{PPV}(v)$$

where $\mathbb{1}_u$ denotes the measure concentrated at vertex u (i.e., the unit vector of u), and $O(u)$ is the set of out-neighbors of u.

This gives us the following algorithm: upon a query u we load the fingerprints for u, the set of out-neighbors $O(u)$, and the fingerprints for the vertices of $O(u)$. From this set of fingerprints we use the above equation to approximate PPV(u) using a higher amount of samples, thus achieving higher precision. This is a tradeoff between query time (database accesses) and precision: with $|O(u)|$ database accesses we can approximate the vector from $|O(u)| \cdot N$ samples. We can iterate this recursion, if want to have even more samples. We mention that such query time iterations are analogous to the basic dynamic programming algorithm of [75]. The main difference is that in our case the iterations are used to increase the number of fingerprints rather than the maximal length of the paths taken into account as in [75].

The increased precision is essential in approximating the PPV of a page with large neighborhood, as from N samples at most N pages will have positive approximated PPR values. Fortunately, this set is likely to contain the pages with highest PPR scores. Using the samples of the neighboring vertices will give more adequate result, as it will be formally analyzed in the next section.

We could also use the expander property of the web graph: after not so many random steps the distribution of the current vertex will be close to the static PageRank vector. Instead of allowing very long fingerprint paths we could combine the PR vector with coefficient $(1-c)^{L+1}$ to the approximation and drop all fingerprints longer than L. This would also solve the problem of the approximated individual PPR vectors having many zeros (in those vertices that have no fingerprints ending there). The indexing algorithms would benefit from this truncation, too.

There is a further interesting consequence of the recursive property. If it is known in advance that we want to personalize over a fixed (maybe large) set of pages, we can introduce an artificial node into the graph with the respective set of neighbors to generate fingerprints for that combination.

2.3 How Many Fingerprints are Needed?

In this section we will discuss the convergence of our estimates, and analyze the required amount of fingerprints for proper precision.

It is clear by the law of large numbers that as the number of fingerprints $N \to \infty$, the estimate $\widehat{\mathrm{PPV}}(u)$ converges to the actual personalized PageRank vector $\mathrm{PPV}(u)$. To show that the rate of convergence is exponential, recall that each fingerprint of u ends at v with probability $\mathrm{PPV}(u,v)$, where $\mathrm{PPV}(u,v)$ denotes the v^{th} coordinate of $\mathrm{PPV}(u)$. Therefore $N \cdot \widehat{\mathrm{PPV}}(u,v)$, the number of fingerprints of u that ends at v, has binomial distribution with parameters N and $\mathrm{PPV}(u,v)$. Then Chernoff's inequality yields the following bound on the error of over-estimating $\mathrm{PPV}(u,v)$ and the same bound holds for under-estimation:

$$\begin{aligned} & \Pr\{\widehat{\mathrm{PPV}}(u,v) > (1+\delta)\,\mathrm{PPV}(u,v)\} \\ = \; & \Pr\{N\,\widehat{\mathrm{PPV}}(u,v) > N(1+\delta)\,\mathrm{PPV}(u,v)\} \\ \le \; & e^{-N\cdot\mathrm{PPV}(u,v)\cdot\delta^2/4}. \end{aligned}$$

Actually, for applications the exact values are not necessary. We only need that the ordering defined by the approximation match fairly closely the ordering defined by the personalized PageRank values. In this sense we have exponential convergence too:

Theorem 8. *For any vertices u, v, w consider $\mathrm{PPV}(u)$ and assume that:*

$$\mathrm{PPV}(u,v) > \mathrm{PPV}(u,w)$$

Then the probability of interchanging v and w in the approximate ranking tends to 0 exponentially in the number of fingerprints used.

Theorem 9. *For any $\epsilon, \delta > 0$ there exists an N_0 such that for any $N \ge N_0$ number of fingerprints, for any graph and any vertices u, v, w such that $\mathrm{PPV}(u,v) - \mathrm{PPV}(u,w) > \delta$, the inequality $\Pr\{\widehat{\mathrm{PPV}}(u,v) < \widehat{\mathrm{PPV}}(u,w)\} < \epsilon$ holds.*

Proof. We prove both theorems together. Consider a fingerprint of u and let Z be the following random variable: $Z = 1$, if the fingerprint ends in v, $Z = -1$ if the fingerprint ends in w, and $Z = 0$ otherwise. Then $\mathbb{E}\, Z = \mathrm{PPV}(u,v) - \mathrm{PPV}(u,w) > 0$. Estimating the PPV values from N fingerprints the event of interchanging v and w in the rankings is equivalent to taking N independent Z_i variables and having $\sum_{i=1}^{N} Z_i < 0$. This can be upper bounded using Bernstein's inequality and the fact that $\mathrm{Var}(Z) = \mathrm{PPV}(u,v) + \mathrm{PPV}(u,w) - (\mathrm{PPV}(u,v) - \mathrm{PPV}(u,w))^2 \leq \mathrm{PPV}(u,v) + \mathrm{PPV}(u,w)$:

$$\begin{aligned}
\Pr\{\tfrac{1}{N}\textstyle\sum_{i=1}^{N} Z_i < 0\} &\leq e^{-N\frac{(\mathbb{E}\,Z)^2}{2\,\mathrm{Var}(Z)+4/3\,\mathbb{E}\,Z}} \\
&\leq e^{-N\frac{(\mathrm{PPV}(u,v)-\mathrm{PPV}(u,w))^2}{10/3\,\mathrm{PPV}(u,v)+2/3\,\mathrm{PPV}(u,w)}} \\
&\leq e^{-0.3N(\mathrm{PPV}(u,v)-\mathrm{PPV}(u,w))^2}
\end{aligned}$$

From the above inequality both theorems follow. □

The first theorem shows that even a modest amount of fingerprints are enough to distinguish between the high, medium and low ranked pages according to the personalized PageRank scores. However, the order of the low ranked pages will usually not follow the PPR closely. This is not surprising, and actually a significant problem of PageRank itself, as [86] showed that PageRank is unstable around the low ranked pages, in the sense that with small perturbation of the graph a very low ranked page can jump in the ranking order somewhere to the middle.

The second statement has an important theoretical consequence. When we investigate the asymptotic growth of database size as a function of the graph size, the number of fingerprints remains constant for fixed ϵ and δ.

2.4 Lower Bounds for PPR Database Size

In this section we will prove several worst case lower bounds on the complexity of personalized PageRank problem. The lower bounds suggest that the exact computation and storage of all personalized PageRank vectors is infeasible for massive graphs. Notice that the theorems cannot be applied to one specific input such as the webgraph. The theorems show that for achieving full personalization the web-search community should either utilize some specific properties of the webgraph or relax the exact problem to an approximate one as in our scenario.

In particular, we will prove that the necessary index database size of a fully personalized PageRank algorithm computing exact scores must be at least $\Omega(V^2)$ bits in worst case, and if personalizing only for H nodes, the size of the database is at least $\Omega(H \cdot V)$. If we allow some small error probability and approximation, then the lower bound for full personalization is linear in V, which is achieved by our algorithm of Section 2.2.

More precisely we will consider *two-phase algorithms*: in the first phase the algorithm has access to the graph and has to compute an index database. In the second phase the algorithm gets a query of arbitrary vertices u, v (and w), and it has to answer based on the index database, i.e., the algorithm cannot access the graph during query-time. An $f(V)$ *worst case lower bound* on the database size holds, if for any two-phase algorithm there exists a graph on V vertices such that the algorithm builds a database of size $f(V)$ in the first phase.

In the above introduced two-phase model, we will consider the following types of queries:

(1) *Exact:* Calculate $\mathrm{PPV}(u,v)$, the v^{th} element of the personalized PageRank vector of u.

(2) *Approximate:* Estimate $\mathrm{PPV}(u,v)$ with a $\widehat{\mathrm{PPV}}(u,v)$ such, that for fixed $\epsilon, \delta > 0$
$$\Pr\{|\widehat{\mathrm{PPV}}(u,v) - \mathrm{PPV}(u,v)| < \delta\} \geq 1 - \epsilon$$

(3) *Positivity:* Decide whether $\mathrm{PPV}(u,v)$ is positive with error probability at most ϵ.

(4) *Comparison:* Decide in which order v and w are in the personalized rank of u with error probability at most ϵ.

(5) *ϵ–δ comparison:* For fixed $\epsilon, \delta > 0$ decide the comparison problem with error probability at most ϵ, if $|\mathrm{PPV}(u,v) - \mathrm{PPV}(u,w)| > \delta$ holds.

(6) *ϕ–ϵ–δ top query*: given the vertex u, with probability $1-\epsilon$ compute the set of vertices W which have personalized PPR values according to vertex u greater than ϕ. Precisely we require the following:
$$\begin{aligned} \forall w \in V &: \mathrm{PPV}(u,w) \geq \phi \Rightarrow w \in W \\ \forall w \in W &: \mathrm{PPV}(u,w) \geq \phi - \delta \end{aligned}$$

Our tool towards the lower bounds will be the asymmetric communication complexity game *bit-vector probing* [69]: there are two players A and B. Player A has an m-bit vector x, player B has a number $y \in \{1, 2, \ldots, m\}$, and their task is to compute the function $f(x,y) = x_y$, i.e., the output is the y^{th} bit of the input vector. To compute the proper output they have to communicate, and communication is restricted in the direction $A \rightarrow B$. The *one-way communication complexity* [84] of this function is the required bits of transfer in the worst case for the best protocol.

Theorem 10 ([69])**.** *Any protocol that outputs the correct answer to the bit-vector probing problem with probability at least $\frac{1+\gamma}{2}$ must transmit at least γm bits in worst case.*

Now we are ready to prove our lower bounds. In all our theorems we assume that personalization is calculated for H vertices, and there are V vertices in total. Notice that in the case of full personalization $H = V$ holds.

Theorem 11. *Any algorithm solving the positivity problem (3) must use an index database of size $\Omega((1-2\epsilon)HV)$ bits in worst case.*

Proof. Set $\frac{1+\gamma}{2} = 1 - \epsilon$. We give a communication protocol for the bit-vector probing problem. Given an input bit-vector x we will create a graph, that 'codes' the bits of this vector. Player A will create a PPV database on this graph, and transmit this database to B. Then Player B will use the positivity query algorithm for some vertices (depending on the requested number y) such that the answer to the positivity query will be the y^{th} bit of the input vector x. Thus if the algorithm solves the PPV indexing and positivity query with error probability ϵ, then this protocol solves the bit-vector probing problem with probability $\frac{1+\gamma}{2}$, so the transferred index database's size is at least γm.

For the $H \leq V/2$ case consider the following graph: let $u_1, \dots, u_H$ denote the vertices for whose the personalization is calculated. Add $v_1, v_2, \dots, v_n$ more vertices to the graph, where $n = V - H$. Let the input vector's size be $m = H \cdot n$. In our graph each vertex v_j has a loop, and for each $1 \leq i \leq H$ and $1 \leq j \leq n$ the edge (u_i, v_j) is in the graph iff bit $(i-1)n + j$ is set in the input vector.

For any number $1 \leq y \leq m$ let $y = (i-1)n + j$; the personalized PageRank value $\text{PPV}(u_i, v_j)$ is positive iff (u_i, v_j) edge was in the graph, thus iff bit y was set in the input vector. If $H \leq V/2$ the theorem follows since $n = V - H = \Omega(V)$ holds implying that $m = H \cdot n = \Omega(H \cdot V)$ bits are 'coded'.

Otherwise, if $H > V/2$ the same construction proves the statement with setting $H = V/2$. □

Corollary 12. *Any algorithm solving the exact* PPV *problem (1) must have an index database of size* $\Omega(H \cdot V)$ *bits in worst case.*

Theorem 13. *Any algorithm solving the approximation problem (2) needs an index database of* $\Omega(\frac{1-2\epsilon}{\delta} H)$ *bits on a graph with* $V = H + \Omega(\frac{1}{\delta})$ *vertices in worst case. If* $V = H + O(\frac{1}{\delta})$*, then the index database requires* $\Omega((1-2\epsilon)HV)$.

Proof. We will modify the construction of Theorem 11 for the approximation problem. We have to achieve that when a bit is set in the input graph, then the queried $\text{PPV}(u_i, v_j)$ value should be at least 2δ, so that the approximation will decide the positivity problem, too. If there are k edges incident to vertex u_i in the constructed graph, then each target vertex v_j has weight $\text{PPV}(u_i, v_j) = \frac{1-c}{k}$. For this to be over 2δ we can have at most $n = \frac{1-c}{2\delta}$ possible $v_1, \dots, v_n$ vertices. With $\frac{1+\gamma}{2} = 1 - \epsilon$ the first statement of the theorem follows.

For the second statement the original construction suffices. □

This radical drop in the storage complexity is not surprising, as our approximation algorithm achieves this bound (up to a logarithmic factor): for fixed ϵ, δ we can calculate the necessary number of fingerprints N, and then for each vertex in the personalization we store exactly N fingerprints, independently of the graph's size.

Using somewhat more extra nodes in the graph we can prove an even stronger lower bound:

Theorem 14. *Any algorithm solving the approximation problem (2) needs a database of* $\Omega(\frac{1}{\delta}\log\frac{1}{\epsilon}\cdot H)$ *bits in worst case, when the graph has at least* $H+\frac{1-c}{8\delta\epsilon}$ *nodes.*

Proof. We prove the theorem by reducing the bit vector probing problem to the ϵ–δ approximation. Given a vector x of $m=\Omega(\frac{1}{\delta}\cdot\log\frac{1}{\epsilon}\cdot H)$ bits, player A will construct a graph and compute a PPR database with the indexing phase of the ϵ–δ approximation algorithm. Then A transmits this database to B. Player B will perform a sequence of queries such that the required bit x_y will be computed with error probability $\frac{1}{4}$. The above outlined protocol solves the bit vector probing with error probability $\frac{1}{4}$. Thus the database size that is equal to the number of transmitted bits is $\Omega(m)=\Omega(\frac{1}{\delta}\cdot\log\frac{1}{\epsilon}\cdot H)$ in worst case by Theorem 10. It remains to show the details of the graph construction on A's side and the query algorithm on B's side.

Given a vector x of $m=\frac{1-c}{2\delta}\cdot\log\frac{1}{4\epsilon}\cdot H$ bits, A constructs the "bipartite" graph with vertex set $\{u_i : i=1,\dots,H\}\cup\{v_{j,k} : j=1,\dots,\frac{1-c}{2\delta}, k=1,\dots,\frac{1}{4\epsilon}\}$. For the edge set, x is partitioned into $\frac{1-c}{2\delta}\cdot H$ blocks, where each block $b_{i,j}$ contains $\log\frac{1}{4\epsilon}$ bits for $i=1,\dots,H$, $j=1,\dots,\frac{1-c}{2\delta}$. Notice that each $b_{i,j}$ can be regarded as a binary encoded number with $0\le b_{i,j}<\frac{1}{4\epsilon}$. To encode x into the graph, A adds an edge $(u_i,v_{j,k})$ iff $b_{i,j}=k$, and also attaches a self-loop to each $v_{j,k}$. Thus the $\frac{1-c}{2\delta}$ edges outgoing from u_i represent the blocks $b_{i,1},\dots,b_{i,(1-c)/2\delta}$.

After constructing the graph A computes an ϵ–δ approximation PPR database with personalization available on $u_1,\dots,u_H$, and sends the database to B, who computes the y^{th} bit x_y as follows. Since B knows which of the blocks contains x_y it is enough to compute $b_{i,j}$ for suitably chosen i,j. The key property of the graph construction is that $\mathrm{PPV}(u_i,v_{j,k})=\frac{1-c}{|O(u_i)|}=2\delta$ iff $b_{i,j}=k$ otherwise $\mathrm{PPV}(u_i,v_{j,k})=0$. Thus B computes $\widehat{\mathrm{PPV}}(u_i,v_{j,k})$ for $k=1,\dots,\frac{1}{4\epsilon}$ by the second phase of the ϵ–δ approximation algorithm. If all $\widehat{\mathrm{PPV}}(u_i,v_{j,k})$ are computed with $|\mathrm{PPV}(u_i,v_{j,k})-\widehat{\mathrm{PPV}}(u_i,v_{j,k})|\le\delta$, then $b_{i,j}$ containing x_y will be calculated correctly. By the union bound the probability of miscalculating any of $\widehat{\mathrm{PPV}}(u_i,v_{j,k})$ is at most $\frac{1}{4\epsilon}\cdot\epsilon=\frac{1}{4}$. □

We now move on to other query types.

Theorem 15. *Any algorithm solving the comparison problem (4) requires an index database of* $\Omega((1-2\epsilon)HV)$ *bits in worst case.*

Proof. We will modify the graph of Theorem 11 so that the existence of the specific edge can be queried using the comparison problem. To achieve this we will introduce a third set of vertices $w_1,\dots,w_n$ in the graph construction, such that w_j is the complement of v_j: A puts the edge (u_i,w_j) in the graph iff (u_i,v_j) was not an edge, which means bit $(i-1)n+j$ was not set in the input vector.

Then upon query for bit $y=(i-1)n+j$, consider $\mathrm{PPV}(u_i)$. In this vector exactly one of v_j,w_j will have positive weight (depending on the input bit x_y), thus the comparison query $\mathrm{PPV}(u_i, v_j) > \mathrm{PPV}(u_i, w_j)$ will yield the required output for the bit-vector probing problem. □

Corollary 16. *Any algorithm solving the ϵ–δ comparison problem (5) needs an index database of $\Omega(\frac{1-2\epsilon}{\delta}H)$ bits on a graph with $V = H + \Omega(\frac{1}{\delta})$ vertices in worst case. If $V = H + O(\frac{1}{\delta})$, then the index database needs $\Omega((1-2\epsilon)HV)$ bits in worst case.*

Proof. Modifying the proof of Theorem 15 according to the proof of Theorem 13 yields the necessary results. □

The strongest lower bound that we can present concerns the top-query problem (6) and is a logarithmic factor stronger than the previous bounds.

Theorem 17. *Any algorithm solving the top query problem (6) with parameters $\epsilon \geq 0$, $\delta > 0$ needs a database of $\Omega(\frac{1-2\epsilon}{\delta}H \log V)$ bits in worst case, when the graph has $V \geq H + \left(\frac{1-c}{2\delta}\right)^2$ nodes.*

Proof. We will proceed according to the proof of Theorem 14.

Let $\phi = 2\delta$ and $k = \lfloor \frac{1-c}{2\delta} \rfloor$ and the graph have nodes $\{u_i : i = 1, \ldots, H\} \cup \{v_j : j = 1, \ldots, n\}$ (with $n = V - H$). By the assumptions on the vertex count, $n = \Omega(V)$ and $\sqrt{n} \geq k$.

Let the size of the bit-vector probing problem's input be $m = H \cdot k \cdot \log n/2$. Assign each of the $k \cdot \log n/2$ sized blocks to a vertex u_i and fix a code which encodes these bits into k-sized subsets of the vertices $\{v_j\}$. This is possible, as the number of subsets is $\binom{n}{k} > (\frac{n}{k})^k \geq \sqrt{n}^k$. These mappings are known to both parties A and B. Note that due to the constraints on n, k, H and V we have $k \cdot \log n/2 = \Omega(\frac{1}{\delta} \log V)$.

Given an input bit-vector of A, for each vertex u_i take its block of bits and compute the corresponding subset of vertices $\{v_j\}$ according to the fixed code. Let u_i have an arc into these vertices. Let all vertices v_j have a self-loop. Now A runs the first phase of the PPR algorithm and transfers the resulting database to B.

Given a bit index y, player B determines its block, and issues a top query on the representative vertex, u_i. As each of the out-neighbors w of u_i has $\mathrm{PPV}(u_i, w) = \frac{1-c}{|O(u_i)|} = \frac{1-c}{k} \geq \phi$, and all other nodes w' have $\mathrm{PPV}(u_i, w') = 0$, the resulting set will be the set of out-neighbors of u_i, with probability $1 - \epsilon$. Applying the inverse of the subset encoding, we get the bits of the original input vector, thus the correct answer to the bit-vector probing problem. Setting $\frac{1+\gamma}{2} = 1 - \epsilon$ we get that the number of bits transmitted, thus the size of the database was at least $\Omega(\gamma H \cdot k \cdot \log n/2) = \Omega(\frac{1-2\epsilon}{\delta} H \log V)$. □

2.5 Experiments

In this section we present experiments that compare our approximate PPR scores to exact PPR scores computed by the personalized PageRank algorithm of Jeh and Widom [75]. Our evaluation is based on the web graph of 80 million pages crawled in 2001 by the Stanford WebBase Project [71]. We also validated the tendencies presented on a 31 million page web graph of the `.de` domain created using the Polybot crawler [103] in April 2004.

In the experiments we personalize on a single page u chosen uniformly at random from all vertices with non-zero outdegree. The experiments were carried out with 1000 independently chosen personalization node u, and the results were averaged.

To compare the exact and approximate PPR scores for a given personalization page u, we measure the difference between top score lists of exact $\mathrm{PPV}(u)$ and approximate $\widehat{\mathrm{PPV}}(u)$ vectors. The length k of the compared top lists is in the range 10 to 1000.

As our primary application area is query result ranking, we chose measures that compare the ordering returned by the approximate PPR method to the ordering specified by the exact PPR scores. In Section 2.5.1 we describe these measures that numerically evaluate the similarity of the top k lists. In Section 2.5.2 we present our experimental results.

2.5.1 Comparison of ranking algorithms

The problem of comparing the top k lists of different ranking algorithms has been extensively studied by the web-search community for measuring the speed of convergence in PageRank computations [78], the distortion of PageRank encodings [64] and the quality of rank-aggregation methods [45, 46, 43, 40].

In our scenario the exact PPR scores provide the ground truth ranking and the following three methods evaluate the similarity of approximate scores to the exact scores.

Let T_k^u denote the set of pages having the k highest personalized PageRank values in the vector $PPV(u)$ personalized to a single page u. We approximate this set by $\widehat{T_k^u}$, the set of pages having the k highest approximated scores in vector $\widehat{\mathrm{PPV}}(u)$ computed by our Monte Carlo algorithm.

The first two measures determine the overall quality of the approximated top-k set $\widehat{T_k^u}$, so they are insensitive to the ranking of the elements within $\widehat{T_k^u}$. *Relative aggregated goodness* [101] measures how well the approximate top-k set performs in finding a set of pages with high aggregated personalized PageRank. Thus relative aggregated goodness calculates the sum of exact PPR values in the approximate set compared to the maximum value achievable (by using the exact top-k set T_k^u):

$$\mathrm{RAG}(k,u) = \frac{\sum_{v \in \widehat{T_k^u}} \mathrm{PPV}(u,v)}{\sum_{v \in T_k^u} \mathrm{PPV}(u,v)}$$

We also measure the *precision* of returning the top-k set in the classical information retrieval terminology (note that as the sizes of the sets are fixed, precision coincides with *recall*):

$$\mathrm{Prec}(k,u) = \frac{|\widehat{T^u_k} \cap T^u_k|}{k}$$

The third measure, *Kendall's* τ compares the exact ranking with the approximate ranking in the top-k set. Note that the tail of approximate PPR ranking contains a large number of ties (nodes with equal approximated scores) that may have a significant effect on rank comparison. Versions of Kendall's τ with different tie breaking rules appear in the literature; we use the original definition as e.g. in [79] that we review next. Consider the pairs of vertices v, w. A pair is *concordant*, if both rankings strictly order this pair and agree on the ordering; *discordant*, if both rankings strictly order but disagree on the ordering of the pair; *e-tie*, if the exact ranking does not order the pair; *a-tie*, if the approximate ranking does not order the pair. Denote the number of these pairs by C, D, U_e and U_a respectively. The total number of possible pairs is $M = \frac{n(n-1)}{2}$, where $n = |T^u_k \cup \widehat{T^u_k}|$. Then *Kendall's* τ is defined as

$$\tau(k,u) = \frac{C - D}{\sqrt{(M - U_e)(M - U_a)}}$$

The range of Kendall's τ is $[-1, 1]$, thus we linearly rescaled it onto $[0, 1]$ to fit the other measures on the diagrams. To restrict the computation to the top k elements, the following procedure was used: we took the union of the exact and approximated top-k sets $T^u_k \cup \widehat{T^u_k}$. For the exact ordering, all nodes that were outside T^u_k were considered to be tied and ranked strictly smaller than any node in T^u_k. Similarly, for the approximate ordering, all nodes that were outside the approximate top-k set $\widehat{T^u_k}$ were considered to be tied and ranked strictly smaller than any node in $\widehat{T^u_k}$.

2.5.2 Results

We conducted experiments on a single AMD Opteron 2.0 Ghz machine with 4 GB of RAM under Linux OS. We used an elementary compression (much simpler and faster than [17]) to store the Stanford WebBase graph in 1.6 GB of main memory. The computation of 1000 approximated personalized PageRank vectors took 1.1 seconds (for $N = 1000$ fingerprints truncated at length $L = 12$). The exact PPR values were calculated using the algorithm by Jeh and Widom [75] with a precision of 10^{-8} in L_1 norm. The default parameters were number of fingerprints $N = 1000$ with one level of recursive evaluation (see Section 2.2.3) and maximal path length $L = 12$.

In our first experiments depicted on Figure 2.2, we demonstrate the exponential convergence of Theorems 8 and 9. We calculated Kendall's τ restricted to pairs that have a difference at least δ in their exact PPR scores. We displayed the effect of the number of fingerprints on this restricted τ for $\delta = 0.01$

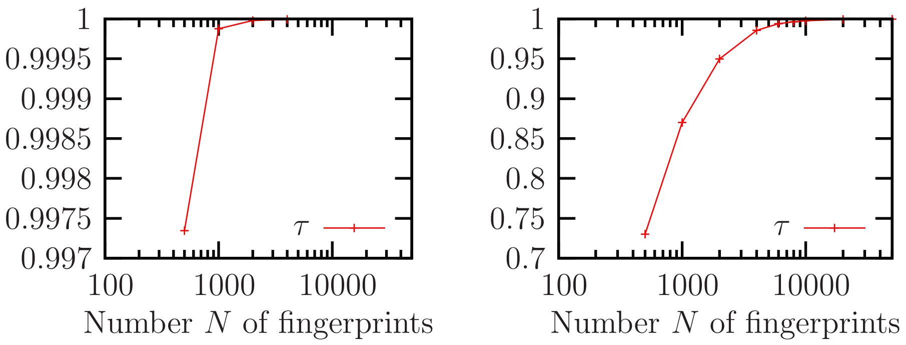


Figure 2.2: Effect of the number of fingerprints on Kendall's τ restricted to pairs with a PPR difference of at least $\delta = 0.01$ (left) and $\delta = 0.001$ (right).

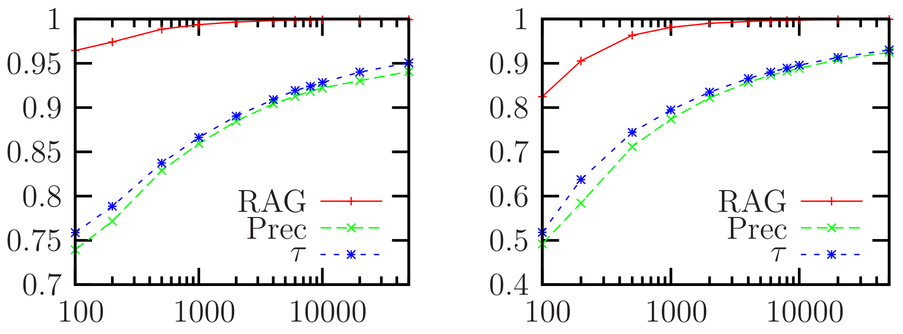


Figure 2.3: Effect of the number of fingerprints on various measures of goodness with (left) or without (right) recursive evaluation.

and $\delta = 0.001$. It can be clearly seen, that a modest amount of fingerprints suffices to properly order the pages with at least δ difference in their personalized PageRank values.

Figure 2.3 demonstrates the effects of the number of fingerprints and the recursive evaluation on the approximate ranking quality (without the previous restriction). The recursion was carried out for a single level of neighbors, which helped to reduce the number of fingerprints (thus the storage requirements) for the same ranking precision by an order of magnitude.

Figure 2.4 shows the effect of truncating the fingerprints at a maximum path length. It can be seen, that paths over length 12 have small influence on the approximation quality, thus the computation costs can be reduced by truncating them.

Finally, Figure 2.5 and Figure 2.6 (Figure 2.6 for N=10000 fingerprints) indicate that as the top list size k increases, the task of approximating the top-k set becomes more and more difficult. This is mainly due to the fact that among lower ranked pages there is a smaller personalized PageRank difference, which is harder to capture using approximation methods (especially Monte Carlo methods).

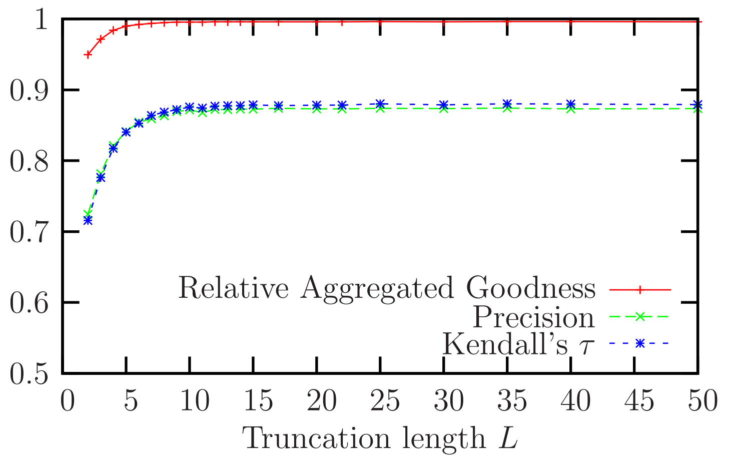


Figure 2.4: Effect of the path length/truncation on various measures of goodness.

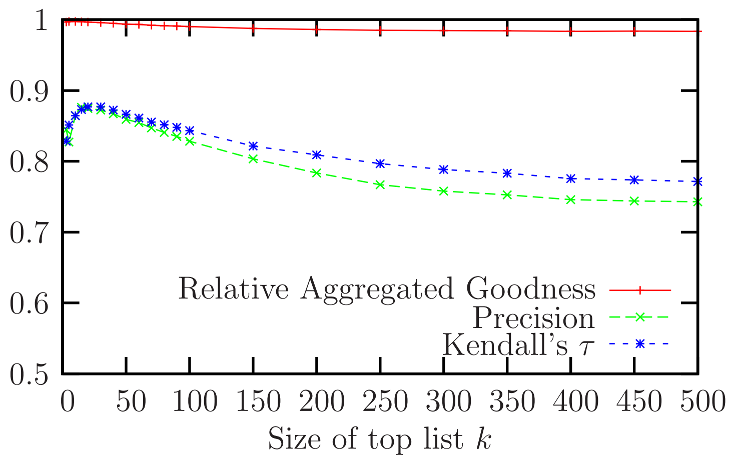


Figure 2.5: Effect of the size k of top set taken on various measures of goodness.

2.6 Conclusions and Open Problems

In this chapter we introduced a new algorithm for calculating personalized PageRank scores. Our method is a randomized approximation algorithm based on simulated random walks on the web graph. It can provide full personalization with a linear space index, such that the error probability converges to 0 exponentially with increasing the index size. The index database can be computed even on the scale of the entire web, thus making the algorithms feasible for commercial web search engines.

We justified this relaxation of the personalized PageRank problem to approximate versions by proving quadratic lower bounds for the full personalization problems. For the estimated PPR problem our algorithm is space-optimal up to a logarithmic factor.

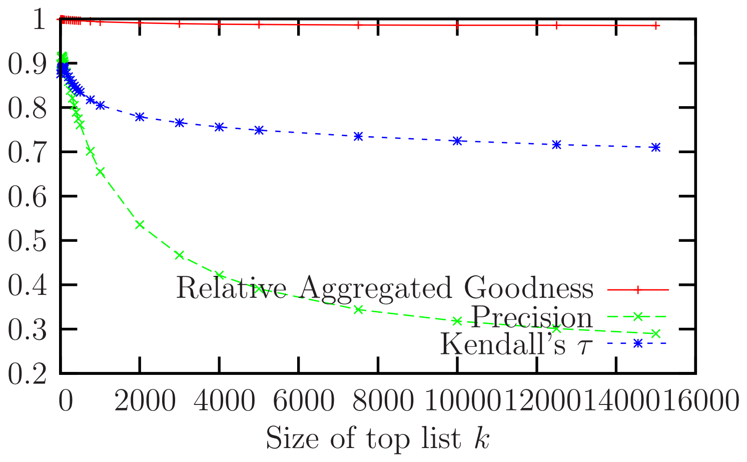


Figure 2.6: Effect of the size k of top set taken on various measures of goodness.

The experiments on 80M pages showed that using no more than $N = 1000$ fingerprints suffices for proper precision approximation.

An important future work is to combine and evaluate the available methods for computing personalized PageRank.

Bibliographical notes

The results in this chapter were first presented at the Third international Workshop for Algorithms and Models for the Web-Graph (WAW2004), held in conjunction with the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS2004) in Rome, Italy, in October 2004. The accompanying paper was published in LNCS vol. 3243 [87]. The core algorithm (Section 2.2 and Section 2.2.3), the external-memory indexing algorithm (Section 2.2.1) and the distributed indexing algorithm (Section 2.2.2) is the work of Balázs Rácz. The convergence speed theorem and the lower bounds (Sections 2.3 and 2.4) were obtained by Dániel Fogaras except the two strongest lower bound theorems 14 and 17 which are the work of Balázs Rácz and appreared first on the 15th Int'l World Wide Web Conference as [100]. The experiments were conducted by Károly Csalogány and Tamás Sarlós and appeared first in the Journal of Internet Mathematics as [54].

Chapter 3

Similarity Search

3.1 Introduction

The development of similarity search algorithms between web pages is motivated by the "related pages" queries of web search engines and web document classification. Both applications require efficient evaluation of an underlying similarity function, which extracts similarities from either the textual content of pages or the hyperlink structure. This chapter focuses on computing similarities solely from the hyperlink structure modeled by the *web graph*, with vertices corresponding to web pages and directed arcs to the hyperlinks between pages. In contrast to textual content, link structure is a more homogeneous and language independent source of information and it is in general more resistant against spamming. The authors believe that complex link-based similarity functions with scalable implementations can play such an important role in similarity search as PageRank [95] does for query result ranking.

Several link-based similarity functions have been suggested over the web graph. Functions introduced in social network analysis, like co-citation, bibliographic coupling, amsler and Jaccard coefficient of neighbors utilize only the *one-step neighborhoods* of pages. To exploit the information in *multi-step neighborhoods*, SimRank [74] and the Companion [38] algorithms were introduced by adapting the link-based ranking schemes PageRank [95] and HITS [82]. Further methods arise from graph theory such as similarity search based on network flows [90]. We refer to [89] containing an exhaustive list of link-based similarity search methods.

Unfortunately, no scalable algorithm has so far been published that allows the computation of multi-step similarity scores in case of a graph with billions of vertices. First, all the above algorithms require random access to the web graph, which does not fit into main memory with standard graph representations. In addition, SimRank iterations update and store a quadratic number of variables: [74] reports experiments on graphs with less than 300K vertices. Finally, related page queries submitted by users need to be served in less than a second, which has not yet been achieved by any published algorithm.

In this chapter we give the first scalable algorithms that can be used to evaluate multi-step link-based similarity functions over billions of pages on a distributed architecture. With a single machine, we conducted experiments on a test graph of 80M pages. Our primary focus is SimRank, which recursively refines the cocitation measure analogously to how PageRank refines in-degree ranking [95]. In addition we give an improved SimRank variant referred to as PSimRank, which refines the Jaccard coefficient of the in-neighbors of pages.

All our methods are Monte Carlo approximations: we precompute independent sets of *fingerprints* for the vertices, such that the similarities can be approximated from the fingerprints at query time. We only approximate the exact values; fortunately, the precision of approximation can be easily increased on a distributed architecture by precomputing independent sets of fingerprints and querying them in parallel.

Besides the algorithmic results we prove several worst case lower bounds on the database size of exact and approximate similarity search algorithms. The quadratic lower bound of the exact computation shows the non-existence of a general algorithm scalable on arbitrary graphs. The results suggest that scalability can only be achieved either by utilizing some specific property of the webgraph or by relaxing the exact computation with approximate methods as in our case.

We started to investigate the scalability of SimRank in [51], and we gave a Monte Carlo algorithm with the naive representation as outlined in the beginning of Section 3.2. The main contributions of this chapter are summarized as follows:

- In Section 3.2.1 we present a scalable algorithm to compute approximate SimRank scores by using a database of fingerprint trees, an efficient representation of precomputed random walks.
- In Section 3.2.2 we introduce and analyze PSimRank, a novel variant of SimRank with better theoretical properties and a scalable algorithm.
- In Section 3.3 we show that all the proposed Monte Carlo similarity search algorithms are especially suitable for distributed computing.
- In Section 3.4 we prove that our Monte Carlo similarity search algorithms approximate the similarity scores with a precision that tends to one exponentially with the number of fingerprints.
- In Section 3.5 we prove quadratic worst case lower bounds on the database size of exact similarity search algorithms and linear bounds in case of randomized approximation computation. The quadratic bounds show that exact algorithms are not scalable in general, while the linear bounds show that our algorithms are almost asymptotically worst case space-optimal.

- In Section 3.6 we report experiments about the quality and performance of the proposed methods evaluated on the Stanford WebBase graph of 80M vertices [71].

In the remainder of the introduction we discuss related results, define "scalability," and recall some basic facts about SimRank.

3.1.1 Related Results

Unfortunately the algorithmic details of "related pages" queries in commercial web search engines are not publicly available. We believe that an accurate similarity search algorithm should exploit both the hyperlink structure and the textual content. For example, the pure link-based algorithms like SimRank can be integrated with classical text-based information retrieval tools [6] by simply combining the similarity scores. A very promising text-based method is when the similarities are extracted from the anchor texts referring to pages as proposed by [27, 65].

Recent years have witnessed a growing interest in the scalability issue of link-analysis algorithms. Palmer et al. [96] formulated essentially the same scalability requirements that we will present in Section 3.1.2; they give a scalable algorithm to estimate the neighborhood functions of vertices. Analogous goals were achieved by the development of PageRank: Brin and Page [95] introduced PageRank algorithm using main memory of size proportional to the number of vertices. Then external memory extensions were published in [30, 62]. A large amount of research was done to attain scalability for personalized PageRank [66, 54]. The scalability of SimRank was also addressed by pruning [74], but this technique could only scale up to a graph with 300K vertices in the experiments of [74]. In addition, no theoretical argument was published about the error of approximating SimRank scores by pruning. In contrast, the algorithms of Section 3.2 were used to compute SimRank scores on a test graph of 80M vertices, and the theorems of Section 3.4 give bounds on the error of the approximation.

The key idea of achieving scalability by Monte Carlo (MC) algorithms was inspired by the seminal papers of Broder et al. [22] and Cohen [31] estimating the resemblance of text documents and size of transitive closure of graphs, respectively. Both papers utilize min-hashing, the fingerprinting technique for the Jaccard coefficient that was also applied in [65] to scale similarity search based on anchor text. Our framework of MC similarity search algorithms presented and analyzed in Section 3.4 is also related to the notion of locality-sensitive hashing (LSH) introduced in [73]. Notice the difference that LSH aggregates 0-1 similarities by testing the equality of hash values (or fingerprints), while our methods aggregate estimated scores from the range [0,1]. MC algorithms with simulated random walks also play an important role in a different aspect of web algorithms, when a crawler attempts to download a uniform sample of web pages and compute various statistics [70, 97, 11] or page decay [12]. A different approach to achieve scalability by forming clusters of objects and performing lookup only in the query-related cluster appears in [109].

Analogous results to the lower bounds of Section 3.5 are presented in [54] about personalized PageRank problem. Our theorems are proved by the techniques of Henzinger et al. [69] showing lower bounds for the space complexities of several graph algorithms with stream access to the edges. We refer to the PhD thesis of Bar-Yossef [9] as a comprehensive survey of this field.

3.1.2 Scalability Requirements

In our framework similarity search algorithms serve two types of queries: the output of a $\mathsf{sim}(u, v)$ *similarity query* is the similarity score of the given pages u and v; the output of a $\mathsf{related}_\alpha(u)$ *related query* is the set of pages for which the similarity score with the queried page u is larger than the threshold α. To serve queries efficiently we allow off-line precomputation, so the scalability requirements are formulated in the *indexing-query model*: we precompute an *index database* for a given web graph off-line, and later respond to queries on-line by accessing the database.

We say that a similarity search algorithm is *scalable* if the following properties hold:

- **Time:** The index database is precomputed within the time of a sorting operation, up to a constant factor. To serve a query the index database can only be accessed a constant number of times.
- **Memory:** The algorithms run in *external memory*: the available main memory is constant, so it can be arbitrarily smaller than the size of the web graph.
- **Parallelization:** Both precomputation and queries can be implemented to utilize the computing power and storage capacity of thousands of servers interconnected with a fast local network.

Observe that the time constraint implies that the index database cannot be too large. In fact our databases will be linear in the number V of vertices (pages). memory requirements do not allow random access to the web graph. We will first sort the edges by their ending vertices using external memory sorting. Later we will read the entire set of edges sequentially as a stream, and repeat this process a constant number of times.

3.1.3 Preliminaries about SimRank

SimRank was introduced by Jeh and Widom [74] to formalize the intuition that

"two pages are similar if they are referenced by similar pages."

The recursive *SimRank iteration* propagates similarity scores with a constant *decay factor* $c \in (0,1)$, with ℓ indexing the iteration:

$$\mathsf{sim}_{\ell+1}(u,v) = \frac{c}{|I(u)| \cdot |I(v)|} \sum_{u' \in I(u)} \sum_{v' \in I(v)} \mathsf{sim}_\ell(u',v')\,,$$

for vertices $u \neq v$, where $I(x)$ denotes the set of vertices linking to x; if $I(u)$ or $I(v)$ is empty, then $\mathsf{sim}_{\ell+1}(u,v) = 0$ by definition. For a vertex pair with $u = v$ we simply let $\mathsf{sim}_{\ell+1}(u,v) = 1$. The SimRank iteration starts with $\mathsf{sim}_0(u,v) = 1$ for $u = v$ and $\mathsf{sim}_0(u,v) = 0$ otherwise. The *SimRank score* is defined as the limit $\lim_{\ell\to\infty} \mathsf{sim}_\ell(u,v)$; see [74] for the proof of convergence. Throughout this chapter we refer to $\mathsf{sim}_\ell(u,v)$ as a SimRank score, and regard ℓ as a parameter.

The SimRank algorithm of [74] calculates the scores by iterating over all pairs of web pages, thus each iteration requires $\Theta(V^2)$ time and memory, where V denotes the number of pages. Thus the algorithm does not meet the scalability requirements by its quadratic running time and random access to the web graph.

We recall two generalizations of SimRank from [74], as we will exploit these results frequently. *SimRank framework* refers to the natural generalization that replaces the average function in SimRank iteration by an arbitrary function of the similarity scores of pairs of in-neighbors. Obviously, the convergence does not hold for all the algorithms in the framework, but still sim_ℓ is a well-defined similarity ranking. Several variants are introduced in [74] for different purposes.

For the second generalization of SimRank, suppose that a random walk starts from each vertex and follows the links backwards. Let $\tau_{u,v}$ denote the first meeting time random variable of the walks starting from u and v; $\tau_{u,v} = \infty$, if they never meet; and $\tau_{u,v} = 0$, if $u = v$. In addition, let f be an arbitrary function that maps the meeting times $0, 1, \ldots, \infty$ to similarity scores.

Definition 18. *The* expected f-meeting distance *for vertices u and v is defined as $\mathbb{E}(f(\tau_{u,v}))$.*

The above definition is adapted from [74] apart from the generalization that we do not assume uniform, independent walks of infinite length. In our case the walks may be pairwise independent, correlated, finite or infinite. For example, we will introduce PSimRank as an expected f-meeting distance of pairwise coupled random walks in Section 3.2.2.

The following theorem justifies the expected f-meeting distance as a generalization of SimRank, formulating it as the expected f-meeting distance with uniform independent walks and $f(t) = c^t$, where c denotes the decay factor of SimRank with $0 < c < 1$. The theorem was proved for infinite ℓ and totally independent set of walks in [74]; here we prove a stronger statement.

Theorem 19. *For uniform, pairwise independent set of reversed random walks of length ℓ, the equality $\mathbb{E}(c^{\tau_{u,v}}) = \mathsf{sim}_\ell(u,v)$ holds, whether ℓ is finite or not.*

Proof. For a fixed graph we proceed by induction on ℓ. Then, the $\ell = \infty$ case follows from $\lim_{\ell\to\infty} \mathsf{sim}_\ell(u,v) = \mathsf{sim}_\infty(u,v)$ and $\lim_{\ell\to\infty} \mathbb{E}\left(c^{\tau^\ell_{u,v}}\right) = \mathbb{E}\left(c^{\tau^\infty_{u,v}}\right)$.

The $\ell = 0$ case is trivial, and the only non-trivial part is the induction step, when $u \neq v$ and $I(u), I(v) \neq \emptyset$ hold. Let us denote by $\mathsf{step}_x(x')$ the event that the reversed walk starting from x proceeds to x'. By applying the pairwise independence and the linearity of expectations, we obtain:

$$\begin{aligned}
\mathbb{E}\left(c^{\tau^{\ell+1}_{u,v}}\right) &= \\
&= \sum_{u'\in I(u)} \sum_{v'\in I(v)} \Pr\left\{\mathsf{step}_u(u') \text{ and } \mathsf{step}_v(v')\right\} \\
&\qquad\qquad \cdot \mathbb{E}\left(c^{\tau^{\ell+1}_{u,v}} \mid \mathsf{step}_u(u') \text{ and } \mathsf{step}_v(v')\right) \\
&= \sum_{u'\in I(u)} \sum_{v'\in I(v)} \Pr\left\{\mathsf{step}_u(u')\right\} \cdot \Pr\left\{\mathsf{step}_v(v')\right\} \\
&\qquad\qquad \cdot c \cdot \mathbb{E}\left(c^{\tau^\ell_{u',v'}}\right) \\
&= \frac{c}{|I(u)| \cdot |I(v)|} \sum_{u'\in I(u)} \sum_{v'\in I(v)} \mathsf{sim}_\ell(u,v) \\
&= \mathsf{sim}_{\ell+1}(u,v).
\end{aligned}$$

□

3.2 Monte Carlo similarity search algorithms

In this section we give the first scalable algorithm to approximate SimRank scores. In addition, we introduce a new similarity function PSimRank accompanied by a scalable algorithm.

All the algorithms fit into the framework of *Monte Carlo similarity search algorithms* that will be introduced through the example of SimRank. Theorem 19 expressed SimRank as the expected value $\mathsf{sim}_\ell(u,v) = \mathbb{E}(c^{\tau_{u,v}})$ for vertices u, v. Our algorithms generate reversed random walks, calculate the first meeting time $\tau_{u,v}$ and estimate $\mathsf{sim}_\ell(u,v)$ by $c^{\tau_{u,v}}$. To improve the precision of approximation, the sampling process is repeated N times and the independent samples are averaged. The computation is shared between indexing and querying as shown in Algorithm 3.2.1, a naive implementation. During the precomputation phase we generate and store N independent reversed random walks of length ℓ for each vertex, and the first meeting time $\tau_{u,v}$ is calculated at query time by reading the random walks from the precomputed index database.

The main concept of Monte Carlo similarity search already arises in this example. In general *fingerprint* refers to a random object (a random walk in the example of SimRank) associated with a node in such a way, that the expected similarity of a pair of fingerprints is the similarity of their nodes. The Monte

Algorithm 3.2.1 Indexing (naive method) and similarity query

N=number of fingerprints, ℓ=path length, c=decay factor. Indexing: Uses random access to the graph.

```
1: for $i := 1$ to $N$ do
2:    for every vertex $j$ of the web graph do
3:       Fingerprint$[i][j][]$:=random reversed path of
         length $\ell$ starting from $j$.
```

Query sim(u,v):

```
1: sim:=0
2: for $i := 1$ to $N$ do
3:    Let $k$ be the smallest offset with
      Fingerprint$[i][u][k]$=Fingerprint$[i][v][k]$
4:    if such $k$ exists then
5:       sim:=sim+$c^k$
6: return sim/$N$
```

Carlo method precomputes and stores fingerprints in an index database and estimates similarity scores at query time by averaging. The main difficulties of this framework are as follows:

- During indexing (generating the fingerprints) we have to meet the scalability requirements of Section 3.1.2. For example, generating the random walks with the naive indexing algorithm requires random access to the web graph, thus we need to store all the links in main memory. To avoid this, we will first introduce algorithms utilizing $\Theta(V)$ main memory and then algorithms using memory of constant size, where V denotes the number of vertices. These computational requirements are referred to as *semi-external memory* [1] and *external memory* models, respectively. The parallelization techniques will be discussed in Section 3.3.

- To achieve a reasonably sized index database, we need a compact representation of the fingerprints. In the case of the previous example, the index database (including an inverted index for related queries) is of size $2 \cdot V \cdot N \cdot \ell$. In practical examples we have $V \approx 10^9$ vertices and $N = 100$ fingerprints of length $\ell = 10$, thus the database is in total 8000 gigabytes. We will show a compact representation that allows us to encode the fingerprints in $2 \cdot V \cdot N$ cells, resulting in an index database of size 800 gigabytes.

- We need efficient algorithms for evaluating queries. For queries the main idea is that the similarity matrix is sparse, for a page u there are relatively few other pages that have non-negligible similarity to u. Our algorithms will enumerate these pages in time proportional to their number.

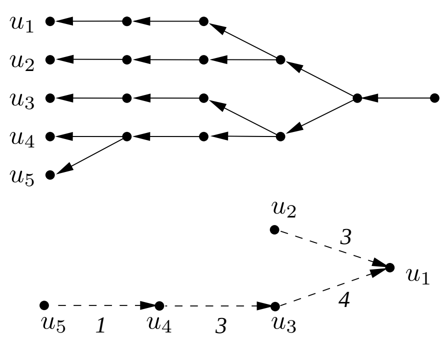


Figure 3.1: Representing the first meeting times of coalescing reversed walks of u_1, u_2, u_3, u_4 and u_5 (top) with a fingerprint graph (bottom). To decode the meeting time of u_2 and u_5: take the paths from u_2 and u_5 on the bottom. They meet first at u_1. The last edges before the meeting have labels 3 and 4. Thus $\tau_{u_2,u_5} = \max\{3,4\} = 4$.

3.2.1 SimRank

The main idea of this section is that we do not generate totally independent sets of reversed random walks as in Algorithm 3.2.1. Instead, we generate a set of *coalescing* walks: each pair of walks will follow the same path after their first meeting time. (This coupling is commonly used in the theory of random walks.) More precisely, we start a reversed walk from each vertex. In each time step, the walks at different vertices step independently to an in-neighbor chosen uniformly. If two walks are at the same vertex, they follow the same edge.

Notice that we can still estimate $\mathsf{sim}_\ell(u,v) = \mathbb{E}(c^{\tau_{u,v}})$ from the first meeting time $\tau_{u,v}$ of coalescing walks, since any pair of walks are independent until they first meet. We will show that the meeting times of coalescing walks can be represented in a surprisingly compact way by storing only one integer for each vertex instead of storing walks of length ℓ. In addition, coalescing walks can be generated more efficiently by the algorithm discussed in Section 3.2.1.3 than totally independent walks.

3.2.1.1 Fingerprint trees

A set of coalescing reversed random walks can be represented in a compact and efficient way. The main idea is that we do not need to reconstruct the actual paths as long as we can reconstruct the first meeting times for each pair of them. To encode this, we define the *fingerprint graph* (FPG) for a given set of coalescing random walks as follows.

The vertices of FPG correspond to the vertices of the web graph indexed by $1, 2, \ldots, V$. For each vertex u, we add a directed edge (u,v) to the FPG for at most one vertex v with

(1) $v < u$ and the fingerprints of u and v first meet at time $\tau_{u,v} < \infty$;

(2) among vertices satisfying (1) vertex v has earliest meeting time $\tau_{u,v}$;

(3) given (1-2), the index of v is minimal.

We label the edge (u, v) with $\tau_{u,v}$. An example for a fingerprint graph is shown as Fig. 3.1.

The most important property of the compact FPG representation that it still allows us to reconstruct $\tau_{u,v}$ values with the following formula. For a pair of nodes u and v consider the unique paths in the FPG starting from u and v. If these paths have no vertex in common, then $\tau_{u,v} = \infty$. Otherwise take the paths until the first common node w; let t_1 and t_2 denote the labels of the edges on the paths pointing to w; and let $t_1 = 0$ (or $t_2 = 0$), if $u = w$ (or $v = w$). Then $\tau_{u,v} = \max\{t_1, t_2\}$. (See the example of Fig. 3.1.) The correctness of this formula is stated in the lemma below.

Another important property appears in the lemma: each FPG is a collection of rooted trees, which will be referred to as *fingerprint trees*. The main observation for storage and query is that the partition of nodes into fingerprint trees preserves the locality of the similarity function.

Lemma 20. *Consider the fingerprint graph for a set of coalescing random walks. This graph is a directed acyclic graph, each node has out-degree at most 1, thus it is a forest of rooted trees with edges directed towards the roots.*

Consider the unique path in the fingerprint graph starting from vertex u. The indices of nodes it visits are strictly decreasing, and the labels on the edges are strictly increasing.

Any first meeting time $\tau_{u,v}$ can be determined by $\tau_{u,v} = \max\{t_1, t_2\}$ as detailed above.

Proof. The first two statements naturally follow from the definition of fingerprint graphs, so we focus on the last statement. Notice that $\tau_{u,v} < \infty$ iff $P(u)$ and $P(v)$ has a common vertex, where $P(x)$ denotes the unique path in the FPG starting from vertex x. This naturally follows from the transitivity of relation $\{\, (u, v) : \tau_{u,v} < \infty \,\}$. Thus, it remains to prove that $\tau_{u,v} = \max\{t_1, t_2\}$ holds for any vertices u, v with $\tau_{u,v} < \infty$.

Let us denote by w the first common vertex of paths $P(u)$ and $P(v)$. For $x = u, v$ let $|P(x, w)|$ be the number of edges in $P(x)$ from x to w; and if $|P(x, w)| > 0$, let x' denote the first vertex of $P(x)$ following x. We will refer to the labels of (u, u') and (v, v') as t'_1 and t'_2. Recall that t_1 and t_2 denote the labels of the edges of $P(u)$ and $P(v)$ with ending vertex w; furthermore $t_1 = 0$ (or $t_2 = 0$) if $|P(u, w)| = 0$ (or $|P(v, w)| = 0$). We refer to Fig. 3.2 summarizing the notation.

We will proceed induction on $k = |P(u, w)| + |P(v, w)|$ to prove that $\tau_{u,v} = \max\{t_1, t_2\}$ holds for any vertices u, v with $\tau_{u,v} < \infty$. The $k = 1$ case is trivial, and the induction step from k to $k + 1$ will be proved from the following fact referred to as the *generalized transitivity*:

$$\infty > \tau_{u,v} \geq \tau_{v,z} \implies \tau_{u,v} = \tau_{u,z}.$$

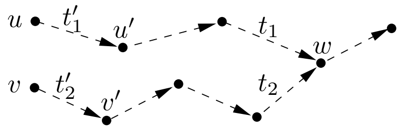


Figure 3.2: Notation of specific vertices and edge labels of a fingerprint graph. In the example $|P(u,w)| = 3$ and $|P(v,w)| = 4$.

We first discuss the case when one of u, v equals w, we may assume wlog that $u = w$ and $v \neq w$. By the definition of FPG $w = u < v$, so $\tau_{u,v} \geq t'_2 = \tau_{v,v'}$. From the generalized transitivity we get $\tau_{u,v} = \tau_{u,v'}$, which is equal to $\max\{t_1, t_2\} = t_2$ by induction.

In case of $u \neq w$ and $v \neq w$ assume (wlog) that $t'_2 \leq t'_1$. If $u < v$, then $\tau_{u,v} \geq t'_2 = \tau_{v,v'}$. If $u > v$, then $\tau_{u,v} \geq t'_1 \geq t'_2 = \tau_{v,v'}$. In both subcases we conclude that $\tau_{u,v} \geq \tau_{v,v'}$, so we get $\tau_{u,v} = \tau_{u,v'}$ by the generalized transitivity. By induction $\tau_{u,v} = \tau_{u,v'} = \max\{t_1, t_2\}$, if $v' \neq w$; otherwise $\tau_{u,v} = \tau_{u,v'} = \max\{t_1, 0\} = \max\{t_1, t_2\}$, the last equality following from $t_1 \geq t'_1 \geq t'_2 = t_2$. □

3.2.1.2 Fingerprint database and query processing

The first advantage of the fingerprint graph (FPG) is that it represents all first meeting times for a set of coalescing walks of length ℓ in compact manner. It is compact, since every vertex has at most one out-edge in an FPG, so the size of one graph is V, and $N \cdot V$ bounds the total size.[1] This is a significant improvement over the naive representation of the walks with a size of $N \cdot V \cdot \ell$.

The second important property of the FPG is that two vertices have non-zero estimated similarity iff they fall into the same fingerprint tree (same component of the FPG). Thus, when serving a $\mathsf{related}(u)$ query it is enough to read and traverse from each of the N fingerprint graphs the unique tree containing u. Therefore in a *fingerprint database*, we store the fingerprint graphs ordered as a collection of fingerprint trees, and for each vertex u we also store the identifiers of the N trees containing u. By adding the identifiers the total size of the database is no more than $2 \cdot N \cdot V$.

A $\mathsf{related}(u)$ query requires $N + 1$ accesses to the fingerprint database: one for the tree identifiers and then N more for the fingerprint trees of u. A $\mathsf{sim}(u, v)$ query accesses the fingerprint database at most $N + 2$ times, by loading two lists of identifiers and then the trees containing both u and v. For both type of queries the trees can be traversed in time linear in the size of the tree.

Notice that the query algorithms do not meet all the scalability requirements: although the number of database accesses is constant (at most N+2), the memory requirement for storing and traversing one fingerprint tree may

[1]To be more precise we need $V(\lceil\log(V)\rceil + \lceil\log(\ell)\rceil)$ bits for an FPG to store the labeled edges. Notice that the weights require no more than $\lceil\log(\ell)\rceil = 4$ bits for each vertex for typical value of $\ell = 10$.

be as large as the number of pages V. Thus, theoretically the algorithm may use as much as V memory.

Fortunately, in case of web data the algorithm performs as an external memory algorithm. As verified by our numerical experiments on 80M pages (see in Section 3.6.3) the average sizes of fingerprint trees are approximately 100–200 for reasonable path lengths. Even the largest trees in our database had at most 10K–20K vertices, thus 50Kbytes of data needs to be read for each database access in worst case.

3.2.1.3 Building the fingerprint database

It remains to present a scalable algorithm to generate coalescing sets of walks and compute the fingerprint graphs.

As opposed to the naive algorithm generating the fingerprints one-by-one, we generate all fingerprints together. With one iteration we extend all partially generated fingerprints by one edge. To achieve this, we generate one uniform in-edge e_j for each vertex j independently. Then extend with edge e_j each of those fingerprints that have the same last node j. This method generates a coalescing set of walks, since a pair of walks will be extended with the same edge after they first meet. Furthermore, they are independent until the first meeting time.

The pseudo-code is displayed as Algorithm 3.2.2, where NextIn[j] stores the starting vertex of the randomly chosen edge e_j, and PathEnd[j] is the ending vertex of the partial fingerprint that started from j. To be more precise, if a group of walks already met, then PathEnd[j]=“stopped” for every member j of the group except for the smallest j. The SaveNewFPGEdges subroutine detects if a group of walks meets in the current iteration, saves the fingerprint tree edges corresponding to the meetings and sets PathEnd[j]=“stopped” for all non-minimal members j of the group. This can be accomplished by a linear time counting sort of the non-stopped elements of PathEnd array.

The subroutine GenRndInEdges may generate a set of random in-edges with a simple external memory algorithm if the edges are sorted by the ending vertices. A significant improvement can be achieved by generating all the required random edge-sets together during a *single scan* over the edges of the graph. Thus, all the $N \cdot \ell$ edge-scans can be replaced by one edge-scan saving many sets of in-edges. Then GenRndInEdges sequentially reads the $N \cdot \ell$ arrays of size V from disk.

The algorithm outlined above fits into the semi-external memory model, since it utilizes $2 \cdot V$ main memory to store the PathEnd and NextIn arrays. (The counter sort operation of SaveNewFPGEdges may reuse NextIn array, so it does not require additional storage capacity.) The algorithm can be easily converted into the external memory model by keeping PathEnd and NextIn arrays on the disk and by replacing Lines 6-8 of Algorithm 3.2.2 with external sorting and merging processes. Furthermore, at the end of the indexing the individual fingerprint trees can be collected with ℓ sorting and merging operations, as the

Algorithm 3.2.2 Indexing (using $2 \cdot V$ main memory)

N=number of fingerprints, ℓ=length of paths. Uses subroutine GenRndInEdges that generates a random in-edge for each vertex in the graph and stores its source in an array.

```
1: for $i := 1$ to $N$ do
2:   for every vertex $j$ of the web graph do
3:     PathEnd[$j$] := $j$ /*start a path from $j$*/
4:   for $k$:=1 to $\ell$ do
5:     NextIn[] := GenRndInEdges();
6:     for every vertex $j$ with PathEnd[$j$]$\neq$"stopped" do
7:       PathEnd[$j$]:=NextIn[PathEnd[$j$]] /*extend the path*/
8:     SaveNewFPGEdges(PathEnd)
9:   Collect edges into trees and save as FPG$_i$.
```

longest possible path in each fingerprint tree is ℓ (due to Lemma 20 the labels are strictly increasing but cannot grow over ℓ).

3.2.2 PSimRank

In this section we give a new SimRank variant with properties extending those of Minimax SimRank [74], a non-scalable algorithm that cannot be formulated in our framework. The new similarity function will be expressed as an expected f-meeting distance by modifying the distribution of the set of random walks and by keeping $f(t) = c^t$.

A deficiency of SimRank can be best viewed by an example. Consider two very popular web portals. Many users link to both pages on their personal websites, but these pages are not reported to be similar by SimRank. An extreme case is when nodes u and v have the same in-neighborhood of size k, which have no in-edges. Though the k pages are totally dissimilar in the link-based sense, we would still intuitively regard u and v as similar. Unfortunately SimRank is counter-intuitive in this case, as $\mathsf{sim}_\ell(u, v) = c \cdot \frac{1}{k}$ tends to zero as the number k of common in-neighbors increases.

3.2.2.1 Coupled random walks

We solve the deficiency of SimRank by allowing the random walks to meet with higher probability when they are close to each other: a pair of random walks at vertices u', v' will advance to the same vertex (i.e., meet in one step) with probability of the Jaccard coefficient $\frac{|I(u')\cap I(v')|}{|I(u')\cup I(v')|}$ of their in-neighborhoods $I(u')$ and $I(v')$. Thus these walks will be *coupled* as opposed to the independent walks of SimRank.

Definition 21. *PSimRank is the expected f-meeting distance with $f(t) = c^t$ (for some $0 < c < 1$) of the following set of random walks. For each vertex u,* *the random walk X_u makes ℓ uniform independent steps on the transposed web graph starting from point u. For each pair of vertices u, v and time t, assume that the random walks are at position $X_u(t) = u'$ and $X_v(t) = v'$. Then*

- *with probability $\frac{|I(u')\cap I(v')|}{|I(u')\cup I(v')|}$ they both step to the same uniformly chosen vertex of $I(u')\cap I(v')$;*
- *with probability $\frac{|I(u')\setminus I(v')|}{|I(u')\cup I(v')|}$ the walk X_u steps to a uniform vertex in $I(u')\setminus I(v')$ and the walk X_v steps to an independently chosen uniform vertex in $I(v')$;*
- *with probability $\frac{|I(v')\setminus I(u')|}{|I(u')\cup I(v')|}$ the walk X_v steps to a uniform vertex in $I(v')\setminus I(u')$ and the walk X_u steps to an independently chosen uniform vertex in $I(u')$.*

We give a set of random walks satisfying the coupling of the definition. For each time $t \geq 0$ we choose an independent random permutation σ_t on the vertices of the web graph. At time t if the random walk from vertex u is at $X_u(t) = u'$, it will step to the in-neighbor with smallest index given by the permutation σ_t, i.e.,

$$X_u(t+1) = \operatorname*{argmin}_{u''\in I(u')} \sigma_t(u'')$$

It is easy to see that the random walk X_u takes uniform independent steps, since we have a new permutation for each step. The above coupling is also satisfied, since for any pair u', v' the vertex $\operatorname{argmin}_{w\in I(u')\cup I(v')} \sigma_t(w)$ falls into the sets $I(u')\cap I(v')$, $I(u')\setminus I(v')$, $I(v')\setminus I(u')$ with probabilities

$$\frac{|I(u')\cap I(v')|}{|I(u')\cup I(v')|}, \frac{|I(u')\setminus I(v')|}{|I(u')\cup I(v')|} \text{ and } \frac{|I(v')\setminus I(u')|}{|I(u')\cup I(v')|}, \text{ resp.}$$

PSimRank can be computed by the following linear equation system, placing PSimRank in the SimRank framework, which also proves that the values do not depend on the actual choice of the coupling:

$$\begin{aligned}
\mathsf{psim}_{\ell+1}(u,v) &= 1, \text{ if } u = v;\\
\mathsf{psim}_{\ell+1}(u,v) &= 0, \text{ if } I(u)=\emptyset \text{ or } I(v)=\emptyset;\\
\mathsf{psim}_{\ell+1}(u,v) &= c\cdot\Bigg[\tfrac{|I(u)\cap I(v)|}{|I(u)\cup I(v)|}\cdot 1 +\\
&+\tfrac{|I(u)\setminus I(v)|}{|I(u)\cup I(v)|}\cdot\tfrac{1}{|I(u)\setminus I(v)||I(v)|}\sum_{\substack{u'\in I(u)\setminus I(v)\\ v'\in I(v)}}\mathsf{psim}_\ell(u',v') +\\
&+\tfrac{|I(v)\setminus I(u)|}{|I(u)\cup I(v)|}\cdot\tfrac{1}{|I(v)\setminus I(u)||I(u)|}\sum_{\substack{v'\in I(v)\setminus I(u)\\ u'\in I(u)}}\mathsf{psim}_\ell(u',v')\Bigg].
\end{aligned}$$

3.2.2.2 Computing PSimRank

To achieve a scalable algorithm for PSimRank we modify the SimRank indexing and query algorithms introduced in Section 3.2.1. The following result allows us to use the compact representation of fingerprint graphs.

Lemma 22. *Any set of random walks satisfying the PSimRank requirements are coalescing, i.e., any pair follows the same path after their first meeting time.*

Proof. Let u and v be arbitrary nodes. By the first coupling requirement, if at time t the random walks X_u and X_v are at the same nodes $u' = v'$, then $I(u') = I(v')$, thus with probability $\frac{|I(u') \cap I(v')|}{|I(u') \cup I(v')}= 1$ they proceed to the same vertex. □

To apply the indexing algorithm of SimRank, we only need to ensure the pairwise coupling. This can be accomplished by replacing the GenRndInEdges procedure, which generated one independent, uniform in-edge for each vertex v in the graph for SimRank. In case of PSimRank, GenRndInEdges chooses a permutation σ at random; and then for each vertex v the in-neighbor with smallest index under the permutation σ is selected, i.e., vertex $\operatorname{argmin}_{v' \in I(v)} \sigma(v')$ is chosen.

As in the case of the GenRndInEdges for SimRank, all required sets of random in-edges can be generated within a single scan over the edges of the web graph, if the graph edges are sorted. The random permutations can be stored in small space by random linear transformations [23], thus the external memory implementation of SimRank can be extended to PSimRank.

3.2.3 Updating the index database

The web graph is not static, it changes over time quite rapidly. Web search engines are required to maintain a fairly up-to-date index database and are constantly refreshing the pages they have indexed. As a consequence, the web graph from which we compute our similarity scores is also changing: nodes and edges are added and removed constantly. In this section we describe possibilities to maintain an up-to-date index database in this chaning environment.

Brute force updating

In general purpose web search engines it is not common to have daily updates of the entire index database. Furthermore, there is a huge difference in workload between peak and off-peak periods, which means that considerable idle resources can be assigned for reindexing. The advantage of our method is that not all the N independent fingerprints are required to be recomputed at the same time, it can be distributed among multiple idle periods and multiple machines. In case of specialized news or blogosphere search engines the amount of data is considerably less, thus less resources are required for rebuilding the index.

Incorporating slight changes

Assume only a single node u has changed. If we don't use the compression given by the Fingerprint Graph representation, but build the index by the fingerprint paths, then the respective inverted index required for the queries has the list of the paths going through node u. These paths have to be examined for possible change. If the original degree was k and one link was added then with probability $\frac{k}{k+1}$ the database is not to be modified, with probability $\frac{1}{k+1}$ the paths have to be routed through the new link (where they coalesce with other walks). Assuming the changes are really slight, a disk-based implementation suffices.

In case of PSimRank we can incorporate slight changes into the FPG database: Given the changed node u and the saved seeds of the random permutations, follow the path from the changed point until the end. (This will be different for each of the ℓ possible positions where u might occur on a fingerprint path.) Denote the set of these ending vertices by U. As each fingerprint tree contains those nodes' paths that coalesced in ℓ steps, the ending vertex of the paths in a fingerprint tree is the same. Thus each fingerprint tree has a unique ending vertex. Select the fingerprint trees that have ending vertex in U (this might require an $N \cdot V$ sized index). Examine them for possible change. It can be seen that at most one edge has to be changed in the FPG, but that might attach a subtree to a different fingerprint tree. As there are at most $|U| \leq \ell$ trees to examine, this is feasible for incorporating slight changes.

Indexing based on the previous index

It is a common technique in PageRank index updates to use the solution of the previous database as a starting point for the new iteration. We can use this approach to gain a very interesting algorithm for index update that works on the FPG database for both SimRank and PSimRank. Performing a single iteration decays the current database and the expected solution's norm-distance by a factor of c. As we can have relatively small c (see Fig. 3.4), a large effect of incorporating the graph changes into the index can be achieved by a single iteration.

A single iteration of the linear system starting from the previous solution is represented in the fingerprint model by *prepending* a random edge to all fingerprints. This can be accomplished as follows: Run GenRndInEdges. Sort the result according to the generated in-neighbors' fingerprint trees. Then for each fingerprint tree and the new edges which we have to prepend to it, do the following: Decompress the fingerprint tree into coalescing reverse walks using unlabeled points as Fig. 3.1 displays. Then prepend the new edges to the reverse walks (attach them to the labeled points). Remove all paths that have no edges prepended[2], then relabel the nodes according to the new starting points. Remove the last step so that ℓ does not change due to the reindexing. The coalesced walks might fall apart into more sets of coalesced walks at this step. Finally each of the resulting sets of walks have to be compressed into a fingerprint tree.

The very interesting property of this update method is that the changes in the graph are introduced from the MSB (most significant 'digit') into the index. Assuming each night one such iteration is performed, then a new edge will be fully reflected only after $\ell = 10$ days, but the most important changes will be visible after the first night. Notice that if so desired, then more than one steps can be prepended to the fingerprint trees with one run of the above algorithm having more GenRndInEdges and sorts but a single iteration over the actual database (the fingerprint trees).

3.3 Monte Carlo parallelization

In this section we discuss the parallelization possibilities of our methods. We show that all of them exhibit features (such as fault tolerance, load balancing and dynamic adaptation to workload) which makes them extremely applicable in large-scale web search engines.

All similarity methods we have given in this chapter are organized around the same concepts:

- we compute a similarity measure by averaging N independent samples from a random variable;
- the independent samples are stored in N instances of an index database, each capable of producing a sample of the random variable for any pair of vertices.

The above framework allows a straightforward *parallelization* of both the indexing and the query: the computation of independent index databases can be performed on up to N different machines. Then the databases are transferred to the backend computers that serve the query requests. When a request arrives to the frontend server, it asks all (up to N) backend servers, averages their answers and returns the results to the user.

The Monte Carlo parallelization scheme has many advantages that make it perfectly suitable to large-scale web search engines:

Fault tolerance. If one or more backend servers cannot respond to the query in time, then the frontend can aggregate the results of the remaining ones and calculate the estimate from the available answers. This will not influence service availability, and results in a slight loss of precision.

[2]If no paths have edges prepended, then the entire tree has to be dropped.

Load balancing. In case of very high query loads, more than N backend servers (database servers) can be employed. A simple solution is to replicate the individual index databases. Better results are achieved if one calculates an independent index database for all the backend servers. In this case it suffices to ask *any* N backend servers for a proper precision answer. This allows seamless load balancing, i.e., you can add more backend servers one-by-one as the demand increases.

Dynamic adaptation to workload. During times of excessive load the number of backend servers asked for each query (N) can be automatically reduced to maintain fast response times and thus service integrity. Meanwhile, during idle periods, this value can be increased to get higher precision for free (along with better utilization of resources). We believe that this feature is extremely important in the applicability of our results.

3.4 Error of approximation

As we have seen in earlier sections, a crucial parameter of our methods is the number N of fingerprints. The index database size, indexing time, query time and database accesses are all linear in N. In this section we formally analyze the number of fingerprints needed for a given precision approximation. Our theorems show that even a modest number of fingerprints (e.g., $N = 100$) suffices for the purposes of a web search engine. A further theoretical consequence that N can be regarded as a constant, when the precision and error probability are fixed.

To state our results we need a suitably general model that can accommodate our methods for SimRank and PSimRank. Suppose that a randomized algorithm assigns N independent sets of fingerprints for the vertices and for any two vertices u, v a function maps the fingerprints to a score $S_i(u, v) \in [0, 1]$ for $i = 1, \dots, N$ such that $\mathbb{E}(S_i(u, v)) = \mathsf{sim}(u, v)$ for some similarity function $\mathsf{sim}(\cdot, \cdot)$. We will refer to $\widehat{\mathsf{sim}}(u, v) = \frac{1}{N}\sum_{i=1}^{N} S_i(u, v)$ as a *Monte Carlo similarity function*, which is a random function with $\mathbb{E}(\widehat{\mathsf{sim}}(u, v)) = \mathsf{sim}(u, v)$ by linearity. Notice that the approximate scores of our algorithms for SimRank and PSimRank can be regarded as Monte Carlo similarity functions $\widehat{\mathsf{sim}}(\cdot, \cdot)$.

Theorem 23. *For any Monte Carlo similarity function* $\widehat{\mathsf{sim}}$ *the absolute error converges to zero exponentially in the number of fingerprints* N *and uniformly over the pair of vertices* u, v*. More precisely, for any vertices* u, v *and any* $\delta > 0$ *we have*

$$\Pr\{|\widehat{\mathsf{sim}}(u, v) - \mathsf{sim}(u, v)| > \delta\} < 2e^{-\frac{6}{7}N\delta^2}$$

Proof. We shall use Bernstein's inequality in the following form. For any independent, identically distributed random variables $Z_i : i = 1, 2, \dots, N$ that have a bounded range $[a, b]$, for any $\delta > 0$:

$$\Pr\{|\frac{1}{N}\sum_{i=1}^{N} Z_i - \mathbb{E}\, Z| > \delta\} \leq 2e^{-N\frac{\delta^2}{2\operatorname{Var} Z + 2\delta(b-a)/3}}$$

Applying this for $Z_i = S_i(u, v)$ and using the bounds $Z_i \in [0, 1]$, $\operatorname{Var} Z_i \leq \frac{1}{4}$, and $\delta < 1$ the statement follows. □

Notice that the bound uniformly applies to all graphs and all Monte Carlo similarity functions, such as our approximations for SimRank and PSimRank. However, this bound concerns the convergence of the similarity score for one pair of vertices only. In the web search scenario, we typically use related queries, thus are interested in the relative order of pages according to their similarity to a given query page u. The above result implies that for a query page u the probability of interchanging two result pages v and w in the result lists tends to zero exponentially:

Corollary 24. *There exist constants β_1, β_2 such that for any Monte Carlo similarity function $\widehat{\mathsf{sim}}$ and pages u, v and w for which $\delta = \mathsf{sim}(u, v) - \mathsf{sim}(u, w) > 0$, the following holds:*

$$\Pr\{\widehat{\mathsf{sim}}(u, v) < \widehat{\mathsf{sim}}(u, w)\} < \beta_1 e^{-\beta_2 N\delta^2}$$

These theorems mean that the Monte Carlo approximation can efficiently capture the big differences among the similarity scores. But in case of small differences, the error of approximation obscures the actual similarity ranking, and an almost arbitrary reordering is possible. We believe, that for a web search inspired similarity ranking it is sufficient to distinguish between very similar, modestly similar, and dissimilar pages. We can formulate this requirement in terms of a slightly weakened version of classical information retrieval measures *precision* and *recall* [6].

Consider a related query for page u with similarity threshold α, i.e., the problem is to return the set of pages $S = \{v : \mathsf{sim}(u, v) > \alpha\}$. Our methods approximate this set with $\widehat{R} = \{v : \widehat{\mathsf{sim}}(u, v) > \alpha\}$. We weaken the notion of precision and recall to exclude a small, δ sized interval of similarity scores around the threshold α: let $R_{+\delta} = \{v : \mathsf{sim}(u, v) > \alpha + \delta\}$, $R_{-\delta} = \{v : \mathsf{sim}(u, v) > \alpha - \delta\}$. Then the *expected δ-recall* of a Monte Carlo similarity function is $\frac{\mathbb{E}(|\widehat{R} \cap R_{+\delta}|)}{|R_{+\delta}|}$ while the *expected δ-precision* is $\frac{\mathbb{E}(|\widehat{R} \cap R_{-\delta}|)}{\mathbb{E}(|\widehat{R}|)}$. We denote by $R^c_{-\delta}$ the complement set of $R_{-\delta}$.

Theorem 25. *For any Monte Carlo similarity function $\widehat{\mathsf{sim}}$, any page u, similarity threshold α and $\delta > 0$ the expected δ-recall is at least $1 - e^{-\frac{6}{7}N\delta^2}$ and the expected δ-precision is at least*

$$1 - \frac{|R^c_{-\delta}|}{|R_{+\delta}|}\frac{1}{e^{\frac{6}{7}N\delta^2} - 1} .$$

Proof. We only prove the bound for the expected δ-recall, the precision can be proved with analogous steps.

$$\begin{aligned}
&\mathbb{E}\left(|\widehat{R} \cap R_{+\delta}|\right) = \\
&= \mathbb{E}\Big(\sum_{v \in R_{+\delta}} \mathbb{1}\{v \in \widehat{R}\} \Big) = \sum_{v \in R_{+\delta}} \Pr\{v \in \widehat{R}\} \\
&\geq \sum_{v \in R_{+\delta}} \left(1 - e^{-\frac{6}{7}N\delta^2}\right) = |R_{+\delta}| \cdot \left(1 - e^{-\frac{6}{7}N\delta^2}\right),
\end{aligned}$$

where the second equation follows from the linearity of expectation; and the inequality follows from the one-sided absolute error bound $\Pr\{\widehat{\mathsf{sim}}(u,v) - \mathsf{sim}(u,v) < -\delta\} < e^{-\frac{6}{7}N\delta^2}$ that can be proved analogously to Theorem 23. □

This theorem shows that the expected δ-recall converges to 1 exponentially and uniformly over all possible similarity functions, graphs and queried vertices of the graphs, while the expected δ-precision converges to 1 exponentially for any fixed similarity function, graph and queried node.

3.5 Lower Bounds for the Similarity Database Size

In this section we will prove several lower bounds on the space complexity of calculating SimRank and PSimRank functions. In particular, we prove that except for the approximate approach, the required similarity database sizes are at least $\Omega(V^2)$ bits for some graphs with V vertices; which in turn means that exact computation is infeasible for large-scale computation. On the other hand, the lower bound for the approximate problem is linear in V, which is matched by our algorithm of Section 2.2 up to a logarithmic factor. Notice, that the worst case bounds cannot be applied to one particular input such as the webgraph. The main consequence of the theorems about similarity search is that the web-search community should either utilize some specific property of the webgraph or relax the exact problem to an approximate one as in our scenario.

More precisely we will consider *two-phase algorithms*: in the first phase the algorithm has access to the edge set of the graph and has to compute an index database; in the second phase the algorithm gets a query, and has to answer based on the index database, i.e., the algorithm cannot access the graph during query-time. A $b(V)$ *worst case lower bound* on the database size holds, if for any two-phase algorithm there exists a graph on V vertices such that the algorithm builds an index database of $b(V)$ bits.

In the two-phase model we will consider the below listed types of queries, where $\mathsf{sim}(\cdot,\cdot)$ denotes a similarity function. The input of the queries are vertices u, v (and w), the numbers ϵ and δ are fixed before the indexing phase.

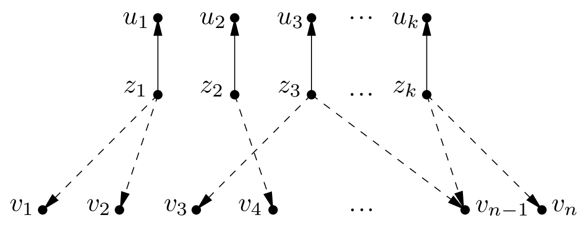


Figure 3.3: Encoding a vector x of $m = n \cdot k$ bits into a graph G_x. The existence of a dashed edge indicates that the corresponding bit x_y was set to $x_y = 1$.

(1) Exact: given the vertices u, v, calculate $\mathsf{sim}(u, v)$.

(2) Approximate: Estimate $\mathsf{sim}(u, v)$ with a $\widehat{\mathsf{sim}}(u, v)$ such that for fixed $\epsilon, \delta > 0$

$$\Pr\{|\widehat{\mathsf{sim}}(u, v) - \mathsf{sim}(u, v)| < \delta\} \geq 1 - \epsilon$$

(3) Positivity: Decide whether $\mathsf{sim}(u, v) > 0$ holds with error probability at most ϵ.

(4) Comparison: given the vertices u, v, w, decide whether $\mathsf{sim}(u, v) > \mathsf{sim}(u, w)$ holds with error probability at most ϵ.

(5) ϵ–δ comparison: given the vertices u, v, w with $|\mathsf{sim}(u, v) - \mathsf{sim}(u, w)| > \delta$, decide whether $\mathsf{sim}(u, v) > \mathsf{sim}(u, w)$ holds with error probability at most ϵ.

Our tool towards the lower bounds will be the asymmetric communication complexity game *bit-vector probing* [69]: there are two players A and B; player A has a vector x of m bits; player B has a number $y \in \{1, 2, \dots, m\}$; and they have to compute the function $f(x, y) = x_y$, i.e., the output is the y^{th} bit of the input vector x. To compute the proper output they have to communicate, and communication is restricted in the direction $A \to B$. The *one-way communication complexity* [84] of this function is the number of transferred bits in the worst case by the best protocol.

Theorem 26 ([69])**.** *Any protocol that outputs the correct answer to the bit-vector probing problem with probability at least* $\frac{1+\gamma}{2}$ *must transmit at least* γm *bits.*

In our theorems, we will substitute the function $\mathsf{sim}(\cdot, \cdot)$ by SimRank and PSimRank with path length $\ell = 1$ and we omit the decay factor by setting $c = 1^-$. So $\mathsf{sim}(u, v) = \frac{|I(u) \cap I(v)|}{|I(u)| \cdot |I(v)|}$ for SimRank and $\mathsf{sim}(u, v) = \frac{|I(u) \cap I(v)|}{|I(u) \cup I(v)|}$ for PSimRank, where $I(w)$ denotes the in-neighbors of w. We mention that all results can be easily extended for any constant c and ℓ. The following construction encodes the bits of a vector into the similarity scores of a graph.

Construction 27. *Suppose that x is a vector of m bits, where $m = k \cdot n$ for some $k \leq n$. Let G_x denote the graph with $2k+n$ vertices denoted by $u_1, \dots, u_k$, $z_1, \dots, z_k$, and $v_1, \dots, v_n$. For each $1 \leq i \leq k$ and $1 \leq j \leq n$ the edge (z_i, v_j) is in the graph iff bit $(j-1)k+i$ is set in the vector x; furthermore, we add an edge (z_i, u_i) for all $1 \leq i \leq k$. See Fig. 3.3 for the notation.*

It easily follows from the construction that $\mathsf{sim}(u_i, v_j) = \frac{1}{|I(v_j)|} \geq \frac{1}{k}$, if the bit $(j-1)k+i$ is 1 in the vector x, and $\mathsf{sim}(u_i, v_j) = 0$ otherwise, where $\mathsf{sim}(\cdot,\cdot)$ denotes either SimRank or PSimRank. Now we are ready to prove our lower bounds.

Theorem 28. *Any algorithm solving the positivity problem (3) of SimRank or PSimRank with probability at least $\frac{1+\gamma}{2}$ must use a database of size $\Omega(\gamma V^2)$ bits in worst case.*

Proof. The proof is the same for the three similarity functions. We give a communication protocol for the bit-vector probing problem as follows. Given an input x of $m = n^2$ bits, Player A creates a graph G_x with the above construction, where $k = n$. Then A computes a similarity index database from G_x and transmits the database to Player B. As B wants to know the bit x_y, he uses the positivity query algorithm for the vertices u_i, v_j, where $y = (j-1)k+i$. By Construction 27 the answer to the query is true iff $x_y = 1$ holds. Thus if the two-phase algorithm solves the positivity query with probability $\frac{1+\gamma}{2}$, then this protocol solves the bit-vector probing problem with probability $\frac{1+\gamma}{2}$, so the size of the transferred database is at least $\gamma m = \gamma n^2 = \gamma (V/3)^2$. □

Corollary 29. *Any algorithm solving the exact problem (1) for SimRank or PSimRank must have a similarity database of size $\Omega(V^2)$ bits in worst case.*

Theorem 30. *Any algorithm solving the approximation problem (2) for SimRank or PSimRank needs in worst case an index database of $\Omega(\frac{1-2\epsilon}{\delta} V)$ bits, if $\delta = \Omega(\frac{1}{V})$; and $\Omega((1-2\epsilon)V^2)$ otherwise.*

Proof. The proof is essentially the same as that of Theorem 28 with different parameters in the construction. Let $k = \min(\frac{1}{2\delta+1}, \frac{V}{3})$ and $n = V - 2k$. Player A encodes a vector x of $m = n \cdot k$ bits into a graph G_x by Construction 27 and transmits the index database to player B. Recall that either $\mathsf{sim}(u_i, v_j) \geq \frac{1}{k}$ or $\mathsf{sim}(u_i, v_j) = 0$ depending on the bit $y = (j-1)k+i$, so $\mathsf{sim}(u_i, v_j) > 2\delta$ iff $x_y = 1$. Then Player B decides on $x_y = 1$ iff $\widehat{\mathsf{sim}}(u_i, v_j) > \delta$ holds for the approximate score. The above outlined protocol solves the bit vector probing with probability $1 - \epsilon = \frac{1+\gamma}{2}$. By Theorem 26 the database size is at least $\gamma m = \gamma k n \geq (1-2\epsilon)k \cdot \frac{V}{3}$, which completes the proof. □

This radical drop in the storage complexity is not surprising, as our approximation algorithm achieves this bound up to a logarithmic factor: for a fixed ϵ, δ we can calculate the necessary number of fingerprints N by Theorem 23 (or by Theorem 24 for the ϵ–δ comparison problem), and then for each vertex in the graph we store exactly N fingerprints, independently of the size of the graph. This is a linear database, though the constant makes it very impractical. In the comparison problems (4) and (5) we have the same results.

Theorem 31. *Any algorithm solving the comparison problem (4) for SimRank or PSimRank with probability $\frac{1+\gamma}{2}$ requires a similarity database of $\Omega(\gamma V^2)$ bits in worst case.*

Proof. We will modify the proof of Theorem 28 by changing the graph construction. Player A encodes x into a graph G_x with $k = n$ by Construction 27. Then another set $w_1, \ldots, w_n$ is added to the vertices of G_x such that w_j is the complement of v_j: Player A puts an additional arc (z_i, w_j) in the graph iff (z_i, v_j) is not an arc, which means that bit $(i-1)n + j$ was not set in the input vector.

Then upon quering bit $y = (i-1)n+j$, exactly one of $\mathsf{sim}(u_i, v_j), \mathsf{sim}(u_i, w_j)$ will be positive (depending on the input bit x_y), thus the comparison query $\mathsf{sim}(u_i, v_j) > \mathsf{sim}(u_i, w_j)$ will yield the required output for the bit-vector probing problem. $\square$

Corollary 32. *Any algorithm solving the ϵ–δ comparison problem (5) for SimRank or PSimRank needs in worst case a similarity database of $\Omega(\frac{1-2\epsilon}{\delta}V)$ bits on graphs with $V = \Omega(\frac{1}{\delta})$ vertices, and $\Omega((1-2\epsilon)V^2)$ bits otherwise.*

Proof. Modifying the proof of Theorem 31 along the lines of the proof of Theorem 30 yields the necessary results. $\square$

3.6 Experiments

This section presents our experiments on the repository of 80M pages crawled by the Stanford WebBase project in 2001. The following problems are addressed by our experiments:

- How do the parameters ℓ, N and c effect the quality of the similarity search algorithms? The dependence on path length ℓ shows that multi-step neighborhoods contain more valuable similarity information than single-step neighborhoods. Quality increases in each step until $\ell = 5$.

- How do the qualities of SimRank and PSimRank relate to each other and to other link-based methods? We conclude that PSimRank outperforms all the other methods.

- What are the average and maximal sizes of fingerprint trees for SimRank and PSimRank? This is important, as the running time and memory requirement of query algorithms are proportional to these sizes. We measured sizes as small as $100 - 200$ on average implying fast running time with low memory requirement.

- What is the actual query serving performance of our methods? How does this performance scale as we add more servers and utilize the Monte Carlo parallelization possibilities?

3.6.1 Similarity Score Quality Measures

We briefly summarize the method of Haveliwala et al. [65] to measure the quality of similarity search algorithms; see [65] for a more detailed description. The most imortant property of this method is that it requires no expensive user survey. This allows a much greater number of pages to be included in the measurement (in our case 218,720 pages, as opposed to a typical user survey covering ≈1000 pages). Furthermore, the experiment is reproducible, and different parameter settings can be evaluated automatically.

The similarity search algorithms will be compared to a ground truth similarity ordering extracted from the Open Directory Project (ODP, [94]) data, a hierarchical collection of webpages managed by thousands of volunteer editors. The ODP category tree implicitly encodes similarity information: view it as an undirected graph and the consider the shortest-path-length function (siblings are more similar than cousins). Given a uniformly chosen query page u, this partial order will be compared to the order returned by the algorithm using the Kruskal-Goodman Γ measure:

$$\Gamma = 2 \cdot \Pr\{\widehat{\mathsf{sim}}(u,v) > \widehat{\mathsf{sim}}(u,w) | \widehat{\mathsf{sim}}(u,v) \neq \widehat{\mathsf{sim}}(u,w) \text{ and } \mathsf{sim}(u,v) > \mathsf{sim}(u,w)\} - 1$$

where $\widehat{\mathsf{sim}}(\cdot,\cdot)$ represents the similarity search algorithm to evaluate and $\mathsf{sim}(\cdot,\cdot)$ is the ODP similarity. As u, v, w are uniform random, the condition restricts computation to pages in the ODP (218,720 in our case) and to capture the top ranked portion of the result lists we further restrict v, w to be chosen from u's siblings and cousins in the ODP tree, denoted by sibling Γ [65].

3.6.2 Comparing the Quality under Various Parameter Settings

All the quality measurements were performed on a web graph of 78,636,371 pages crawled and parsed by the Stanford WebBase project in 2001. In our copy of the ODP tree 218,720 urls were found falling into 544 classes after collapsing the tree at level 3. The indexing process took 4 hours for SimRank and 14 hours for PSimRank with path length $\ell = 10$ and $N = 100$ fingerprints. We ran a semi-external memory implementation on a single machine with 2.8GHz Intel Pentium 4 processor, 2GB main memory and Linux OS. The total size of the computed database would have been 68GB. Since sibling Γ is based on similarity scores between vertices of the ODP pages, we only saved the fingerprints of the 218,720 ODP pages. A nice property of our methods is that this truncation (resulting in a size of 200Mbytes) does not affect the returned scores for the ODP pages.

We compared the qualities of (P)SimRank with that of cocitation measure and Jaccard coefficient measure extended to the ℓ-step neighborhoods of pages with exponentially decreasing weights. The latter measure will be referred to as *XJaccard* and it is defined as follows for given distance ℓ and decay factor $0 < c < 1$.

$$\mathsf{xjac}_\ell(u, v) = \sum_{k=1}^{\ell} \frac{|I_k(u) \cap I_k(v)|}{|I_k(u) \cup I_k(v)|} \cdot c^k(1 - c),$$

where $I_k(w)$ denotes the set of vertices from where w can be reached using at most k directed edges. XJaccard scores were evaluated by the min-hash fingerprinting technique of Broder [22] with an external memory algorithm that enabled us to collect fingerprints from the ℓ-step neighborhoods.

The results of the experiments are depicted on Fig. 3.4. Sibling Γ expresses the average quality of similarity search algorithms with Γ values falling into the range $[-1, 1]$. The extreme $\Gamma = 1$ result would show that similarity scores completely agree with the ground truth similarities, while $\Gamma = -1$ would show the opposite. Our $\Gamma = 0.3 - 0.4$ values imply that our algorithms agree with the ODP familial ordering in $65 - 70\%$ of the pairs.

The radically increasing Γ values for path length $\ell = 1, 2, 3, 4$ on the first diagram supports our basic assumption that the multi-step neighborhoods of pages contain valuable similarity information. The quality slightly increases for $\ell > 4$ in case of PSimRank and SimRank, while sibling Γ has maximum value for $\ell = 4$ in case of XJaccard. The quality of cocitation measure defined for the one-step neighborhoods is exceeded by all other measures for $\ell > 1$. Theoretically, cocitation could also be extended to the $\ell > 1$ case, but no scalable algorithm is known for evaluating it.

The second diagram shows the tendency that the quality of similarity search can be increased by smaller decay factor. This phenomenon suggests that we should give higher priority to the similarity information collected in smaller distances and rely on long-distance similarities only if necessary. The bottom diagram depicts the changes of Γ as a function of the number N of fingerprints. The diagram shows slight quality increase as the estimated similarity scores become more precise with larger values of N.

Finally, we conclude from all the three diagrams that PSimRank scores introduced in Section 3.2.2 outperform all other similarity functions. We also deduce from the experiments that path length ℓ has the largest impact on the quality of the similarity search compared to parameters N and c. Notice the difference between the scale of the first diagram and that of the other two diagrams.

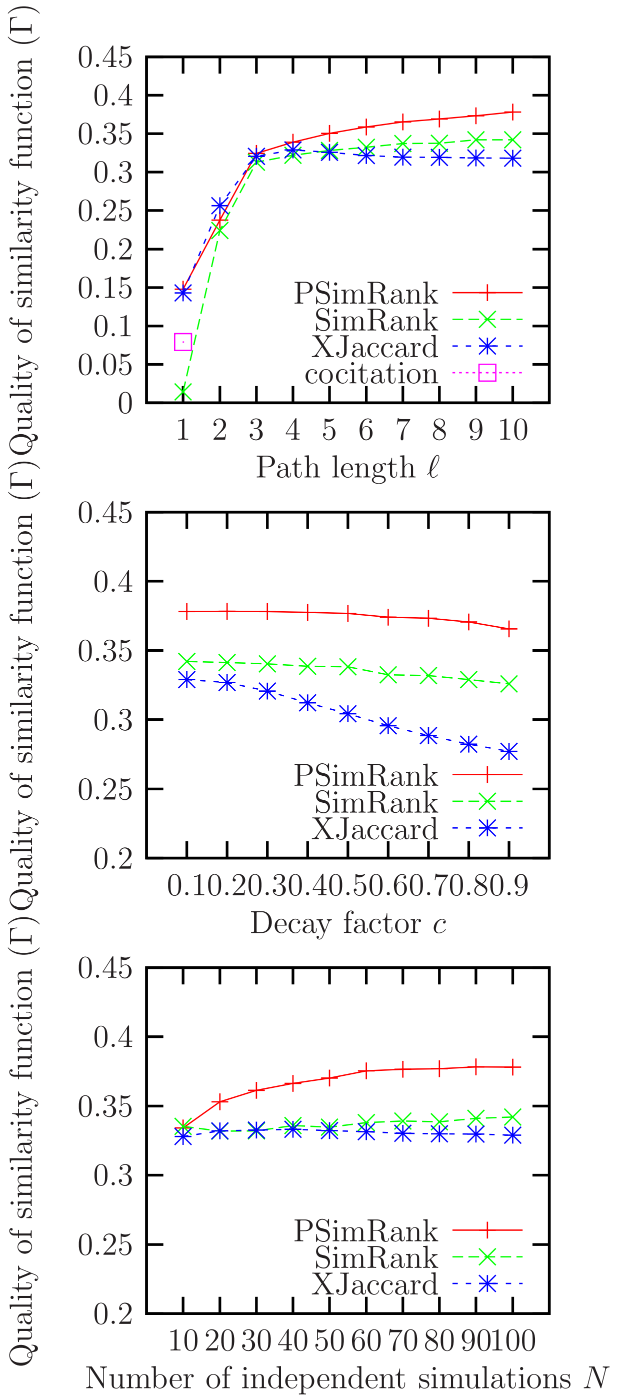


Figure 3.4: Varying algorithm parameters independently with default settings $\ell = 10$ for SimRank and PSimRank $\ell = 4$ for XJaccard, $c = 0.1$, and $N = 100$.

3.6.3 Time and memory requirement of fingerprint tree queries

Query evaluation for SimRank and PSimRank requires N fingerprint trees to be loaded and traversed (see Section 3.2.1.2). N can be easily increased with

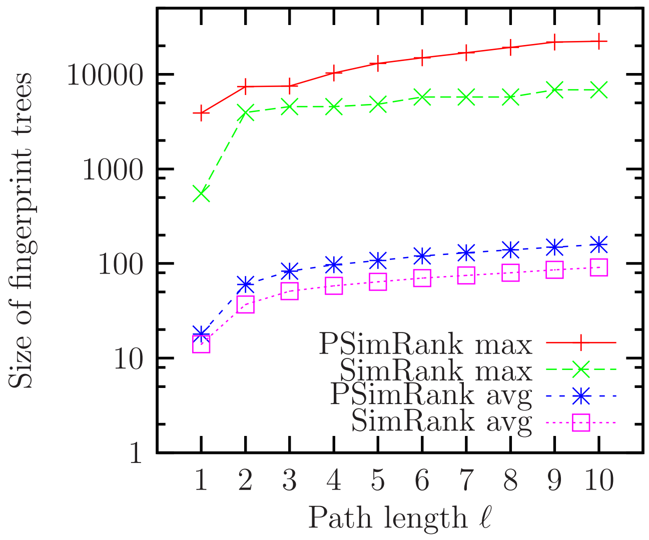


Figure 3.5: Fingerprint tree sizes for 80M pages with $N = 100$ samples.

Monte Carlo parallelization, but the sizes of fingerprint trees may be as large as the number V of vertices. This would require both memory and running time in the order of V, and thus violate the scalability requirements of Section 3.1.2. The experiments verify that this problem does not occur in case of real web data.

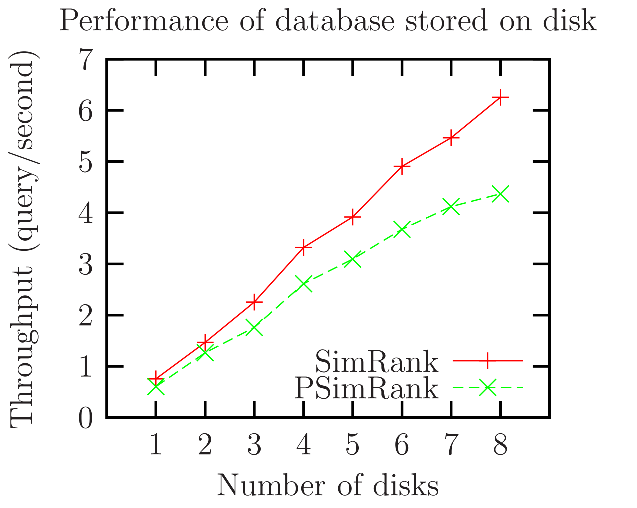
Fig. 3.5 shows the growing sizes of fingerprint trees as a function of path length ℓ in databases containing fingerprints for all vertices of the Stanford WebBase graph. The trees are growing when random walks meet and the corresponding trees are joined into one tree. It is not surprising that the tree sizes of PSimRank exceed that of SimRank, since the correlated random walks meet each other with higher probabilities than the independent walks of SimRank.

We conclude from the lower curves of Fig. 3.5 that the average tree sizes read for a query vertex are approximately 100–200, thus the algorithm performs like an external-memory algorithm on average in case of our web graph. Even the largest fingerprint trees have no more than 10–20K vertices, which is still very small compared to the 80M pages.

3.6.4 Run-time Performance and Monte Carlo Parallelization

In the third set of our experiments we show the actual related query serving performance of a sample implementation of our algorithms. In particular we are interested in whether our methods can be scaled to be a backend for an industrial strength web search engine. This focuses our experiments on the parallelization features.

We have to show the following two properties:



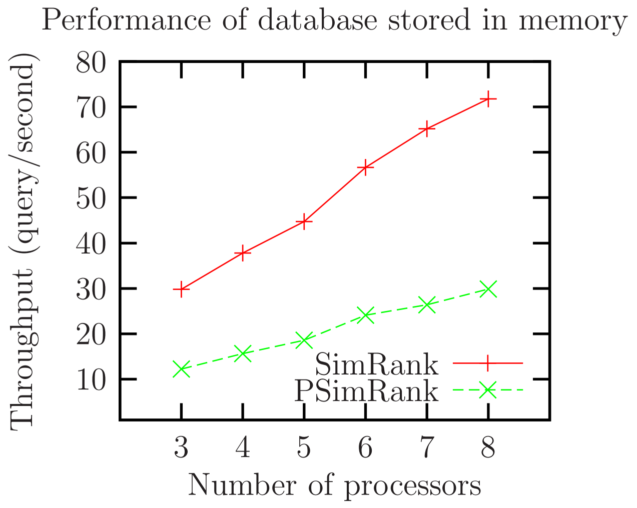


Figure 3.6: Actual query serving performance of a cluster of servers.

- The query serving response time is adequate for the requirement of on-line query (i.e. should be approx. 0.5 seconds).
- The query throughput of a server farm is reliably calculable and scales linearly (with a reasonable constant) in the computing resources available. This makes it possible to design and build a system for reliably serving an arbitrary large query workload.

Our experiments were conducted on dual Opteron 2.0 GHz machines having 4 GB of RAM each and using cheap 7200rpm IDE disks. The web graph we used for these experiments was taken from our national query engine indexing the `.hu` domain and contains 19,550,391 pages. The parameters used for the experiments were $N = 100$ and $\ell = 10$.

We examined two scenarios: In the first scenario the index database is stored on disks and for each of the N independent simulations a disk seek is required to load the fingerprint tree of the queried page. Then these trees are traversed and the result list is returned. Parallelization is obtained by distributing the N independent simulations to more disks/servers. The minimum required number of servers is one. In the second scenario the entire database is stored in main memory. This way there is no need to wait for the disk seeks. Parallelization is again obtained by distributing the independent simulations to more servers. The minimum required number of servers is determined so that their total available memory is at least the total database size (i.e. $N \cdot V$ cells); in our case the database was 10.5 GB, thus we needed at least 3 servers to run memory-based queries.

We measured query throughput by running a fixed sequential batch of 100 random queries in the different configurations and measuring the time required. The resulting throughput expressed in evaluated queries/second is depicted on Fig. 3.6. Note that the two requirements we stated above are clearly confirmed: the total throughput increases linearly in the number of computing nodes added, and the query serving response time is adequate from as low as 3 computing nodes. With our 4 dual-processor PCs we can serve as much as 6 million SimRank queries a day.

The performance difference between SimRank and PSimRank can be attributed to the size difference of the typical fingerprint trees. Note that in return we get considerably more and more precise results to a related query as using SimRank, thus the computing time is not wasted.

It is important to note when interpreting the scalability factor, that a single server in the farm can have more independent disks connected (in case of the disk based query serving) or more CPU(core)s available for calculations (in case of the memory based query serving). This makes it even more feasible to build a large capacity cluster for serving Monte Carlo similarity functions. In particular, connecting 8 (cheap) disks to a computing node gives approximately the same performance in case of PSimRank as the memory based method. This can get even more balanced as the actual query workload might utilize the disk cache to serve frequently queried pages faster[3].

We also ran performance comparison tests on existing and our methods. All results were normalized to show throughput per node, in units of query/second/ /node. Here a node means a processor (in case of memory-based methods) or disk (in case of disk-based methods). The contestants and results are summarized in Table 3.1.

3.7 Conclusion and open problems

We introduced the framework of link-based Monte Carlo similarity search to achieve scalable algorithms for similarity functions evaluated from the multi-step neighborhoods of web pages. Within this framework, we presented the

[3]In our experiments the disk cache was emptied between the test runs.

Method	Throughput (query/sec/nd)	Comment
XJaccard (disk)	0.688	fingerprints: $N = 50$, path length: $\ell = 4$
Co-citation (mem)	108.208	very low quality
Co-citation (disk)	3.700	very low quality
Text-based (disk)	0.193	inverted index of min-hash fingerprints stored in Berkeley DB
SimRank (disk)	0.779	should be multiplied by # of disks/node
SimRank (mem)	9.344	
PSimRank (disk)	0.606	should be multiplied by # of disks/node
PSimRank (mem)	3.872	

Table 3.1: Query performance comparison of similarity search methods.

first algorithm to approximate SimRank scores with a near linear external memory method and parallelization techniques sufficient for large scale computation. We also presented the new similarity functions extended Jaccard-coefficient and PSimRank. In addition, we showed that the index database used for serving queries can be efficiently updated for the changing webgraph. We proved asymptotic worst-case bounds on the required database size for exact and approximate computation of the similarity functions. These bounds suggest that exact computation is infeasible for large-scale computation in general, and our algorithms are near space-optimal for the approximate computation. We were the first to conduct experiments on large-scale web dataset for SimRank. Our results on the Stanford WebBase graph of 80M pages suggest that the novel PSimRank outperforms SimRank, cocitation and extended Jaccard coefficient in terms of quality. To demonstrate scalability, we measured that the query throughput of our algorithms increases linearly with the number of nodes in a cluster of servers. With 8 medium-sized servers, we were able to serve 70 queries per second on a collection of 19M pages.

Finally we phrase some interesting future directions of research:

- Monte-Carlo methods have an extended literature. The convergence speed of the straightforward application of the Monte-Carlo approximation could be improved by using more advanced methods. This could translate in direct improvements in quality or decreasing the resource requirements while maintaining the same quality level.

- Although we showed that the fingerprint tree-based query algorithm requires a constant number of database accesses, the time and memory requirements depend on the sizes of the individual trees. We conducted experiments to show that this is manageable on a large scale, but it would be nice to have a theoretical explanation on why. Assuming some graph construction model about the web graph (or some measured parameters is that is enough) computing the expected size (or the size distribution) of fingerprint trees would be interesting.

- It would be important to compare the performance (mainly quality) of link-based similarity search methods to that of text-based similarity search methods over the web. We expect that a suitable combination of text- and link-based methods would outperform both, but finding that suitable combination is an open problem.

Bibliographical notes

The first ideas of this chapter were presented at a workshop of the 9th International Conference on Extending Database Technology, in 2004 as [51]. The efficient representation of fingerprint trees, the new methods PSimRank and XJaccard as well as the thorough comparative experimental evaluation were presented at the 14th International World Wide Web Conference in 2005 as [52]. Further indexing methods and parallelization experiments were presented in the IEEE Transactions on Knowledge Discovery and Data Mining as [53].

The basic Monte-Carlo and the fingerprinted SimRank algorithm, PSimRank, XJaccard (parts of Section 3.2) and the quality experiments (parts of Section 3.6) are results of Dániel Fogaras. The compacted fingerprint graph/tree representation, update methods (Section 3.2.3), parallelization (Section 3.3), error bounds (Section 3.4), lower bounds (Section 3.5) and performance and parallelization experiments (Section 3.6.4) are the work of Balázs Rácz.

Chapter 4

The Common Neighborhood Problem

4.1 Introduction

We study the problem of finding pairs of vertices with large common neighborhoods in directed graphs. We consider the space complexity of the problem in the data stream model proposed by Henzinger, Raghavan, and Sajagopalan [67]. In this model of computation, the input arrives as a sequence of elements (for a graph, e.g., a sequence of arcs). Complexity is measured in terms of the number of times an algorithm can scan the input (in order) and the amount of space it requires to store intermediate results. Buchsbaum, Giancarlo, and Westbrook [24] claimed results for common neighborhood problems (defined below) in these models, but some of their lower-bound proofs are incorrect. We present improved results that rectify these issues.

The motivation for studying such problems in data stream models was established in the paper [24] as follows. Many large-scale systems generate massive sequences of data: records of telephone calls in a voice network [32, 72], transactions in a credit card network [28, 102], alarms signals from network monitors [88, 105], etc. From a practical standpoint, many applications require real-time decision making based on current information: e.g., fraud and intrusion detection [28, 32, 102] and fault recovery [88, 105]. Data must be analyzed as they arrive, not off-line after being stored in a central database. From a theoretical (as well as practical) standpoint, processing and integrating the massive amounts of data generated by a myriad of continuously operating sources poses many problems. For example, external memory algorithms [107] are motivated by the fact that many classical algorithms do not scale when data sets do not fit in main memory. At some point, however, data sets become so large as to preclude most computations that require more than one scan of the data, as they stream by, without the ability to recall arbitrary pieces of input previously encountered.

Common neighborhoods represent a natural, basic relationship between pairs of vertices in a graph. In transactional data like telephone calls and credit card purchases, common neighborhoods indicate users with shared interests (like whom they call or what they buy); inverted, they also represent market-basket information [47, 58, 106] (e.g., which products tend to be purchased together). In graphs representing relationships such as hyperlinks in the World Wide Web or citations by articles in a scientific database, common neighborhoods can yield clues about authoritative sources of information [81] or seminal items of general interest [67].

Informally, we show that any $O(1)$-pass, randomized (two-sided error) data stream algorithm that determines if *any* two vertices in a given directed graph have more than c common neighbors for a given c requires $\Omega(\sqrt{c}n^{3/2})$ bits of space. The definitions in Section 4.2 formalize the problems, and the results are formally presented in Theorems 38, 39, 41, and 42. In addition to using reductions from communication complexity, we also use results from extremal graph theory to prove our claims.

4.2 Preliminaries

4.2.1 Data Stream Models

In the *k-pass data stream model*, an algorithm $\mathcal{A}$ accesses a one-way, read-only *input tape*, a two-way, read-write *work tape*, and a one-way, write-only *output tape*. $\mathcal{A}$ is allowed an arbitrary amount of internal computation (albeit restricted to use the tapes for input, working memory, and output) as well as an arbitrary number of the standard operations on any of the tapes: read or write the symbol under the head, and move the head to the next position. $\mathcal{A}$ is also allowed $k-1$ *rewind* operations on the input tape, each of which resets the head to point to the first symbol on the tape. The space required is the length of the work tape in the worst-case over all possible inputs. A *randomized k-pass data stream algorithm* $\mathcal{A}$ can also access a one-way, read-only *random tape*, which contains an infinitely long string of random bits, and must report the correct answer with probability $1-\epsilon$ for a given ϵ. The space required is the length of the work tape in the worst-case over all possible inputs and random tapes.

These models were formalized by Henzinger, Raghavan, and Sajagopalan [67], who consider some neighborhood and connectivity problems in directed graphs. Variations now underly a substantial literature in streaming algorithms, aptly surveyed by Muthukrishnan [92].

4.2.2 Common Neighborhoods

Let $G=(V,A)$ be a directed graph. In what follows, $n=|V|$ and $m=|A|$. For each vertex a, define $N(a)=\{b:(a,b)\in A\}$; we call each $b\in N(a)$ a *neighbor* of a. Also define $deg(a)=|N(a)|$, the *out-degree* of a. Given two vertices a and

b, define $N(\{a, b\}) = N(a) \cap N(b)$; we call $N(\{a, b\})$ the *common neighborhood* of a and b.

Define $B(G) = \{\{u, v\} : |N(\{u, v\})| \geq T(\{u, v\})\}$, where $T(\{u, v\})$ is a given threshold function, which may depend on u and v. Given the threshold function, we wish to find $B(G)$. Since we are primarily interested in lower bounds, we concentrate on variations with uniform thresholds, in particular the following.

1. For all $u, v \in V$, $T(\{u, v\}) = c$, for some $c \in [1, n - 1]$.
2. For all $u, v \in V$, $T(\{u, v\}) = \alpha \min(|N(u)|, |N(v)|)$, where $0 < \alpha \leq 1$.

Also due to our focus on lower bounds, we consider only the corresponding decision problem of determining if $|B(G)| \geq \tau$, for some integer parameter τ. When $\tau = 1$, solving the decision problem determines whether there exists *any* pair of vertices whose common neighborhood size is greater than the given threshold function; we call this the *non-emptiness query* or the *non-emptiness problem.*

Define $f^{\epsilon}_{\text{CN}}(n, c)$ to be the space required for a randomized data stream algorithm to answer the non-emptiness query on an n-vertex graph, for $T(\{u, v\}) = c$, with probability $1 - \epsilon$ of being correct. We assume that the input graph G is given as an adjacency list; i.e., as a sequence of the form $\{(a_1, N(a_1)); (a_2, N(a_2)); \ldots; (a_n, N(a_n))\}$, for some arbitrary ordering of the vertices in V. Because any adjacency list corresponds to an edge list, but edge lists must be sorted to obtain adjacency lists, this assumption provides more powerful lower bounds than those assuming edge-list inputs.

4.2.3 Communication Complexity

We use reductions from two basic problems in communication complexity. Henzinger, Raghavan, and Rajagopalan [67] also used these problems to lower bound several data stream problems on graph algorithms. See Nisan and Kushilevitz [93] for a general introduction to communication complexity.

In the following problems there are two players, A and B; their goal is to compute a function value $f(x, y)$ with A having parameter x and B having parameter y. Thus they need to communicate. In a *one-round protocol*, A sends B a single message (sequence of bits), after which B must compute $f(x, y)$. In a *multi-round protocol*, A and B alternate the transmission of messages, and the protocol ends when one of them computes $f(x, y)$. The cost of a protocol is the total number of bits transmitted over all rounds in the worst case over all possible input pairs (x, y).

- In the *bit-vector index problem*, denoted IND, player A has an n-bit vector x, player B has an index $y \in \{1, 2, \ldots, n\}$, and the function to compute is $f(x, y) = x_y$; i.e., B must output the y^{th} bit of the input vector x.

- In the *bit-vector disjointness problem*, denoted DISJ, each player A and B has an n-bit vector, x and y respectively, under the assumption that there is at most one index i such that $x_i = y_i = 1$; $f(x, y) = 1$ if there exists such an index i, and $f(x, y) = 0$ otherwise.[1]

We are particularly interested in randomized protocols for these problems. We assume the *private-coin, two-sided model*, in which both A and B have access to private sources of random bits and for a given error parameter ϵ, the reported answer must be correct with probability $1-\epsilon$. The cost in this model is the worst-case number of bits transmitted in all rounds over all possible inputs and random choices of A and B.

Denote by $f^{\epsilon}_{\mathrm{IND}}(n)$ and $f^{\epsilon}_{\mathrm{DISJ}}(n)$ the respective complexities of the IND and DISJ functions for some arbitrary by fixed protocol in this model. Kremer, Nisan, and Ron [83, Theorem 3.7] show that for any one-round protocol and constant $\epsilon < 1/3$, $f^{\epsilon}_{\mathrm{IND}}(n) = \Omega(n)$. Bar-Yossef et al. [8, Theorem 6.6] show that for any multi-round protocol, $f^{\epsilon}_{\mathrm{DISJ}}(n) \geq \frac{n}{4}(1 - 2\sqrt{\epsilon})$.

4.2.4 Previous Results on Streaming Graph Problems

While plentiful results on streaming algorithms for many types of problems now grace the literature [92], such results for graph problems are still relatively few.

Henzinger, Raghavan, and Sajagopalan [67] give upper and lower bounds for solving various neighbor and path queries on graphs. Bar-Yossef, Kumar, and Sivakumar [7] give a general reduction tool for a class of "list-efficient" primitives and use it to devise a streaming algorithm for counting triangles in undirected graphs. Feigenbaum et al. [48, 49] give streaming results for various distance problems on undirected graphs, including the construction of spanners, which was also discussed by Elkin and Zhang [42]. Feigenbaum et al. also give streaming results for computing matchings [48] as well as $\Omega(n)$ results for a general class of "balanced" graph problems [49].

For computing non-emptiness in directed graphs, Buchsbaum, Giancarlo, and Westbrook [24] claimed results similar to ours, albeit in the deterministic model only. Their results for single-pass algorithms relied on [24, Lemma 2.1], the proof of which is incorrect. Specifically, they assume that a particular association from a graph to a memory image is onto, but this assumption is unjustified and potentially false. For $O(1)$-pass algorithms, they use a different proof technique to show an $\Omega(n)$-bit lower bound, which we strictly improve.

Our results show that finding common neighborhoods does not even fit into the "semi-streaming" model of Feigenbaum et al. [48], which allows $O(n \log^{O(1)} n)$ bits of space to process an n-vertex input graph.

[1]The assumption that x and y share at most one 1-bit is typical in the literature but not strictly necessary for our reduction.

4.3 Single-pass data stream algorithms

Lemma 33. *Assume that for a given s, m, and integer $c \in [2, s]$, there exists an undirected graph G_u with s vertices and m edges that does not contain undirected $K_{2,c}$ as a subgraph. Then there exists a family of 2^m directed graphs, each of which has $3s + c - 1$ vertices, for which any randomized, single-pass data stream algorithm $\mathcal{A}$ that with probability $1 - \epsilon$ correctly answers the non-emptiness query for $T(\{u, v\}) = c$ must use at least $f^{\epsilon}_{\mathrm{IND}}(m)$ bits of space.*

Proof. Form a directed graph G by arbitrarily directing the edges of G_u. Now augment G so that each original vertex has out-degree at least c and has a unique neighbor not shared by any other vertex, as follows.

Assign an arbitrary labeling $1, \ldots, m$ to the original arcs. For each original vertex u, add two new vertices, f_u and h_u, and add arc (u, f_u); this is the only arc to f_u. Each original vertex now has out-degree at least one. We call h_u the *shadow vertex* of u and will use it below. Finally add $c - 1$ new vertices $g_1, \ldots, g_{c-1}$, and for each original vertex u' with fewer than c neighbors, add arcs (u', g_i) for $1 \le i \le c - deg(u')$.

Note that only original vertices have positive out-degree, each original vertex now has out-degree at least c, and the augmented graph still has no $\overrightarrow{K_{2,c}}$. The latter claim follows because:

1. Only original vertices have non-zero out-degree.

2. Any original vertex u has f_u as a neighbor not shared by any other vertex and thus requires degree exceeding c to have c neighbors in common with another vertex.

3. Any vertex u with degree exceeding c has only f_u in addition to its original neighbors, no other vertex is adjacent to f_u, and no $\overrightarrow{K_{2,c}}$ existed before the augmentation.

Now we reduce IND to the non-emptiness problem. Players A and B both start with knowledge of the augmented graph.

Player A has an m-bit vector x. For each $1 \le i \le m$, if $x_i = 1$, he maintains the original i^{th} arc, say (a, b), in the augmented graph; otherwise, he changes this arc to (a, h_b). He then runs $\mathcal{A}$ on the modified graph and transmits the final memory image (contents of the work tape), which represents the entire state of $\mathcal{A}$, to Player B.

Player B has a query index $i \in [1, m]$, which he will use to create a new vertex v and an appropriate neighborhood. He will finish running algorithm $\mathcal{A}$, starting from the received memory image, by feeding $(v, N(v))$ to it, so that the answer to the non-emptiness query will be identical to bit i in A's input vector x.

Let (a, b) be the i^{th} original arc. First set the neighbors of v to be the following: b, f_a, and any $c - 2$ additional neighbors of a. Then add to the neighbor set of v the shadow vertex for each original vertex in $N(v)$ except that for b. Notice that:

1. Based on the random choices of player A, for each pair x, y of original vertices, at most one of (x, y) and (x, h_y) appears as an arc in the modified graph on which player A runs $\mathcal{A}$. Thus any $v' \neq v$ may have at most c common neighbors with v.

2. Because only a and v have arcs to f_a, only a may have c common neighbors with v.

3. If bit i was zero in the input x of player A, then b is not a common neighbor of a and v, and thus $B = \emptyset$; if bit i was one, then $B \supseteq \{\{a, v\}\}$.

Thus the output of $\mathcal{A}$ after B's additional input reports an answer to original instance of IND with the same correctness criteria. Because the only message transmitted from A to B was the memory image of $\mathcal{A}$ after the modified graph was input to it, the lemma follows. □

Remark 34. If the directed graph G in the preceding proof has only vertices of degree r and 0, then any graph on which $\mathcal{A}$ is run has only vertices of degree $r + 1$ and 0 (except v). Thus we can change the threshold function to be $T(\{u, v\}) = \alpha \min(|N(u)|, |N(v)|)$ so long as $c \leq \alpha(r+1)$, and the proof will still work if we ensure that v has at least $r+1$ neighbors: b, f_a, and $\lceil \alpha(r+1) \rceil - 2$ neighbors of the respective vertex a along with any shadow vertices of such, and possibly some new vertices. The number of extra vertices needed is linearly bounded, so the lower bound applies asymptotically to any algorithm solving the non-emptiness problem for $T(\{u, v\}) = \alpha \min(|N(u)|, |N(v)|)$.

For s and c denote by $ex(s, K_{2,c})$ the maximum number of edges a graph vith s vertices may have without containing a subgraph isomorphic to $K_{2,c}$. Lemma 33 implies that $f^{\epsilon}_{\mathrm{CN}}(n, c) = \Omega(ex(n, K_{2,c}))$. Bounding $ex(n, K_{2,c})$ in terms of n and c is a problem in extremal graph theory, related to an open problem posed by Zarankiewicz [110]. Füredi [55] has given an algebraic graph construction to gain exact asymptotics for this problem. The following theorem and proof closely match Füredi's result. We specify them to make the constructed graphs regular, which will benefit the sequel.

Theorem 35 (Füredi [55]). *For any prime power q and c such that $\frac{q-1}{c-1}$ is an integer, there exists a bipartite q-regular undirected graph on two classes of $\frac{q^2-1}{c-1}$ vertices that has no $K_{2,c}$ as a subgraph.*

Proof. For simplicity, let $t = c - 1$. Let $\mathbb{F}$ be the q-element field, and let $h \in \mathbb{F}$ be an element of order t. Let $H = \{h^\alpha \colon \alpha = 0, 1, \ldots, t-1\}$ be the set of powers of h.

Consider the set $\mathbb{F} \times \mathbb{F} \setminus \{(0, 0)\}$ and the following equivalence relation $\sim$: for any pairs (x, y) and (x', y'), let $(x, y) \sim (x', y')$ if and only if there exists an

α such that $x' = xh^\alpha$ and $y' = yh^\alpha$. Each equivalence class contains t pairs; thus the number of equivalence classes is $\frac{q^2-1}{t} = \frac{q^2-1}{c-1}$.

Consider a bipartite graph on vertex sets A and B, each vertex set corresponding to the set of equivalence classes of $\mathbb{F} \times \mathbb{F} \setminus \{(0,0)\}$. For each $(x,y) \in A$ and $(x',y') \in B$, connect them with an edge if and only if $xx' + yy' \in H$. This is clearly compatible with the relation $\sim$.

Let $(x,y) \in A$ be a vertex. Assume without loss of generality that $x \neq 0$. Then for any $y' \in F$ and $\alpha \in \{0,1,\dots,t-1\}$ there is a unique solution for x' to the equation $xx' + yy' = h^\alpha$. There are thus tq solutions (x',y') to $xx' + yy' \in H$, which form equivalence classes of size t each. Therefore (x,y) in A has q neighbors in B.

Consider a pair (x,y) and (x',y') of distinct vertices in A. For any common neighbor $(u,v) \in B$ of them, the following equations hold for some $\alpha, \beta \in \{0,1,\dots,t-1\}$:

$$\begin{aligned} xu + yv &= h^\alpha \\ x'u + y'v &= h^\beta. \end{aligned}$$

Considering this as a linear system of equations in variables u, v it follows that:

- If the determinant
$$\begin{vmatrix} x & y \\ x' & y' \end{vmatrix}$$
is non-zero, then for each α, β the equation has a unique solution for (u,v). There are t^2 choices for the pair α, β. As the solutions come in equivalence classes of size t, the vertices (x,y) and (x',y') have exactly t common neighbors.

- If the above determinant is zero, then there is an element $g \in F \setminus \{0\}$ such that $x' = xg$ and $y' = yg$. (Recall that neither (x,y) nor (x',y') can be $(0,0)$.) Then the system of equations has no solution unless $h^\beta = h^\alpha g$, from which $g \in H$, and thus $(x,y) \sim (x',y')$, which is a contradiction.

□

We formalize the end of the preceding proof for later use.

Lemma 36. *For the bipartite graph for parameters q and c constructed by Theorem 35, the following holds: The vertices of A can be partitioned into $q+1$ classes, each having $\frac{q-1}{c-1}$ elements, such that any two vertices in the same class have no common neighbors and any two vertices in different classes have exactly $c-1$ common neighbors.*

Proof. Recall the construction in the proof of Theorem 35. For any $(x,y) \in A$, consider the vertices $(x',y') = (xg, yg) \in A$ for $g \in \mathbb{F} \setminus \{0\}$. As g traverses the cosets of H, we get a class of $\frac{q-1}{t}$ vertices in A, no pair of which has a common neighbor. Thus we can classify the vertices of A into $\frac{q^2-1}{t} \cdot \frac{t}{q-1} = q+1$

classes, each having $\frac{q-1}{t}$ elements, such that any two vertices in the same class have disjoint neighborhoods, and any two vertices from different classes share exactly t common neighbors. □

The following upper bound will be essential in giving near-optimal algorithms for the common neighborhood problem.

Claim 37. *For directed graphs $ex(n, \overrightarrow{K_{2,c}}) \leq \sqrt{c}n^{3/2}$.*

Proof. Theorem 2.3 of Bollobás [18, p. 310] states that

$$ex(n, \overrightarrow{K_{s,t}}) \leq \frac{1}{2}(s-1)^{1/t}(n-t-1)n^{1-1/t} + \frac{1}{2}(t-1)n.$$

The claim follows by setting $s = c$ and $t = 2$. □

We are now ready to state the main theorems of this section.

Theorem 38. *For any constant $\epsilon < 1/3$, there exist infinitely many values of n and c such that for n-node graphs and c common neighbors $f^{\epsilon}_{\text{CN}}(n, c) = \Omega(\sqrt{c}n^{3/2})$ for one-pass data stream algorithms. There exists a deterministic, one-pass data stream algorithm that solves the non-emptiness problem with $T(\{u, v\}) = c$ using $O(\sqrt{c}n^{3/2})$ cells ($O(\sqrt{c}n^{3/2} \log n)$ bits) of space.*

Proof. By Theorem 35, for infinitely many values of s and c there exists a graph with s vertices and $\Omega(\sqrt{c}s^{3/2})$ arcs that does not contain $K_{2,c}$. The lower bound follows from Lemma 33 by setting $n = 3s + c - 1 \leq 4n$ and using the result from Kremer, Nisan, and Ron [83, Theorem 3.7] that for any one-round protocol and constant $\epsilon < 1/3$, $f^{\epsilon}_{\text{IND}}(k) = \Omega(k)$.

For the upper bound consider the following algorithm. Store the entire graph and process it off-line if the number of arcs does not exceed $\sqrt{c}n^{3/2}$. Otherwise by Claim 37 the proper answer to the non-emptiness query is "yes." □

To get a similar lower bound for the non-emptiness problem with $T(\{u, v\}) = \alpha \min(|N(u)|, |N(v)|)$ we use Remark 34 with the graphs of Theorem 35.

Theorem 39. *For any $\alpha \in (0, 1]$, there exist infinitely many values of n such that for n-node graphs $f^{\epsilon}_{\text{CN}}(n, \alpha \min(|N(u)|, |N(v)|)) = \Omega(\alpha n^2)$.*

Proof. For any fixed $0 < \alpha$, we can fix a $\alpha/2 < \beta < \alpha$ such that there are infinitely many prime powers q and c dividing $q - 1$ with $\frac{c}{q-1} \in [\beta, \alpha]$. Thus there are graphs for infinitely many n having $n = 2\frac{q^2-1}{c} = \Theta(q/\alpha)$ vertices and $\Theta(nq) = \Theta(\frac{q^2}{\alpha}) = \Theta(\alpha\frac{q^2}{\alpha^2}) = \Theta(\alpha n^2)$ arcs. The theorem follows from Remark 34. □

4.4 $O(1)$-pass data stream algorithms

In this section we will derive bounds similar to those of the previous section. As the algorithms are allowed a constant number of passes over the input, the players can exchange messages in both directions, so we can no longer use a small and isolated augmentation of the graph depending only on the input of one player. This makes the reduction more complex, and our main tool will be the DISJ problem, for which multi-round protocols have been bounded.

Lemma 40. *Assume that for some m and $d > 0$ there exists a bipartite graph $G_0(X, Y, E)$ with m edges such that:*

1. *X is partitioned into known* classes *$X_1, \ldots, X_k$;*

2. *no pair of vertices in any one class of X has more than one common neighbor;*

3. *no pair of vertices in two distinct classes of X has more than d common neighbors.*

Then for any $c > d$ there exists a family of directed graphs with $3|X|+|Y|+k(c-2)$ vertices each for which any randomized, $O(1)$-pass data stream algorithm $\mathcal{A}$ that with probability $1-\epsilon$ correctly answers the non-emptiness query for $T(\{u, v\}) = c$ must use $\Omega(f^{\epsilon}_{\mathrm{DISJ}}(m))$ bits of space.

Proof. We will give a reduction from DISJ, using a similar framework to the reduction used in the proof of Lemma 33. Using G_0 we will define a family of directed graphs G with three classes X', Y', and Z' of vertices. Neighborhoods of vertices in X' (rsp., Y') will depend on the input vector x of player A (rsp., y of player B); vertices in Z' will have out-degree zero. Then player A will run $\mathcal{A}$ on the adjacency lists of vertices in X', after which he will transmit the memory image of $\mathcal{A}$ to player B. Player B will then continue the run of $\mathcal{A}$ by inputting the adjacency lists of vertices in Y' and transmit the resulting memory image back to player A. A and B will iterate this procedure, always starting from the most current memory image received, until one declares the answer to the non-emptiness query. By construction, this answer will be identical to that for the instance of DISJ, and hence the lower bound for the latter will apply to the total number of bits transmitted in the protocol for $\mathcal{A}$. By assumption only $O(1)$ passes are used, so the bound applies asymptotically to at least one pass.

Assign an arbitrary labeling $1, \ldots, m$ to the edges of graph G_0. Let the vertices of graph G be the following:

- for each vertex $u \in X$ in G_0, three vertices $\mu_0(u), \mu_1(u), \mu_2(u)$;

- for each vertex $v \in Y$ in G_0, one vertex $\nu(v)$;

- for $1 \le i \le k$, $c-2$ vertices $\gamma^i_1, \ldots, \gamma^i_{c-2}$.

Define the arcs of G as follows.

- For each $u \in X$ in G_0, let i be such that $u \in X_i$, and create arcs $(\mu_1(u), \mu_0(u))$, $(\mu_2(u), \mu_0(u))$, and for $1 \leq j \leq c-2$, $(\mu_1(u), \gamma^i_j)$ and $(\mu_2(u), \gamma^i_j)$. Thus $\mu_0(u), \gamma^i_1, \gamma^i_2, \ldots, \gamma^i_{c-2}$ are common neighbors of $\mu_1(u)$ and $\mu_2(u)$ in G.

- For each i such that bit $x_i = 1$, create arc $(\mu_1(u), \nu(v))$ corresponding to the i^{th} edge (u, v) of G_0.

- For each i such that bit $y_i = 1$, create arc $(\mu_2(u), \nu(v))$ corresponding to the i^{th} edge (u, v) of G_0.

Define vertex classes $X' = \{\mu_1(u) : u \in X\}$, $Y' = \{\mu_2(u) : u \in X\}$, and $Z' = \{\mu_0(u) : u \in X\} \cup \{\nu(v) : v \in B\} \cup \{\gamma^i_j : 1 \leq i \leq k,\ 1 \leq j \leq c-2\}$. Then based upon G_0 and their respective input vectors, players A and B each know the vertex set of G; player A knows the neighbors of vertices in X'; and player B knows the neighbors of vertices in Y'. Vertices in Z' have out-degree zero.

For any $a \neq b \in X$, we claim $\mu_1(a)$ and $\mu_1(b)$ in G have at most $c-1$ common neighbors. If a and b are in the same class X_i, then $\mu_1(a)$ and $\mu_1(b)$ have $c-2$ common neighbors $\gamma^i_1, \ldots, \gamma^i_{c-2}$ plus a common neighbor $\nu(z)$ for any z that is a common neighbor of a and b in G_0; by assumption there is at most one such z. If, on the other hand, a and b are in distinct classes of X, then $\mu_1(a)$ and $\mu_2(b)$ only have common neighbors $\nu(z)$ for all z that are common neighbors of a and b in G_0; by assumption there are at most $d < c$ such z's. Similarly the pairs $\mu_1(a), \mu_2(b)$; $\mu_2(a), \mu_1(b)$; and $\mu_2(a), \mu_2(b)$ have at most $c-1$ common neighbors in G.

Thus the only pairs of vertices in G with at least c common neighbors are of the form $\mu_1(u), \mu_2(u)$ for some $u \in X$. Such a pair has c common neighbors if and only if there is some $v \in Y$ such that $(\mu_1(u), \nu(v))$ and $(\mu_2(u), \nu(v))$ are arcs in G, which occurs if and only if the i^{th} bit is 1 in both input vectors, where (u, v) is the i^{th} edge in G_0.

The answer to the non-emptiness query is therefore identical to that for the instance of DISJ. Note that it does not matter whether the specification of DISJ assumes that the input vectors have at most one common bit set. $\square$

We can use Füredi's constructions to bound the space requirement in terms of the number n of vertices for a fixed c.

Theorem 41. *For any fixed c, there exist infinitely many values of n such that for n-node graphs and c common neighbors $f^{\epsilon}_{\mathrm{CN}}(n, c) = \Omega(n^{3/2}(1 - 2\sqrt{\epsilon}))$ for $O(1)$-pass data stream algorithms.*

Proof. Theorem 35 posits the existence of $K_{2,2}$-free bipartite graphs on $s + s$ vertices with $\Theta(s^{3/2})$ edges for infinitely many s. To any such graph, apply Lemma 40 with just one $K_{2,2}$-free class of vertices, and set $n = 4s + c - 2$. Bar-Yossef et al.'s result [8, Theorem 6.6] that for any multi-round protocol, $f^{\epsilon}_{\mathrm{DISJ}}(k) \geq \frac{k}{4}(1 - 2\sqrt{\epsilon})$, completes the proof. $\square$

To gain asymptotics similar to Theorem 38 for the space needed in the $O(1)$-pass model, we exploit the more detailed view given by Lemma 36.

Theorem 42. *There exist infinitely many values of n and c with $c = O(n^{1/3})$ such that for n-node graphs $f^{\epsilon}_{\mathrm{CN}}(n, c) = \Omega(\sqrt{c}n^{3/2}(1-2\sqrt{\epsilon}))$ for $O(1)$-pass data stream algorithms. This is sharp up to a logarithmic factor; i.e., there exists an algorithm that solves the non-emptiness query with $O(\sqrt{c}n^{3/2}\log n)$ bits of space.*

Proof. For infinitely many values of q and c, Lemma 36 posits the existence of bipartite graphs $G_0(X, Y, E)$ such that $|X| = |Y| = \frac{q^2-1}{c-1}$; $|E| = q\frac{q^2-1}{c-1}$; and X can be partitioned into $q+1$ classes of $\frac{q-1}{c-1}$ vertices each, such that any two vertices in identical classes have disjoint neighborhoods, and any two vertices in different classes have $c-1$ common neighbors. To any such graph, apply Lemma 40 with $d = c-1$—the partition of X is given by Lemma 36—and set $n = 4\frac{q^2-1}{c-1} + (q+1)(c-2)$. To keep $n = \Theta(\frac{q^2}{c})$, we must further bound $qc = O(\frac{q^2}{c})$, yielding the requirement $c = O(\sqrt{q})$. Thus $\frac{q^2}{c} = \Omega(q^{3/2})$, so it suffices to assume $c = O(n^{1/3})$.

Now, $|E| = q\frac{q^2-1}{c-1} = \Theta(\frac{q^3}{c}) = \Theta(\sqrt{c}\frac{q^3}{c^{3/2}}) = \Theta(\sqrt{c}n^{3/2})$. Again, Bar-Yossef et al.'s result [8, Theorem 6.6] completes the proof of the lower bound. The upper bound was presented in the proof of Theorem 38. □

4.5 Conclusion and open problems

We have provided lower bounds on the space needed for $O(1)$-pass, randomized data stream algorithms to determine if a given directed graph has a pair of vertices with a common neighborhood of a given size. An open problem is to remove the restriction "$c = O(n^{1/3})$" from the result of Theorem 42, or provide an appropriate algorithm if the bound is sharp.

Bibliographical notes

This chapter focuses on the problems studied in [24]. In that paper the proofs given by the original authors were incorrect. This chapter provides correct proofs and generalizations of the theorems that are the work of Balázs Rácz, and were first published in the journal Theoretical Computer Science as [25].

Chapter 5

References

[1] James Abello, Adam L. Buchsbaum, and Jeffery Westbrook. A functional approach to external graph algorithms. In *European Symposium on Algorithms*, pages 332–343, 1998.

[2] Micah Adler and Michael Mitzenmacher. Towards compressing web graphs. In *Data Compression Conference*, pages 203–212, 2001. URL `citeseer.ist.psu.edu/adler00towards.html`.

[3] E. Amitay. Using common hypertext links to identify the best phrasal description of target web documents, 1998. URL `citeseer.nj.nec.com/amitay98using.html`.

[4] R. Amsler. Application of a citation-based automatic classification. Technical report, The University of Texas at Austin, Linguistics Research Center, 1972.

[5] Arvind Arasu, Junghoo Cho, Hector Garcia-Molina, Andreas Paepcke, and Sriram Raghavan. Searching the web. *ACM Transactions on Internet Technology (TOIT)*, 1(1):2–43, August 2001.

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

[7] Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proc. 13th ACM-SIAM Symp. on Discrete Algorithms*, pages 623–32, 2002.

[8] Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *Journal of Computer and System Sciences*, 68(4):702–32, 2004.

[9] Ziv Bar-Yossef. *The Complexity of Massive Data Set Computations*. PhD thesis, UC Berkeley, 2002.

[10] Ziv Bar-Yossef and Maxim Gurevich. Efficient search engine measurements. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 401–410, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-654-7. doi: http://doi.acm.org/10.1145/1242572.1242627.

[11] Ziv Bar-Yossef, Alexander Berg, Steve Chien, Jittat Fakcharoenphol, and Dror Weitz. Approximating aggregate queries about web pages via random walks. In *Proceedings of the 26th International Conference on Very Large Data Bases*, pages 535–544. Morgan Kaufmann Publishers Inc., 2000. ISBN 1-55860-715-3.

[12] Ziv Bar-Yossef, Andrei Z. Broder, Ravi Kumar, and Andrew Tomkins. Sic transit gloria telae: Towards an understanding of the web's decay. In *Proceedings of the 13th World Wide Web Conference (WWW)*, pages 328–337. ACM Press, 2004. ISBN 1-58113-844-X. doi: http://doi.acm.org/10.1145/988672.988716.

[13] Luiz André Barroso, Jeffrey Dean, and Urs Hölzle. Web Search for a Planet: The Google Cluster Architecture. *IEEE Micro*, 23(2):22–28, 2003. ISSN 0272-1732. doi: http://dx.doi.org/10.1109/MM.2003.1196112.

[14] F. Bodon. Adatbányászati algoritmusok. Technical report, Budapesti Műszaki és Gazdaságtudományi Egyetem, 2004-2009.

[15] P. Boldi, B. Codenotti, M. Santini, and S. Vigna. Ubicrawler: A scalable fully distributed web crawler. *Software: Practice & Experience*, 34(8): 721–726, 2004. URL `http://citeseer.ist.psu.edu/650719.html`.

[16] Paolo Boldi and Sebastiano Vigna. The WebGraph framework I: Compression techniques. Technical Report 293-03, Universita di Milano, Dipartimento di Scienze dell'Informazione, 2003.

[17] Paolo Boldi and Sebastiano Vigna. The webgraph framework I: Compression techniques. In *Proceedings of the 13th World Wide Web Conference (WWW)*, pages 595–602. ACM Press, 2004. ISBN 1-58113-844-X. doi: http://doi.acm.org/10.1145/988672.988752.

[18] B. Bollobás. *Extremal Graph Theory.* Academic Press, New York, 1978.

[19] Alan Borodin, Gareth O. Roberts, Jeffrey S. Rosenthal, and Panayiotis Tsaparas. Finding authorities and hubs from link structures on the world wide web. In *Proceedings of the 10th World Wide Web Conference (WWW)*, pages 415–429, 2001. URL `citeseer.nj.nec.com/borodin01finding.html`.

[20] E. Brewer. Lessons from giant-scale services. URL `citeseer.nj.nec.com/476298.html`.

[21] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1-7): 107–117, 1998. URL `citeseer.nj.nec.com/brin98anatomy.html`.

[22] Andrei Z. Broder. On the Resemblance and Containment of Documents. In *Proceedings of the Compression and Complexity of Sequences (SEQUENCES'97)*, pages 21–29, 1997. ISBN 0-8186-8132-2.

[23] 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. URL `citeseer.ist.psu.edu/broder98minwise.html`.

[24] A. L. Buchsbaum, R. Giancarlo, and J. R. Westbrook. On finding common neighborhoods in massive graphs. *Theoretical Computer Science*, 299(1-3):707–18, 2004.

[25] A. L. Buchsbaum, R. Giancarlo, and B. Racz. New results for finding common neighborhoods in massive graphs in the data stream model. *Theoretical Computer Science*, 407(1-3):302–309, 2008. ISSN 0304-3975. doi: http://dx.doi.org/10.1016/j.tcs.2008.06.056.

[26] Carlos Castillo. *Effective Web Crawling*. PhD thesis, University of Chile, 2004.

[27] Soumen Chakrabarti, Byron E. Dom, and Piotr Indyk. Enhanced hypertext categorization using hyperlinks. In Laura M. Haas and Ashutosh Tiwary, editors, *Proceedings of SIGMOD-98, ACM International Conference on Management of Data*, pages 307–318, Seattle, US, 1998. ACM Press, New York, US. URL `citeseer.nj.nec.com/chakrabarti98enhanced.html`.

[28] P. Chan, W. Fan, A. Prodromidis, and S. Stolfo. Distributed data mining in credit card fraud detection. *IEEE Intelligent Systems*, 14(6):67–74, 1999.

[29] Moses Charikar, A. Broder, A. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *Journal of Computer Systems and Sciences*, 60(3):630–659, 2000.

[30] Yen-Yu Chen, Qingqing Gan, and Torsten Suel. I/O-efficient techniques for computing PageRank. In *Proceedings of the 11th Conference on Information and Knowledge Management (CIKM)*, pages 549–557, 2002. ISBN 1-58113-492-4. doi: http://doi.acm.org/10.1145/584792.584882.

[31] Edith Cohen. Size-estimation framework with applications to transitive closure and reachability. *J. Comput. Syst. Sci.*, 55(3):441–453, 1997. ISSN 0022-0000. doi: http://dx.doi.org/10.1006/jcss.1997.1534.

[32] C. Cortes, K. Fisher, D. Pregibon, A. Rogers, and F. Smith. Hancock: A language for analyzing transactional data streams. *ACM Transactions on Programming Languages and Systems*, 26(2):301–38, 2004.

[33] Tom Costello, Anna Patterson, and Russell Power. The cuil web search engine., 2008-. URL `http://www.cuil.com`.

[34] Tom Costello, Anna Patterson, and Russell Power. The Cuil web search engine, FAQ., Downloaded on 2009-01-11. URL `http://www.cuil.com/info/faqs`.

[35] Marco Cristo, Pável Calado, Edleno Silva de Moura, Nivio Ziviani, and Berthier Ribeiro-Neto. Link information as a similarity measure in web classification. In *String Processing and Information Retrieval*, pages 43–55. Springer LNCS 2857, 2003.

[36] Maurice de Kunder. The size of the world wide web. URL `http://www.worldwidewebsize.com/`.

[37] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proceedings of the 6th USENIX Symposium on Operating System Design and Implementation (OSDI)*, San Francisco, CA, USA, 2004. USENIX Association.

[38] Jeffrey Dean and Monika R. Henzinger. Finding related pages in the World Wide Web. In *Proceedings of the 8th World Wide Web Conference (WWW)*, pages 1467–1479, 1999. URL `citeseer.nj.nec.com/dean99finding.html`.

[39] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990. URL `citeseer.nj.nec.com/deerwester90indexing.html`.

[40] Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. Rank aggregation methods for the Web. In *Proceedings of the 10th International World Wide Web Conference (WWW)*, pages 613–622, Hong Kong, 2001.

[41] Nadav Eiron and Kevin S. McCurley. Locality, hierarchy, and bidirectionality in the web. In *Second Workshop on Algorithms and Models for the Web-Graph (WAW 2003)*, 2003.

[42] M. Elkin and J. Zhang. Efficient algorithms for constructing $(1+\epsilon,\beta)$-spanners in the distributed and streaming models. In *Proc. 23rd ACM Symp. of Principles of Distributed Computing*, pages 160–8, 2004.

[43] Ronald Fagin, Ravi Kumar, Kevin S. McCurley, Jasmine Novak, D. Sivakumar, John A. Tomlin, and David P. Williamson. Searching the workplace web. In *Proceedings of the 12th International World Wide Web Conference (WWW)*, pages 366–375, 2003. URL `citeseer.ist.psu.edu/fagin03searching.html`.

[44] Ronald Fagin, Ravi Kumar, Kevin S. McCurley, Jasmine Novak, D. Sivakumar, John A. Tomlin, and David P. Williamson. Searching the workplace web. 2003.

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

[46] Ronald Fagin, Ravi Kumar, Mohammad Mahdian, D. Sivakumar, and Erik Vee. Comparing and aggregating rankings with ties. In *Proceedings of the 23rd ACM Symposium on Principles of Database Systems (PODS)*, 2004. URL `citeseer.ist.psu.edu/article/fagin03comparing.html`.

[47] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J. D. Ullman. Computing iceberg queries efficiently. In *Proc. 24th Int'l. Conf. on Very Large Databases*, pages 299–310, 1998.

[48] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. On graph problems in a semi-streaming model. In *Proc. 31st Int'l. Coll. on Automata, Languages, and Programming*, volume 3142 of *Lecture Notes in Computer Science*, pages 531–43. Springer, 2004.

[49] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. Graph distances in the streaming model: The value of space. In *Proc. 16th ACM-SIAM Symp. on Discrete Algorithms*, pages 745–54, 2005.

[50] Dániel Fogaras. Where to start browsing the web? In *Proceedings of the 3rd International Workshop on Innovative Internet Community Systems (I2CS)*, volume 2877/2003 of *Lecture Notes in Computer Science (LNCS)*, pages 65–79, Leipzig, Germany, June 2003. Springer-Verlag.

[51] Dániel Fogaras and Balázs Rácz. A scalable randomized method to compute link-based similarity rank on the web graph. In *Proceedings of the Clustering Information over the Web workshop (ClustWeb) in conjunction with the 9th International Conference on Extending Database Technology (EDBT)*, volume 3268/2004 of *Lecture Notes in Computer Science (LNCS)*, pages 557–567, Crete, Greece, March 2004. Springer-Verlag.

[52] Dániel Fogaras and Balázs Rácz. Scaling link-based similarity search. In *Proceedings of the 14th World Wide Web Conference (WWW)*, pages 641–650, Chiba, Japan, 2005.

[53] Daniel Fogaras and Balazs Racz. Practical algorithms and lower bounds for similarity search in massive graphs. *IEEE Transactions on Knowledge and Data Engineering*, 19(5):585–598, 2007. ISSN 1041-4347. doi: http://doi.ieeecomputersociety.org/10.1109/TKDE.2007.1008.

[54] Dániel Fogaras, Balázs Rácz, Károly Csalogány, and Tamás Sarlós. Towards Scaling Fully Personalized PageRank: Algorithms, Lower Bounds, and Experiments. *Internet Mathematics*, 2(3):333–358, 2005. Preliminary version from the first two authors appeared in WAW 2004.

[55] Z. Füredi. New asymptotics for bipartite Turán numbers. *Journal of Combinatorial Theory (A)*, 75(1):141–4, 1996.

[56] Small H. G. Co-citation in the scientific literature; a new measure of the relationship between two documents. *Journal of the American Society of Information Science*, 24:265–269, 1973.

[57] Volker Gaede and Oliver Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998. URL `citeseer.ist.psu.edu/article/gaede97multidimensional.html`.

[58] V. Ganti, J. Gehrke, and R. Ramakrishnan. Mining very large databases. *IEEE Computer*, 32(8):38–45, 1999.

[59] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, 1983.

[60] Google. Commercial search engine founded by the originators of PageRank. located at `http://www.google.com`.

[61] Antonio Gulli and Alessio Signorini. The indexable web is more than 11.5 billion pages. In *Proceedings of the 14th World Wide Web Conference (WWW), Special interest tracks and posters*, pages 902–903, 2005.

[62] Taher H. Haveliwala. Efficient computation of PageRank. Technical Report 1999-31, Stanford University, 1999.

[63] Taher H. Haveliwala. Topic-sensitive PageRank. In *Proceedings of the 11th World Wide Web Conference (WWW)*, pages 517–526, 2002. URL `citeseer.nj.nec.com/haveliwala02topicsensitive.html`.

[64] Taher H. Haveliwala. Efficient encodings for document ranking vectors. In *Proceedings of the 4th International Conference on Internet Computing (IC)*, pages 3–9, Las Vegas, Nevada, USA, 2003.

[65] Taher H. Haveliwala, Aristides Gionis, Dan Klein, and Piotr Indyk. Evaluating strategies for similarity search on the web. In *Proceedings of the 11th World Wide Web Conference (WWW)*, pages 432–442, 2002. ISBN 1-58113-449-5.

[66] Taher H. Haveliwala, Sepandar Kamvar, and Glen Jeh. An analytical comparison of approaches to personalizing PageRank. Technical Report 2003-35, Stanford University, 2003. URL `http://dbpubs.stanford.edu:8090/pub/2003-35`.

[67] M. R. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. Technical Report 1998-011, DEC SRC, 1998.

[68] Monika R. Henzinger, Allan Heydon, Michael Mitzenmacher, and Marc Najork. Measuring index quality using random walks on the Web. In *Proceedings of the 8th World Wide Web Conference (WWW)*, pages 213–225, 1999.

[69] Monika R. Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. Computing on data streams. In *External Memory Algorithms, DIMACS Book Series vol. 50.*, pages 107–118. American Mathematical Society, 1999. ISBN 0-8218-1184-3.

[70] Monika R. Henzinger, Allan Heydon, Michael Mitzenmacher, and Marc Najork. On near-uniform url sampling. In *Proceedings of the 9th international World Wide Web conference on Computer networks*, pages 295–308, 2000. doi: http://dx.doi.org/10.1016/S1389-1286(00)00055-4.

[71] Jun Hirai, Sriram Raghavan, Hector Garcia-Molina, and Andreas Paepcke. WebBase: A repository of web pages. In *Proceedings of the 9th World Wide Web Conference (WWW)*, pages 277–293, 2000.

[72] A. Hume, S. Daniels, and A. MacLellen. Gecko: Tracking a very large billing system. In *Proc. USENIX Ann. Technical Conference*, pages 93–106, 2000.

[73] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the 30th ACM Symposium on Theory of Computing (STOC)*, pages 604–613, 1998. URL `citeseer.ist.psu.edu/article/indyk98approximate.html`.

[74] Glen Jeh and Jennifer Widom. SimRank: A measure of structural-context similarity. In *Proceedings of the 8th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 538–543, 2002. URL `http://dbpubs.stanford.edu:8090/pub/2001-41`.

[75] Glen Jeh and Jennifer Widom. Scaling personalized web search. In *Proceedings of the 12th World Wide Web Conference (WWW)*, pages 271–279. ACM Press, 2003. ISBN 1-58113-680-3. doi: http://doi.acm.org/10.1145/775152.775191.

[76] Thorsten Joachims. Optimizing search engines using clickthrough data. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142, New York, NY, USA, 2002. ACM. ISBN 1-58113-567-X. doi: http://doi.acm.org/10.1145/775047.775067.

[77] Sepandar Kamvar, Taher H. Haveliwala, Christopher Manning, and Gene Golub. Exploiting the block structure of the web for computing PageRank. Technical Report 2003-17, Stanford University, 2003. URL `citeseer.ist.psu.edu/kamvar03exploiting.html`.

[78] Sepandar D. Kamvar, Taher H. Haveliwala, Christopher D. Manning, and Gene H. Golub. Extrapolation methods for accelerating PageRank computations. In *Proceedings of the 12th World Wide Web Conference (WWW)*, pages 261–270. ACM Press, 2003. ISBN 1-58113-680-3. doi: http://doi.acm.org/10.1145/775152.775190.

[79] Maurice G. Kendall. *Rank Correlation Methods*. Hafner Publishing Co., New York, 1955.

[80] M.M. Kessler. Bibliographic coupling between scientific papers. *American Documentation*, 14:10–25, 1963.

[81] J. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–32, 1999.

[82] Jon Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999. URL `citeseer.nj.nec.com/article/kleinberg98authoritative.html`.

[83] I. Kremer, N. Nisan, and D. Ron. On randomized one-round communication complexity. *Computational Complexity*, 8(1):21–49, 1999. *Errata*, 10(4):314–5, 2001.

[84] Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, 1997. ISBN 0-521-56067-5.

[85] Hsin-Tsang Lee, Derek Leonard, Xiaoming Wang, and Dmitri Loguinov. Irlbot: scaling to 6 billion pages and beyond. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 427–436, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-085-2. doi: http://doi.acm.org/10.1145/1367497.1367556.

[86] Ronny Lempel and Shlomo Moran. Rank stability and rank similarity of link-based web ranking algorithms in authority connected graphs. In *Second Workshop on Algorithms and Models for the Web-Graph (WAW 2003)*, 2003.

[87] Stefano Leonardi, editor. *Algorithms and Models for the Web-Graph: Third International Workshop, WAW 2004, Rome, Italy, October 16, 2004, Proceeedings*, volume 3243 of *Lecture Notes in Computer Science*, 2004. Springer. ISBN 3-540-23427-6.

[88] C.-S. Li and R. Ramaswami. Automatic fault detection, isolation, and recovery in transparent all-optical networks. *Journal of Lightwave Technology*, 15(10):1784–93, 1997.

[89] David Liben-Nowell and Jon Kleinberg. The link prediction problem for social networks. In *Proceedings of the 12th Conference on Information and Knowledge Management (CIKM)*, pages 556–559, 2003. ISBN 1-58113-723-0. doi: http://doi.acm.org/10.1145/956863.956972.

[90] Wangzhong Lu, Jeannette Janssen, Evangelos Milios, and Nathalie Japkowicz. Node similarity in networked information spaces. In *Proceedings of the Conference of the Centre for Advanced Studies on Collaborative research*, page 11, 2001.

[91] Ulrich Meyer, Peter Sanders, and Jop Sibeyn. *Algorithms for Memory Hierarchies, Advanced Lectures.* Springer-Verlag, Berlin, 2003.

[92] S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.

[93] N. Nisan and E. Kushilevitz. *Communication Complexity.* Cambridge University Press, Cambridge, UK, 1997.

[94] Open Directory Project (ODP). Open Directory Project (ODP). `http://www.dmoz.org`.

[95] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford University, 1998. URL `citeseer.nj.nec.com/page98pagerank.html`.

[96] Christopher R. Palmer, Phillip B. Gibbons, and Christos Faloutsos. ANF: A fast and scalable tool for data mining in massive graphs. In *Proceedings of the 8th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 81–90. ACM Press, 2002. ISBN 1-58113-567-X. doi: http://doi.acm.org/10.1145/775047.775059.

[97] Paat Rusmevichientong, David M. Pennock, Steve Lawrence, and C. Lee Giles. Methods for sampling pages uniformly from the world wide web. In *AAAI Fall Symposium on Using Uncertainty Within Computation*, pages 121–128, 2001. URL `citeseer.ist.psu.edu/article/rusmevichientong01methods.html`.

[98] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 24:513–523, 1988.

[99] G. Salton and M. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, 1983.

[100] Tamás Sarlós, András A. Benczúr, Károly Csalogány, Dániel Fogaras, and Balázs Rácz. To randomize or not to randomize: Space optimal summaries for hyperlink analysis. In *Proceedings of the 15th International World Wide Web Conference (WWW)*, pages 297–306, 2006. Full version available at http://www.ilab.sztaki.hu/websearch/Publications/.

[101] Pavan Kumar C. Singitham, Mahathi S. Mahabhashyam, and Prabhakar Raghavan. Efficiency-quality tradeoffs for vector score aggregation. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB)*, pages 624–635, 2004.

[102] S. Stolfo, W. Fan, W. Lee, A. Prodromidis, and P. Chan. Cost-based modeling for fraud and intrusion detection: Results from the JAM project. In *Proc. DARPA Information and Survivability Conference and Exposition*, volume 2, pages 130–144, 2000.

[103] Torsten Suel and V. Shkapenyuk. Design and implementation of a high-performance distributed web crawler. In *Proceedings of the 3rd IEEE International Conference on Data Engineering*, February 2002.

[104] Danny Sullivan. Search engine size wars & google's supplemental results. URL `http://searchenginewatch.com/3071371`.

[105] S. G. Tzafestas and P. J. Dalianis. Fault diagnosis in complex systems using artificial neural networks. In *Proc. 3rd IEEE Conf. on Control Applications*, volume 2, pages 877–82, 1994.

[106] J. D. Ullman. The MIDAS data-mining project at Stanford. In *Proc. 1999 IEEE Symp. on Database Engineering and Applications*, pages 460–4, 1999.

[107] J. S. Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing Surveys*, 33(2):209–71, 2001.

[108] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing gigabytes (2nd ed.): Compressing and indexing documents and images.* Morgan Kaufmann Publishers Inc., 1999. ISBN 1-55860-570-3.

[109] Gui-Rong Xue, Chenxi Lin, Qiang Yang, WenSi Xi, Hua-Jun Zeng, Yong Yu, and Zheng Chen. Scalable collaborative filtering using cluster-based smoothing. In *SIGIR '05: Proceedings of the 28th annual international*

ACM SIGIR conference on research and development in information retrieval, pages 114–121, New York, NY, USA, 2005. ACM Press. ISBN 1-59593-034-5. doi: http://doi.acm.org/10.1145/1076034.1076056.

[110] K. Zarankiewicz. Problem P 101. *Colloquium Mathematicum*, 2:301, 1951.

Index of Citations

Acknowledgement

Above all I would like to thank my co-author and at the time fellow PhD student Dániel Fogaras for introducing me to the field of link-based search algorithms and the several years of fruitful collaboration that followed.

I would like to say special thanks to the leading research scientists of the Data Mining and Web Search Group: András Benczúr, who was my advisor, and András Lukács. They have supported my career during my time at the Hungarian Academy of Sciences, and supplied me with a constant stream of very diverse tasks ranging from CS research through software engineering of large-scale infrastructure systems to R&D grants work and public procurement, which have developed a versatile set of skills that turned out to be very helpful in my further career. I am thankful to my colleagues and co-authors in the research group, Károly Csalogány and Tamás Sarlós for implementing some of our algorithms and doing experimental evaluation, and for all the feedback they gave during the research work that kept us on the right course. The quality of our manuscripts and papers were singificantly improved by comments from the aforementioned people, as well as Lajos Rónyai, Dániel Marx, Glen Jeh, Andrew Twigg, Adam L. Buchsbaum and Raffaele Giancarlo. I would like to especially thank to Ferenc Bodon, with whom I worked together for a long time on a field of research outside this thesis.

I am especially grateful to Professors Lajos Rónyai, András Recski and Bálint Tóth, who, as the heads of the three departments related to my field (Algebra, Computer Science and Probability, respectively) have supported my development with advice, challenges and opportunities.