Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

PaperRank for Literature Research

Group Project

Fabian Mentzer, Timon Ruban, Jan Schulze

mentzerf@ee.ethz.ch, truban@ee.ethz.ch, schulzej@ee.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zurich

Supervisors:
Tobias Langner, Jochen Seidel
Prof. Dr. Roger Wattenhofer

June 14, 2014

Abstract

While most common literature search engines use search queries based on keywords, we are interested in finding important as well as relevant literature based on other literature. To accomplish this, we introduce the PaperRank algorithm, an adapted version of the PageRank algorithm. PaperRank attributes a single score of absolute importance to scientific publications. To find the best configuration for the parameters of the PaperRank, we conduct an empirical evaluation. We go on to describe a search algorithm that uses PaperRank to find and rank relevant literature given one or more input papers. In comparing our search engine to others, we show how our approach of only using paper metadata – information about references and authors – is already producing meaningful results.

Contents

Chapter 1

Introduction

Most will agree that conducting literature research is an important, yet mundane exercise. Fortunately, as scientific publications are becoming more and more accessible through the web, search engines can be used to simplify this task.

In this group project we set out to develop a convenient and easy-to-use application aimed at facilitating literature research. While most common literature search engines use search queries based on keywords, we are interested in finding important as well as relevant literature based on other literature. The key issue we want to address is how to rank scientific publications by their importance[1]. In particular, we are interested to see if it is possible to adapt the popular PageRank algorithm, used to rank webpages, to our problem of ranking papers. To build a useful search engine, we identify means to narrow down the search space and find not just important, but relevant papers. The adapted PageRank, henceforth called *PaperRank*, can then in turn rank these search results.

Using papers as inputs for the search engine is especially useful in situations like the following: One might be looking into a new research field in which one is not yet familiar with the distinct scientific terminology and thus does not know which terms to search for. In this case an explicit keyword or author search, as offered by most search engines, is not of much use. Sometimes this holds even if one already has expertise in the topic in question. This is because different papers might use different terminology describing the same thing. Then, the search for specific keywords might exclude literature that is actually relevant. For instance, consider the two heavily related concepts of "Random Walks" and "Brownian Motion". Explicitly searching for one term or the other can rule out many interesting papers.

In this report we document the results of our group project. We first give a short overview of other applications used to conduct literature research (Chapter 2). Next we introduce PaperRank and describe it in detail (Chapter 3). In Chapter 4 we discuss how the search algorithm works. Lastly, we present a conclusion to our work and give an outlook on possible future research (Chapter 5).

[1] We use the terms paper, publication and article interchangeably throughout this report.

Chapter 2

Related Work

2.1 Google Scholar & Co.

We give a short overview of various existing ways to conduct literature research. Foremost it is important to distinguish between bibliographic databases and more sophisticated search engines used for literature research.

2.1.1 Bibliographic Databases

Bibliographic databases are usually run and updated by a staff that collects and indexes scholarly content. The consequence is that they offer high-quality content, but are often only accessible through paid subscriptions. Since there usually is no ranking employed, these databases are mostly addressed at experts who know what specific scientific keywords or authors to search for. Examples for bibliographic databases are the Web of Science[1] or the online collections of articles, papers and books found in university libraries (e.g. the ETH Library[2]).

2.1.2 Search Engines

In contrast to bibliographic databases, search engines for scholarly literature especially aid researchers in the process of finding new, relevant literature. They usually allow for text-based search queries and display a ranked list of results. The ranking is often based on metadata of the scientific publications like the count of references or when and where it was published. We describe two tools that allow for such searches.

[1] http://wokinfo.com
[2] http://www.library.ethz.ch/de

Google Scholar

Google Scholar is a search engine that can be used to search for scholarly literature. It provides access to abstracts and metadata of scientific papers, and often times even the full text article. It can also be used to analyze the references of authors and their publications.

After entering a search query a list of results is displayed in a ranked order. A *Related Articles* function, as the name implies, shows articles related to one of the results.

While Google does not reveal the methods they use to rank the papers and to find related papers, they paraphrase their approach in the following way:

> *"Google Scholar aims to rank documents the way researchers do, weighing the full text of each document, where it was published, who it was written by, as well as how often and how recently it has been cited in other scholarly literature."* [Goo]

Mendeley

In their own words, Mendeley "*is a free reference manager [...] that can help you organize your research, collaborate with others online, and discover the latest research.*" [Men]. Its central feature is managing and organizing references, but it also provides a search engine that lets one explore their crowd-sourced research catalogs. In particular, in their standalone desktop application they offer the functionality of finding relevant, related research based on selected papers. As to how they find relevant literature, Mendeley does not offer an explanation.

Chapter 3

Ranking Scientific Publications

First we spell out the basics of the PageRank algorithm and its interpretation, the Random Surfer Model, in Section 3.2. We then explain the train of thought leading up to the adapted version of the PageRank algorithm: the PaperRank (Section 3.3). After this we give a detailed description thereof (Section 3.4) and provide insight into our implementation (Section 3.5). In Section 3.6 we explain how to find good parameters for the PaperRank algorithm and evaluate our findings.

3.1 Preliminaries

Markov Chains

An introduction to stochastic processes and Markov chains is given in [SS01]. Another review of Markov chain theory is also presented in [BG92]. Here, we only state the results that will be needed later on.

Definition 3.1. Consider a discrete-time stochastic process $(X_t,\ t \in \mathbb{N}_0)$ and a finite, countable set $S = \{0, \ldots, n-1\}$. The stochastic process is called a *finite-state Markov chain* on the *state space* S if the random variables $X_0, X_1, X_2, \ldots$ only take on values in S and X_{t+1} only depends on X_t, namely if $\forall\ t \geq 0$ and $s_0, \ldots, s_t, s_{t+1} \in S$:

$$\Pr[X_{t+1} = s_{t+1} \mid X_0 = s_0, X_1 = s_1, \ldots, X_t = s_t] = \Pr[X_{t+1} = s_{t+1} \mid X_t = s_t]$$

If in addition

$$p_{ij} := \Pr[X_{t+1} = i \mid X_t = j], \quad \forall\ i, j \in S$$

does not depend on t, the Markov chain is called *time-homogeneous* and a state transition matrix P can be defined as

$$P = (p_{ij})_{0 \leq i,j \leq n}\ .$$

Let

$$\vec{q}^{\,(0)} = (q_0^{(0)}, \ldots, q_{n-1}^{(0)})$$

be the probability distribution of X_0 (i.e. $\Pr[X_0 = i] = q_i^{(0)}, \forall\ i \in S$). Then the probability distribution of X_{t+1} can be calculated as

$$\vec{q}^{\,(t+1)} = \vec{q}^{\,(t)} \cdot P, \quad t \geq 0$$

or as

$$\vec{q}^{\,(t+1)} = \vec{q}^{\,(0)} \cdot P^{t+1}, \quad t \geq 0\ .$$

Definition 3.2. A probability vector $\vec{\pi}$ (with $\pi_j \geq 0, j \in S$ and $\sum_{j \in S} \pi_j = 1$) is called the *stationary distribution* of the Markov chain with state transitition matrix P if

$$\vec{\pi} = \vec{\pi} \cdot P\ .$$

We now state an important theorem on ergodic Markov chains. A definition of ergodicity and a proof of the theorem can be found in [SS01].

Theorem 3.3. *If a Markov chain* $(X_t,\ t \in \mathbb{N}_0)$ *on the state space* S *is ergodic (aperiodic and irreducible), it has a unique stationary distribution* $\vec{\pi}$ *and it converges towards this stationary distribution independently of the initial probability distribution of* X_0 *(i.e.* $\lim_{t\to\infty} \vec{q}^{\,(t)} = \vec{\pi}$*).*

A simple (sufficient) testing condition for ergodicity is the following: if all elements of the state transition matrix P are strictly positive, the Markov chain is ergodic.

3.2 The PageRank Algorithm

The *PageRank* algorithm was first introduced in [PBMW99]. A different explanation is given in [Hav02].

3.2.1 Web Graph

The PageRank algorithm is used to rank websites using the underlying structure of the world wide web. To do this the Internet is modeled as a directed graph $G_I = (V, A)$. Each node $v \in V$ represents a web page and every edge $a = (u, v)$ (with $a \in A$ and $u, v \in V$) corresponds to a hyperlink on web page u referring to web page v. The total number of web pages is given by $n = |V|$. In addition, we label each web page 1 through n (i.e. $V = \{1, \ldots, n\}$). The number of edges $a \in A$ emanating from node u is called *out-degree* of node u and is denoted by $\deg^+(u)$.

The goal of PageRank is to assign a certain rank to each web page $v \in V$ that determines the absolute importance of v among all the web pages. If we define this process of assigning a rank to a web page as the mapping

$$\mathbf{r} : V \to \mathbb{R}_+$$

we can denote the output of PageRank by the *rank vector*

$$\vec{r} = (r(1), \ldots, r(n)) \in \mathbb{R}_+^{1\times n} \ .$$

The basic idea of the algorithm is that a web page u recommends another web page v if it links to it. Naturally, a recommendation by a more important page should be worth more than a recommendation by a less important page. Therefore the algorithm recursively takes the rank and the number of recommendations (i.e. $\deg^+(u)$) of the recommending page u into account.

An intuitive basis for the calculation of the rank vector $\vec{r}$ is given by the *Random Surfer Model* that was introduced in [PBMW99].

3.2.2 Random Surfer Model

The *random surfer* explores the Internet and jumps from web page to web page by consecutively clicking random hyperlinks. This is equivalent to doing a random walk on the graph G_I. However, every now and then (with a certain probability α) the random surfer becomes bored and jumps to a random web page (i.e. enters a completely new URL).

If we model this behavior of the random surfer as a Markov chain on the state space V, the rank vector $\vec{r}$ is given as the stationary distribution of said Markov chain. The corresponding state transition matrix M' is given by

$$M' = (1-\alpha)\cdot M + \alpha \cdot B, \qquad \alpha \in [0,1]$$

where $B \in R^{n\times n}$ is the uniform matrix ($b_{ij} = \frac{1}{n}, \forall\ i,j \in V$) and M is defined as follows:

$$M = \begin{pmatrix} m_{11} & m_{12} & \cdots & m_{1n} \\ m_{21} & m_{22} & \cdots & m_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n1} & m_{n2} & \cdots & m_{nn} \end{pmatrix} \in \mathbb{R}^{n\times n}, \quad m_{uv} = \begin{cases} 1/\deg^+(u) & \text{if } (u,v) \in A \\ 0 & \text{else} \end{cases}$$

Note that M corresponds to the state transition matrix of a random walk on the graph $G_I = (V, A)$.

The rank vector $\vec{r}$ (i.e. the stationary distribution of the Markov chain with state transition matrix M') can now be calculated as the limit for $t \to \infty$ of the probability distribution $\vec{r}^{\,t+1}$ that is defined by

$$\vec{r}^{\,(t+1)} = \vec{r}^{\,(t)} \cdot M', \quad t \geq 0$$

Bear in mind that because of B all elements of M' are strictly positive (i.e. $m'_{uv} > 0, \forall\ u, v \in V$) and it follows that the Markov chain with state transition matrix M' is ergodic. Using Theorem 3.3 it immediately follows that the rank vector $\vec{r}$ exists and $\vec{r}^{\,(t)}$ converges towards it independently of $\vec{r}^{\,(0)}$ (i.e. $\vec{r} = \lim_{t\to\infty} \vec{r}^{\,(t)}$).

Since it does not matter what the initial vector $\vec{r}^{\,(0)}$ looks like, as long as it is a valid probability distribution, we simply choose the uniform vector $\vec{r}^{\,(0)} = (\frac{1}{n}, \ldots, \frac{1}{n})$.

Dangling Links

A *dangling link* is a link that points to a web page u with no outgoing links (i.e. $\deg^+(u) = 0$). It is not obvious how to redistribute the rank of this web page. In the calculation of the rank vector, rank would get lost for the web page with no outgoing links (there would be an all zero row in the transition matrix M). This is called a *rank sink*. In the original PageRank version suggested in [PBMW99] all dangling links are removed recursively to avoid this predicament.

Note that if there are no dangling links, the L^1-norm is preserved in the calculation steps of the PageRank (i.e. $\|\vec{r}^{\,(t+1)}\|_1 = \|\vec{r}^{\,(t)}\|_1$). This can be seen, as follows:

Proof.

$$\begin{aligned}\|\vec{r}^{\,(t+1)}\|_1 &= \sum_{u=1}^{n} |r_u^{(t+1)}| = \sum_{u=1}^{n} r_u^{(t+1)} = \sum_{u=1}^{n}\sum_{v=1}^{n} r_v^{(t)} m'_{vu} \\ &= \sum_{v=1}^{n} r_v^{(t)} \sum_{u=1}^{n} m'_{vu} = \sum_{v=1}^{n} r_v^{(t)} = \sum_{v=1}^{n} |r_v^{(t)}| = \|\vec{r}^{\,(t)}\|_1\end{aligned}$$

□

where we used that all elements of the rank vector are nonnegative and that all rows of M and B sum to 1 (because there are no dangling links and thereby no all zero rows in M) and thus also all rows of M' sum to one (i.e. $\sum_{u=1}^{n} m'_{vu} = 1$).

3.3 Adapting the PageRank Algorithm

The PageRank algorithm is very useful for ranking web pages. It can be calculated offline, is resistant to most manipulations and gives a single number to the importance of a web page. It seems reasonable to try and apply such a successful algorithm in a different context. However, we can not directly apply it to the problem of ranking scientific publications.

3.3.1 Graph of the Web vs. Graph of a Citation Network

The perhaps natural extension of the PageRank algorithm to our context would be to use references as the "hyperlinks" between two papers. However, the graph of a citation network (with papers as nodes and references as directed edges) causes some immediate issues. Unlike the graph of the World Wide Web the graph of a citation network has a tree-like structure – new publications almost exclusively cite older publications. While it is possible that two collaborating research groups writing their scientific papers at the same time will cite each other, only very few (short) cycles will be introduced to the citation network, as time elapses and the graph grows.

Recursively removing all of the dangling links, as in the original PageRank, would now result in the removal of almost the entire graph. Even if this issue can be resolved (it can!), the structure of the graph still leads to the rank distribution being a one-way road. Thus older papers will accumulate much more rank than newer publications.

3.3.2 A Random Literature Searcher

Keeping these challenges in mind, we now turn back to the intuition behind the PageRank algorithm, the Random Surfer Model. The Random Surfer surfs through the web by clicking random hyperlinks (following the recommendations of the respective websites) and from time to time jumping to a random web page. In the context of literature research it seems to make little sense to only follow one reference after the other. Our goal is therefore to teach the Random Surfer a thing or two about how to conduct literature research and turn him into a *Random Literature Searcher*.

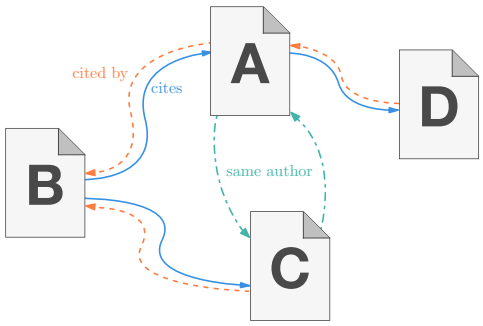

Figure 3.1: A graph showing the relationship between scientific publications

Looking at Figure 3.1 we see that there is more information to be gained from paper A than just its references. It seems to be reasonable to assume that papers B and C are also related to A in some way. The Random Literature Searcher will try to make use of this information. Essentially he shall not only

jump to references of a paper, but also to papers citing the current paper (we call them *cited-bys*) as well as other papers by the same authors[1]. This can be easily accomplished by introducing new directed edges to the graph. Note that the tree-like structure that caused problems earlier on is now no longer an issue as there are many new edges in the graph.

However, are the papers written by the same author more or less relevant than the cited-bys? Does it really make sense for an important paper to recommend all papers that are citing it the same way it recommends its references? It is not clear a priori if the references, cited-bys and papers by the same author are equally relevant. To account for this problem we attribute different weights to the edges pointing to the references, cited-bys and papers by the same authors. This corresponds to the Random Literature Searcher choosing between the different edges with different probabilities. We give a more formal explanation in the next section.

3.4 The PaperRank Algorithm

3.4.1 Setup

Similar to Section 3.2.1 we first define a graph $G_P = (P, E)$ to model our network of scientific publications. The set of nodes P represents the set of all scientific papers labeled 1 through n, where $n = |P|$. E is the set of directed edges that are labeled as *reference*, *cited-by* or *sharing one or more authors* and are denoted by:

$$e := (p, q)_i \in E, \quad p, q \in P \text{ and } i \in \{r, c, a\}$$

It can be written as the union of three disjoint sets of labeled, directed edges $E = E_r \cup E_c \cup E_a$, where $\forall\ p, q \in P$:

$(p, q)_r \in E_r$	if p cites q	*(reference)*
$(p, q)_c \in E_c$	if q cites p	*(cited-by)*
$(p, q)_a \in E_a$	if p and q share at least one author	*(same author)*

Note that there can be two edges e_1 and e_2 pointing from p to q, as long as they have different labels. Further we define three different out-degrees as

$$\deg_i^+(p) := |\{q \in P : (p, q)_i \in E_i\}|, \quad p \in P \text{ and } i \in \{r, c, a\}$$

so that

$$\deg^+(p) = \deg_r^+(p) + \deg_c^+(p) + \deg_a^+(p)$$

[1] A technicality: To make things simpler, we consistently write "papers by the same authors" when we really mean "papers written by one or more of the authors that wrote the other paper", even if "the other paper" only has one author.

holds.

As with the PageRank (see Section 3.2) the output of the PaperRank algorithm will be a rank vector $\vec{r} \in \mathbb{R}_+^{1\times n}$ that defines a mapping

$$\textbf{rank} : P \to \mathbb{R}_+ \tag{3.1}$$

that assigns a rank to each paper $p \in P$. We will sometimes simply write the *PaperRank*, when we actually mean the rank vector $\vec{r}$ defining the **rank** mapping.

3.4.2 The Algorithm

We now explain step by step how we model the behavior of the Random Literature Searcher, described in Section 3.3.2, with the transition matrix M' of a Markov chain on state space P. In analogy to the PageRank we can then calculate the PaperRank as the stationary distribution of this Markov chain.

Just as the Random Surfer, the Random Literature Searcher becomes "bored" with a certain probability α and then looks at a completely random paper instead of continuing with his normal "routine" (modeled by M). This gives rise to

$$M' = (1-\alpha)\cdot M + \alpha \cdot B, \qquad \alpha \in [0,1] \tag{3.2}$$

where $B \in R^{n\times n}$ is the uniform matrix ($b_{ij} = \frac{1}{n}, \forall\, i,j \in P$) and M will be given next.

Recall that the Random Literature Searcher not only considers references when deciding what paper to look at next, but cited-bys and papers by the same authors as well. But instead of simply doing a random walk on the graph G_P, we differentiate between edges with different labels. The Random Literature Searcher first chooses what "kind" of paper it wants to look at next and then selects the next paper of this "kind" at random. Formally speaking, it conducts an experiment with sample space $\Omega = \{r, c, a\}$, where the probability of the three events is given by

$$\Pr[r] = 1-\beta-\gamma \tag{3.3}$$
$$\Pr[c] = \beta \tag{3.4}$$
$$\Pr[a] = \gamma \tag{3.5}$$

where $\beta, \gamma \in [0,1]$ so that $\beta + \gamma \leq 1$. It observes event $i \in \Omega$ and then chooses one of the edges labeled i uniformly at random. This can be written as:

$$M = (1-\beta-\gamma)\cdot M_r + \beta \cdot M_c + \gamma \cdot M_a \tag{3.6}$$

where M_i ($i \in \{r, c, a\}$) is defined as:

$$M_i = \begin{pmatrix} m_{i,11} & m_{i,12} & \cdots & m_{1n} \\ m_{i,21} & m_{i,22} & \cdots & m_{i,2n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{i,n1} & m_{i,n2} & \cdots & m_{i,nn} \end{pmatrix} \in \mathbb{R}^{n \times n}, \quad m_{i,pq} = \begin{cases} 1/\deg_i^+(p) & \text{if } (p,q)_i \in E_i \\ 0 & \text{else} \end{cases}$$

Note that M' is a valid state transition matrix (i.e. all rows sum to 1) only if $\deg_i^+(p) > 0$ for all $i \in \{r, c, a\}$[2]. We describe what to do when this is not the case in Section 3.4.3.

3.4.3 Treating Dangling Papers

In Section 3.2.2 we defined a dangling link as a link that points to a web page u with no outgoing links (i.e. $\deg^+(u) = 0$). In the context of scientific publications and the graph $G_P = (P, E)$ we define a *dangling paper* as a paper that has no references, no cited-bys or no papers by the same authors. That is $p \in P$ is a *dangling paper* if $\exists i \in \{r, c, a\} : \deg_i^+(p) = 0$.

Such a dangling paper p represents a rank sink, since the rank attributed to it is not fully redistributed. In other words, if the Random Literature Searcher is currently looking at p and observes a label i for which $\deg_i^+(p) = 0$, it would not know where to go next[3].

We propose three different modes to solve this problem. In Mode 1 the Random Literature Searcher will return to the dangling paper p. In Mode 2 it jumps to a completely random paper $q \in P$. In Mode 3 the Random Literature Searcher recognizes if it deals with a dangling paper p. In this case it will conduct a different experiment, where it only chooses between labels $i \in \{r, c, a\}$ for which $\deg_i^+(p) > 0$ holds. For instance, if there are no references it will only choose between the labels c and a for cited-bys and papers by the same authors. It then falls back to choosing its next paper of this "kind" at random, as it does there are no dangling papers.

For ease of notation, as in [Lap09], we use $I\{\text{statement}\}$ to denote the indicator of the statement. Namely

$$I\{\text{statement}\} = \begin{cases} 1 & \text{if statement is true,} \\ 0 & \text{if statement is false.} \end{cases}$$

Also note that a dangling paper p, for which (the stronger condition)

$$\deg^+(p) = \deg_r^+(p) + \deg_c^+(p) + \deg_a^+(p) = 0$$

[2]Otherwise, if $\deg_i^+(p) = 0$ the p-th row of M_i would be an all zero row. This leads to the p-th row of M' not summing to 1.

[3]Remember that the Random Literature Searcher irrevocably chooses what "kind" of paper it wants to visit next, before choosing one of these papers at random.

holds, contributes no information about its own importance or the importance of other papers. We therefore remove all such papers from the graph G_P.

Mode 1

In Mode 1 the Random Literature Searcher goes back to the same dangling paper p, if it observes a label $i \in \{r, c, a\}$ for which $\deg_i^+(p) = 0$. M' is as in (3.2) and M as in (3.6). However, the elements $m_{i,pq}$ of $M_i \in \mathbb{R}^{n\times n}$ now depend on $\deg_i^+(p)$ and are defined as follows, for $i \in \{r, c, a\}$

$$m_{i,pq} = \begin{cases} I\{(p,q)_i \in E_i\} \cdot \frac{1}{\deg_i^+(p)} & \text{if } \deg_i^+(p) > 0 \\ I\{u = v\} & \text{if } \deg_i^+(p) = 0 \ . \end{cases}$$

Mode 2

In Mode 2 the Random Literature Searcher jumps to a paper in P uniformly at random. M' is as in (3.2) and M as in (3.6). The elements $m_{i,pq}$ of $M_i \in \mathbb{R}^{n\times n}$ for $i \in \{r, c, a\}$ are defined as

$$M_i \in \mathbb{R}^{n\times n}, \quad m_{i,pq} = \begin{cases} I\{(p,q)_i \in E_i\} \cdot \frac{1}{\deg_i^+(p)} & \text{if } \deg_i^+(p) > 0 \\ \frac{1}{n} & \text{if } \deg_i^+(p) = 0 \ . \end{cases}$$

Mode 3

In Mode 3 we avoid the situation that the Random Literature Searcher chooses a label $i \in \{r, b, a\}$, for which $\deg_i^+(p) = 0$ holds, altogether. We denote by $\Pr[i \mid p]$ the probability that the Random Literature Searcher chooses label i, when it is currently looking at paper p. The values of $\Pr[i \mid p]$ are given in the following table.

$\Pr[r \mid p]$	$\Pr[c \mid p]$	$\Pr[a \mid p]$	$\deg_r^+(p)$	$\deg_c^+(p)$	$\deg_a^+(p)$
$1-\beta-\gamma$	β	γ	>0	>0	>0
0	$\frac{\beta}{\beta+\gamma}$	$\frac{\gamma}{\beta+\gamma}$	$=0$	>0	>0
$\frac{1-\beta-\gamma}{1-\beta}$	0	$\frac{\gamma}{1-\beta}$	>0	$=0$	>0
$\frac{1-\beta-\gamma}{1-\gamma}$	$\frac{\beta}{1-\gamma}$	0	>0	>0	$=0$
1	0	0	>0	$=0$	$=0$
0	1	0	$=0$	>0	$=0$
0	0	1	$=0$	$=0$	>0

Table 3.1: The values of $\Pr[i \mid p]$ depending on $\deg_i^+(p)$ for $i \in \{r, c, a\}$.

Remember that we remove all papers, where $\deg_i^+(p) = 0, \forall i \in \{r, c, a\}$, so we do not have to worry about this case.

Because in Mode 3 the probability, with which a certain label i is picked, is no longer constant over all papers, we do not break down the matrix M into the matrices M_r, M_c and M_a as in (3.6). Instead we directly define the elements m_{pq} for $M \in \mathbb{R}^{n \times n}$ as follows:

$$m_{pq} = I\{(p,q)_r \in E_r\} \cdot \frac{\Pr[r \mid p]}{\deg_r^+(p)} + I\{(p,q)_c \in E_c\} \cdot \frac{\Pr[c \mid p]}{\deg_c^+(p)} + I\{(p,q)_a \in E_a\} \cdot \frac{\Pr[a \mid p]}{\deg_a^+(p)}$$

Note that if $\deg_i^+(p) > 0, \forall i \in \{r, c, a\}$ and $p \in P$ the p-th row of M as defined above is equivalent to the p-th row of M as defined in (3.6).

3.4.4 Approximating the Rank Vector

Now that M' has been properly defined, we give an algorithm to approximate the stationary distribution (i.e. the rank vector) of the Markov chain with state transition matrix M'.

Let $\vec{s} \in \mathbb{R}_+^{1 \times n}$ be the uniform vector $\vec{s} = (\frac{1}{n}, \ldots, \frac{1}{n})$ and $\epsilon \in \mathbb{R}$ an arbitrary small threshold.

Algorithm 1 Rank Vector Approximation

$R^{(0)} \leftarrow \vec{s}$
$i \leftarrow 0$
repeat
  $R^{(i+1)} = R^{(i)} \cdot M'$
  $\delta \leftarrow \|R^{(i+1)} - R^{(i)}\|_1$
  $i \leftarrow i + 1$
until $\delta < \epsilon$
return $R^{(i)}$

3.5 Implementation

We implemented and tested the PaperRank algorithm using a MySQL database provided by the Distributed Computing Group at the ETH Zurich. The content was crawled from the ACM Digital Library[4], a collection of scientific publications in the field of computing and information technology.

We only use publications for which we have at least one reference, cited-by or paper by the same authors in the database. We also exclude all publications that

[4] http://dl.acm.org

do not have a Parent ID in the ACM Digital Library, where Parent ID refers to the ID of the conference proceeding it was published in. Books are an example of entries without parent ID. The Parent ID refers to the ID of the conference proceeding it was published in. This leaves us with a total of 1 674 288 papers, 7 060 598 references between these papers and 4 485 749 different authors.

We implemented the algorithm utilizing the Python[5] libraries NumPy[6] and SciPy[7] for array and matrix manipulations. The crucial part of the implementation, regarding time and memory usage, is calculating the matrix M. To allow for an efficient calculation of the PaperRank, we calculate a basic, incomplete version of M exactly once. When calculating a specific PaperRank we then load this information and adapt the matrix according to the desired mode and the desired parameter values. Further, to reduce memory usage, M is stored in a sparse format. However, in Mode 2 some rows of M are no longer sparse. To circumvent this problem and still keep a sparse matrix in the implementation, we keep track of all these rows and take them into account separately from M in the calculation of the PaperRank.

3.6 Evaluating PaperRank

In this Section we want to assess the PaperRank algorithm. A matter of particular interest is trying to find the best mode and the best values for α, β and γ. As an objective measure of how good the PaperRank is, we use the PaperRank to compute our own conference ranking and check if it correlates with other established conference rankings. We then compare the PaperRanks among themselves to examine the effect of changing individual parameters. A discussion of our finding forms the last part of the evaluation.

To compare two rank vectors to each other we will use the Kendall rank correlation coefficient τ (also called Kendall's τ). It is a non-parametric test that measures the similarity between two rankings. Depending on the number of inversion between the two rankings τ takes on values between -1 (perfect negative correlation) and 1 (perfect positive correlation). A detailed description is provided in [Abd07].

3.6.1 Comparison with Conference Rankings

For a sensible PaperRank we would expect papers from important conferences to have a high rank. Vice versa a paper with high rank will often times be from an important conference. We therefore compute the rank of a conference c by

[5] http://www.python.org
[6] http://www.numpy.org
[7] http://www.scipy.org

averaging the ranks of all papers that belong to c. More formally, we can express this as a mapping **conf-rank** that assigns a rank value to each conference. Let C be a set of conferences. Fix a conference $c \in C$ and let $\{p_1, \ldots, p_k\}$ be the set of all papers published at c, then **conf-rank** can be defined as:

$$\begin{aligned} \textbf{conf-rank} : C &\to \mathbb{R}_+ \\ c &\mapsto \frac{\sum_{i=1}^{k} rank(p_i)}{k} \quad , \end{aligned}$$

where **rank** is the function defined in (3.1).

We compare our conference ranking to the Microsoft [Mic] and the CORE [COR] conference ranking by calculating the Kendall rank correlation coefficient. The Microsoft ranking assigns a so called *field rating* ranging from 0 to 182 to the conferences in their database, whereas the CORE ranking divides the conferences into four categories (A*, A, B, C). For the CORE ranking we were able to match 145 711 papers for 2 686 conferences, for the Microsoft ranking we matched 228 948 papers for 5 260 conferences.

Mode	Kendall's τ	α	β	γ
1	0.37081	0.1	0.2	0.5
1	0.37045	0.1	0.1	0.6
1	0.37030	0.1	0.1	0.7
1	0.36747	0.1	0.2	0.4
1	0.35986	0.1	0.1	0.5
2	0.37033	0.1	0.1	0.6
2	0.36578	0.1	0.1	0.5
2	0.36505	0.1	0.1	0.7
2	0.35981	0.2	0.1	0.6
2	0.35907	0.1	0.2	0.5
3	0.36664	0.1	0.1	0.6
3	0.36439	0.1	0.1	0.7
3	0.36009	0.1	0.1	0.5
3	0.35974	0.1	0.2	0.5
3	0.35697	0.1	0.2	0.4

CORE Ranking

Mode	Kendall's τ	α	β	γ
1	0.25684	0.1	0.1	0.5
1	0.25496	0.1	0.1	0.4
1	0.25374	0.1	0.1	0.6
1	0.25281	0.1	0.2	0.3
1	0.25059	0.1	0.2	0.4
2	0.22367	0.1	0.1	0.5
2	0.22170	0.1	0.1	0.4
2	0.22108	0.1	0.1	0.6
2	0.21685	0.1	0.1	0.3
2	0.21553	0.2	0.1	0.5
3	0.24423	0.1	0.1	0.5
3	0.24308	0.1	0.1	0.4
3	0.24071	0.1	0.1	0.6
3	0.23890	0.1	0.1	0.3
3	0.23368	0.2	0.1	0.4

Microsoft Ranking

Figure 3.2: The best five results for every mode when comparing our conference ranking to the CORE ranking [COR] and the Microsoft Ranking [Mic]. We computed the PaperRanks, our according conference rankings and the Kendall rank correlation coefficient for all three modes, $\alpha = 0.1, 0.2 \ldots, 0.8$, $\beta = 0.1, 0.2 \ldots, 0.8$ and $\gamma = 0.1, 0.2, \ldots, 0.8$.

Looking at Figure 3.2 one can see that Mode 1 achieves the highest correlation coefficient τ for both rankings. Further the PaperRanks leading to the best results have very similar parameter values for α, β and γ. While γ varies between 0.3 and 0.7, α and β only vary between 0.1 and 0.2. This already gives us some insight on what the best parameter configuration looks like.

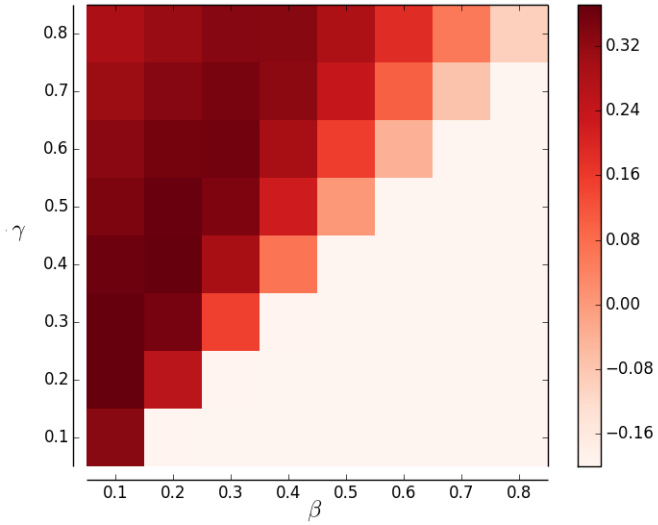

Figure 3.3: This figure shows a heatmap of the Kendall tau values of the comparison of our conference rank to the CORE ranking, plotted as a function of β and γ. For every pair of β and γ the conference ranking is calculated with the corresponding PaperRank and with $\alpha = 0.1$ and using Mode 1. The maximum kendall tau value is at $\beta = 0.2$, $\gamma = 0.5$ with a value of 0.3708. The plot is shown in 3D in Figure A.3 in Appendix A.

From Figure 3.3 we can observe that choosing other values for β and γ can lead to a worse performance. Especially choosing large values for β substantially decreases the correlation coefficient.

3.6.2 Adjusting the Parameters

Next we examine the impact that the individual parameters have on the PaperRank. We want to see how much the ranking is altered if we change α, β or γ. To do this we fix all parameters but one. We then change this parameter (in steps of 0.1), compute the different PaperRanks and compare them using the Kendall rank correlation coefficient. Note that changing β or γ, of course also means changing the parameter $1 - \beta - \gamma$ affiliated with the references.

Corresponding to Figure 3.4 varying the values of β and γ (i.e. the probabilities that the Random Literature Searcher chooses a reference, a cited-by or a paper by the same authors next) has a large effect on our ranking. Further, we see that the value of α does not have much influence on our ranking. However, it does have influence on the convergence properties of the PaperRank. We observed that higher values of α lead to a speedier convergence in the approximation of the rank vector (as in Algorithm 1). As we mostly care about the

ranking itself, we simply choose α to be equal to 0.1, but higher values could be chosen if one attaches more importance to the time needed to compute the PaperRank.

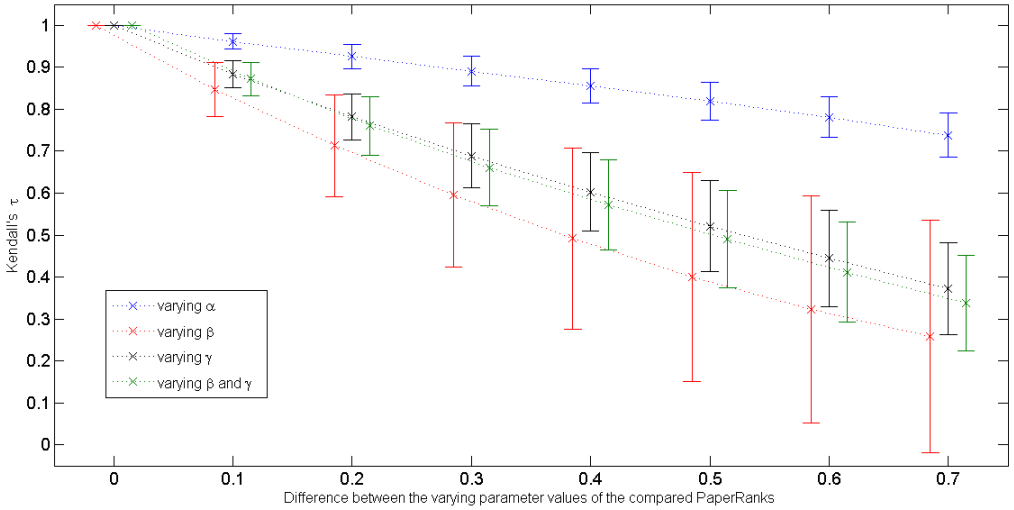

Figure 3.4: The plot shows the average Kendall rank correlation coefficient and its standard deviation computed for different pairs of PaperRanks. The parameter values we used in the calculation of a pair of PaperRanks are the same for all but one parameter. The x-axis indicates how much the value of this parameter differs between the two PaperRanks. For reasons of readability the curves are spread out.
The pairs of PaperRanks with varying α were calculated for all three modes, $\beta = 0.1, 0.2 \ldots, 0.8$ and $\gamma = 0.1, 0.2, \ldots, 0.8$. The pairs of PaperRanks with varying β were calculated for all three modes, $\alpha = 0.1$ and $\gamma = 0.1, 0.2, \ldots, 0.8$. The pairs of PaperRanks with varying γ were calculated for all three modes, $\alpha = 0.1$ and $\beta = 0.1, 0.2, \ldots, 0.8$. The pairs of PaperRanks with varying β and γ were calculated for all three modes, $\alpha = 0.1$, $\beta = 0.1, 0.2, \ldots, 0.8$ and $\gamma = 0.1, 0.2, \ldots, 0.8$. Here the parameter values used to calculate the two respective PaperRanks differ in β as well as γ, but in exchange $(1 - \beta - \gamma)$ was kept constant (i.e. $\Delta\beta = -\Delta\gamma$).

3.6.3 Discussion

Our evaluation reveals that there is a noticeable correlation between our conference ranking, computed using PaperRank, and the two other conference rankings if one uses the right parameters. However, the correlation coefficient is not strikingly high. An explanation for this could be that in the CORE and the Microsoft

rankings theoretical conferences (like STOC or FOCS) have a high rank. Compared to conferences of a more practical nature (like INFOCOM or SIGGRAPH) theses conferences have a smaller number of publications and cited-bys. Unlike in the CORE and Microsoft rankings, this most likely results in a lower rank in our own conference ranking. Also, if we had more data (e.g. more papers and references of the respective conferences) we expect even better results.

Further the relatively high values for γ that produce good results, suggest that CORE and Microsoft put a lot of weight on how "good" the authors publishing in a conference are.

Nonetheless our evaluation enables us to choose a best parameter configuration. For $\alpha = 0.1, \beta = 0.2, \gamma = 0.5$ and Mode 1, Table 3.2 lists the ten papers with the highest PaperRank. All ten papers seem to be important in their field as they all have a high citation count and many of them are written by well-known authors.

Title	PaperRank	Citedbys according to Google Scholar [Goo]
A method for obtaining digital signatures and public-key cryptosystems	8.622E-05	14275
Bagging predictors	6.422E-05	11221
A relational model of data for large shared data banks	6.388E-05	8650
Learning internal representations by error propagation	6.249E-05	16666
The anatomy of a large-scale hypertextual Web search engine	6.173E-05	11864
Distinctive Image Features from Scale-Invariant Keypoints	5.775E-05	24476
Support-Vector Networks	5.772E-05	13922
Time, clocks, and the ordering of events in a distributed system	5.499E-05	8557
Fast Algorithms for Mining Association Rules in Large Databases	5.370E-05	16300
Induction of Decision Trees	5.233E-05	13311

Table 3.2: This table shows the ten papers with the highest PaperRank calculated with the parameters $\alpha = 0.1$, $\beta = 0.2$, $\gamma = 0.5$ and Mode 1 and their number of cited-bys according to GoogleScholar [Goo].

Chapter 4

Finding Relevant Papers

So far, we have only been concerned with the absolute importance of a paper: We know for instance which of all the papers in the database is the most important. In order to find *related* papers, however, the absolute importance is not the only information we need. After all, the most important paper is not necessarily the most relevant one for every single paper.

In this section, we introduce our search algorithm, whose goal is to find papers that are related to some papers specified by the user of our program. The challenge lies in defining *related* in this context.

We start with a brief informal description of how we choose related papers (Section 4.1). We then introduce formally our search algorithm (Section 4.2), and describe its implementation (Section 4.3). Finally, the search results are compared to those of other engines (Section 4.4).

4.1 Overview

We use the directed graph $G_P = (P, E)$ of papers, where $E = E_r \cup E_c \cup E_a$, from Section 3.4.1. For a paper $p \in P$, we consider the neighbors of p as being related to p. After all, this set contains all the papers that p cites, all the papers that cite p, and all the papers written by the authors that wrote p, apart from p. Intuitively, these papers can be considered related to p because: references exist to cite previous or related work in the area. Also, future papers on a similar topic might cite the paper. And authors are likely to write more than one paper in one area of research, meaning that some of their papers should cover similar topics.

However, only looking at the immediate neighborhood is not sufficient. Not all related papers are directly cited by the input paper or are written by the same author. Therefore, relevant papers appear not only in the *immediate* neighborhood of all input papers, but also in its extended neighborhood.

What exactly this means is introduced formally in the next section.

4.2 Formal Description

Definition 4.1. Let $p \in P$ be a paper. We define:

$$R(p) := \{q \in P : (p,q)_r \in E_r\} \qquad \text{(references of the paper } p\text{)}$$
$$C(p) := \{q \in P : (q,p)_c \in E_c\} \qquad \text{(papers citing } p\text{)}$$
$$A(p) := \{q \in P \setminus \{p\} : (p,q)_a \in E_a\} \qquad \text{(papers sharing an author with } p\text{)}$$

We introduce the notion of a weighted set of papers as follows:

Definition 4.2. Let $P_\star \subseteq P$ and $\mathbf{w}$ be a mapping

$$\mathbf{w} : P \to \mathbb{R}_+ \text{ such that } w(p) = 0 \;\forall\; p \notin P_\star. \tag{4.1}$$

The pair $(P_\star, \mathbf{w})$ is called a *Paper Weight Tuple* and $w(\cdot)$ is called its *weight function*. For some paper $p \in P_\star$, $w(p)$ is called the *weight* of p[1].

We refer to $w(\cdot)$ as the weight function, but initially it can also be thought of as a rank function: the initial weights of papers will be chosen to be their PaperRanks.

To be able to set the initial weights to PaperRanks, we introduce the $\mathbf{rank}_{P_\star}$ function. It differs from the default PaperRank function in that it adheres to Equation 4.1. It can therefore be used as the weight function of some Paper Weight Tuple.

Definition 4.3. Using the $rank(\cdot)$ function from Equation 3.1 that assigns to *all* papers $p \in P$ a PaperRank, we define a function that maps to a PaperRank only for papers in some $P_\star \subseteq P$:

$$\mathbf{rank}_{P_\star} : P \to \mathbb{R}_+$$
$$p \mapsto \begin{cases} rank(p) & \text{if } p \in P_\star \\ 0 & \text{else} \end{cases}$$

We introduce $\mathcal{T}$ to refer to the space of all Paper Weight Tuples and use it to define some useful operations on Paper Weight Tuples:

Definition 4.4. Let $n \in \mathbb{N}$. The $\mathbf{best}_n$ function on some $T \in \mathcal{T}$ is the function that returns the Paper Weight Tuple with the n papers from T that have the n highest weights.

[1] For functions, we use the same notation as [Lap09]: If a function has a name, the name is written in bold as in $\mathbf{f}$. Alternatively, we sometimes denote a function $\mathbf{f}$ as $f(\cdot)$. We write $f(t)$ for the result of applying the function $\mathbf{f}$ to t.

Definition 4.5 (Scalar Multiplication on $\mathcal{T}$)**.**

$$\begin{aligned} \cdot : \mathbb{R}_+ \times \mathcal{T} &\to \mathcal{T} \\ (\alpha, (P_\star, \mathbf{w})) &\mapsto (P_\star, \alpha \cdot \mathbf{w}) \end{aligned}$$

We define union and intersection of two Paper Weight Tuples as one would expect intuitively: The sets of papers get united or intersected and the weight functions get added together.

Definition 4.6 (Union or $\cup$ on $\mathcal{T}$)**.**

$$\begin{aligned} \cup : \mathcal{T} \times \mathcal{T} &\to \mathcal{T} \\ ((P_1, \mathbf{w}_1), (P_2, \mathbf{w}_2)) &\mapsto (P_1 \cup P_2, \mathbf{w}_1 + \mathbf{w}_2) \end{aligned}$$

Definition 4.7 (Intersection or $\cap$ on $\mathcal{T}$)**.**

$$\begin{aligned} \cap : \mathcal{T} \times \mathcal{T} &\to \mathcal{T} \\ \big((P_1, \mathbf{w}_1), (P_2, \mathbf{w}_2)\big) &\mapsto \big(P_1 \cap P_2, \mathbf{w} : p \mapsto I\{p \in P_1 \cap P_2\} \cdot (w_1(p) + w_2(p))\big) \end{aligned}$$

(Note the use of the indicator $I(\cdot)$ from Section 3.4.3 to make sure that $\mathbf{w}$ adheres to Equation 4.1.)

Definition 4.8. For some $T = (P_\star, \mathbf{w}) \in \mathcal{T}$, we denote with $|T|$ the number of elements in $P_\star$.

4.2.1 Search Algorithm

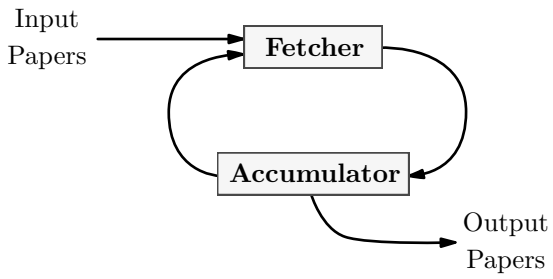

Figure 4.1: Schematic of the Search Algorithm

We now describe the actual search algorithm. Figure 4.1 gives a schematic overview. We start with two important components of our algorithm: the *Fetcher* and the *Accumulator*.

The *Fetcher* is an operation that returns *relevant* papers for the papers in a given input Paper Weight Tuple. It formalizes the intuition introduced in Section 4.1 that the outgoing neighbors of a paper are relevant to it, but that not all of them are equally relevant: We introduce weight parameters $w_R, w_C, w_A \in$

$[0, 1]$ to reduce the ranks of references, citations and/or papers by the same authors.

The parameters $b_R, b_C, b_A \in \mathbb{N}_0$ are used to specify how many references, citations or papers by the same authors are to be returned, b stands for *best*. This allows us for instance to return more references than papers by the same authors.

More precisely, the Fetcher is a function $fetcher(\cdot)$ with parameters $w_R, w_C, w_A \in [0, 1]$ and $b_R, b_C, b_A \in \mathbb{N}_0$:

$$\begin{aligned}
&\textbf{fetcher} : \mathcal{T} \to \mathcal{T} \\
&\quad (P_\star, w) \mapsto \bigcup_{p \in P_\star} \Big[best_{b_R}\big((R(p), w_R \cdot \mathbf{rank}_{R(p)})\big) \cup \\
&\qquad\qquad best_{b_C}\big((C(p), w_C \cdot \mathbf{rank}_{C(p)})\big) \cup \\
&\qquad\qquad best_{b_A}\big((A(p), w_A \cdot \mathbf{rank}_{A(p)})\big) \Big]
\end{aligned}$$

The *Accumulator* first reduces the weights of the output of the Fetcher by some parameter ρ. It then merges the output with the previous output: it accumulates output. The parameter b ensures that only the best b papers are in the output.

More precisely, the *Accumulator* is a function $accumulator(\cdot)$ with parameters $\rho \in [0, 1]$ and $b \in \mathbb{N}_0$.

$$\begin{aligned}
&\textbf{accumulator} : \mathbb{N}_0 \times \mathcal{T} \times \mathcal{T} \to \mathcal{T} \\
&\qquad (s, F^{(s)}, A^{(s-1)}) \mapsto best_b\left((\rho^s \cdot F^{(s)}) \cup A^{(s-1)}\right)
\end{aligned}$$

We can now introduce the search algorithm, Algorithm 2. It takes as input a set of *input papers* $P_{in} \in P$ and outputs a Paper Weight Tuple $T_{out} \in \mathcal{T}$. In addition to the parameters of the Fetcher and the Accumulator, it uses the following parameters:

- $maxs \in \mathbb{N}_0$: The maximum number of steps before the algorithm stops.
- $minchange \in \mathbb{N}_0$: The minimum number of papers that need to change in each step for the algorithm to keep running.

Algorithm 2 Search Algorithm

$F^{(1)} \leftarrow (P_i, \mathbf{rank}_{P_i})$
$A^{(0)} \leftarrow (\varnothing, \mathbf{rank}_{\varnothing})$
$s \leftarrow 1$
while $s < maxs$ **do**
  $F^{(s)} \leftarrow \text{fetcher}(F^{(s)})$
  $A^{(s)} \leftarrow \text{accumulator}(s, F^{(s)}, A^{(s-1)})$
  if $|A^{(s)} \cap A^{(s-1)}| \leq minchange$ **then**
    break
  $F^{(s+1)} \leftarrow A^{(s)}$
  $s \leftarrow s + 1$
return $A^{(s)}$

Reduction ρ and Termination

The Accumulator reduces the weights of the result of each step by multiplying ρ^s and only keeps the best b papers. This means that after a certain number of steps, the weights of the newly fetched papers will be too low to make it into the output, and therefore the condition $|A^{(s)} \cap A^{(s-1)}| \leq minchange$ will be met, causing the algorithm to terminate.

Reducing the weights makes sense because in each step, the distance from the input papers grows, and therefore one probably fetches papers that are less and less relevant. After all, with $\rho = 1$ and $maxs \to \infty$, one would accumulate almost[2] all the papers in the database.

Merging

An important aspect of our algorithm is how papers from different sources are merged. We defined intersection and concatenation such that merging two tuples $T_1, T_2 \in \mathcal{T}$ adds up the weights of all the papers that are in both. This has consequences for the Fetcher and the Accumulator:

For the Fetcher, this means that if a paper p_A is written by one of the authors of a paper p_B and if p_A also cites p_B, the Fetcher yields p_A once, but with a rank – here it makes more sense to refer to the weight as rank – equal to $rank(p_A) \cdot (w_R + w_A)$, i.e. possibly larger than just $rank(p_A)$. This makes sense because p_A sharing an author *and* a citation with p_B is a stronger indication that p_A is from the same field as p_B than if they just shared an author for instance.

For the Accumulator, this means that if a paper p_A was previously fetched and is now fetched again, its rank is increased. This is justified in that this means that p_A has been reached by multiple paths in the graph.

[2] G_P is not necessarily connected.

4.3 Implementation with a GUI

4.3.1 GUI

We implemented the search algorithm with a graphical user interface, or GUI for short. For this, we used *Python*[3] and the *Tkinter* framework[4]. Appendix A.1 shows a few screenshots of the GUI.

4.3.2 Local Paper Titles Search Index

In order to provide the search algorithm with input papers, we use a local database of all paper titles and a search index on that database. This allows users of the GUI to quickly find a paper by title. The search polls results as the user types, so that partial search queries are sufficient to find most things. We use the *whoosh* framework[5] for this

4.3.3 Search Algorithm

We implemented the search algorithm as presented in the previous section. The main challenge was to execute the searching on a background process so that the GUI would not become unresponsive. However, Python allows to solve this elegantly with the *multiprocessing* package[6].

As for the parameters, we used: $w_R = 1, w_C = 1, w_A = 1, b_R = 45, b_C = 25, b_A = 25, b = 95, \rho = 0.25, maxs = 10, minchange = 2$. This was determined by trial and error.

4.4 Evaluation and Comparison

To evaluate the results of our search algorithm, we compare them with Google Scholar and Mendeley, as introduced in Section 2.1.2.

4.4.1 Google Scholar

Google Scholar has a *Related Articles* functionality that is similar to our search. It can be accessed by first searching for a paper by title and then selecting the Related Articles link. In contrast to our search engine however, only one paper can be specified as input.

[3] https://www.python.org
[4] https://wiki.python.org/moin/TkInter
[5] https://pypi.python.org/pypi/Whoosh
[6] https://docs.python.org/2/library/multiprocessing.html

We first select 9 different papers from our database at random. We then run our search algorithm and the Google Scholar Related Articles functionality on each of them. From the Google Scholar Related Articles, we remove all the papers that we do not have in our database, because our result is never going to contain them. Then we compare how many of the first 10 and of the first 20 papers of our search result are in Google's results[7]. The result of this comparison can be seen in Table 4.1.

	References	*Google Scholar Related Articles*		
Paper	in our database	in our database	in our Top 20	in our Top 10
A	96	3	2 *66%*	-
B	24	7	4 *57%*	3 *42%*
C	15	1	1 *100%*	-
D	9	5	5 *100%*	5 *100%*
E	6	2	1 *50%*	-
F	3	3	2 *66%*	2 *66%*
G	1	1	-	-
H	0	7	-	-
I	0	2	-	-

Table 4.1: Comparison of the results of our search algorithm with Google Scholar. The second column shows how many of the references of a paper are stored in our database - not necessarily *all* the references of the paper. The next three columns show how many papers of the result of the Google Scholar Related Articles search are in our database, in the Top 20 and in the Top 10 of our search.

The results allow for the following observations:

- If there are a lot of citations stored in our database, and if a lot of the Google results are in our database as well, the results match to a certain degree. For instance for paper D, our Top 10 contains all the papers from Google Scholar.
- If there are no citations in our database, our search performs poorly in comparison to Google Scholar. Citations seem to be quite important for finding papers that Google Scholar also considers relevant.

A problem of the comparison is that Google finds a lot of papers that are not in our database, and therefore comparison is not very meaningful. Also, for a lot of papers, our database does not store sufficient metadata for a good search.

[7]We chose 10 because Google shows 10 Related Articles and 20 because the first 20 results of our search seem to be most relevant.

4.4.2 Mendeley

In comparing our results to Mendeley, we came to the conclusion that their Related Articles function uses title comparison and is therefore not comparable to our algorithm. For instance, when looking for Related Articles for "The PageRank *citation ranking*: *Bringing order* to the web." we get these results:

- "InstanceRank: *Bringing order* to datasets"
- "Citation counting, *citation ranking*, and h-index of human-computer interaction researchers: A comparison of scopus and web of science"
- "Object-Level Ranking : *Bringing Order* to Web Objects"
- "*Bringing order* to the web: Automatically categorizing search results"
- "The PageRank citation ranking:bringing order to the web." *(Same as input paper but without a space after the colon.)*
- "TextRank: *Bringing order* into texts"
- "PostingRank: *Bringing order* to web forum postings"

Chapter 5

Conclusion and Outlook

In this paper we have shown how PageRank can be adapted to rank scientific papers, resulting in PaperRank. We have implemented a search algorithm that provides a means of finding relevant papers to a set of input papers. While we have shown that PaperRank ranks papers in a meaningful way and that our search algorithm produces useful results, there are ways to improve both the rank and the algorithm. We present some ideas here.

Apart from citations and information about authors, information about conferences could also be used in ranking the papers: papers that were presented on the same conference are likely to be related. Moreover, a more thorough evaluation of the PaperRank would be desirable. For instance, similar to how we used conference rankings as a measure of quality, an author ranking (for example based on the h-index) could be used to further assess the PaperRank.

The search process could be facilitated by including the title search index in the database and have everything online. As with the PaperRank itself, the results of the search algorithm could be evaluated in a more sophisticated manner. Firstly, better papers from our database could be used, i.e. papers with a lot of neighbors. However, it is not trivial to find such papers. Secondly, more than just the first page of results of Google Scholar could be used. If this was to be done with a lot of papers, a scientific evaluation of the results would be feasible. With a better comparison, the parameters of the algorithm could be fine-tuned in a more scientific fashion. To prevent the results from becoming a replica of other search engines' results, one could try to identify more elaborate ways of finding good values for the tuning parameters.

Another area for improvement is the database itself. While the ACM database already provides a lot of information, it still has some flaws. For example, there are references missing and some authors that appear multiple times in the database, with different spellings of their name.

Bibliography

[Abd07] Abdi, Hervé: *Chapter: The Kendall Rank Correlation Coefficient 1 Overview, in Encyclopedia of Measurement and Statistics (by Neil J. Salkind)*. Sage, 2007. – 508–510 p.

[BG92] Bertsekas, Dimitri ; Gallager, Robert: *Data Networks (2nd Ed.) Appendix A Review of Markov Chain Theory*. Prentice-Hall, Inc., 1992. – 259–261 p.

[COR] CORE: *"Core Ranking", [online]*. core.edu.au/index.php/categories/conference%20rankings/1. – last access 28.05.2014

[Goo] Google: *"Google Scholar", [online]*. www.scholar.google.com. – last access 12.06.2014

[Hav02] Haveliwala, Taher H.: Topic-sensitive PageRank. In: *Proceedings of the 11th International Conference on World Wide Web*, ACM, 2002 (WWW '02), p. 517–526

[Lap09] Lapidoth, Amos: *A Foundation in Digital Communication, Chapter: Some Essential Notation*. Cambridge University Press, 2009. – 1–3 p.

[Men] Mendeley: *"Mendeley Reference Manager", [online]*. www.mendeley.com/research-papers. – last access 28.05.2014

[Mic] Microsoft: *"Microsoft Ranking", [online]*. academic.research.microsoft.com/RankList?entitytype=3&topdomainid=2&subdomainid=0&orderby=6. – last access 28.05.2014

[PBMW99] Page, Lawrence ; Brin, Sergey ; Motwani, Rajeev ; Winograd, Terry: The PageRank Citation Ranking: Bringing Order to the Web. Stanford InfoLab, November 1999 (1999-66). – Technical Report

[SS01] Schickinger, Thomas ; Steger, Angelika: *Diskrete Strukturen (Band 2) Chapter: Prozesse mit diskreter Zeit*. Springer-Verlag, 2001. – 169–185 p.

Appendix A

Appendix Chapter

A.1 Screenshots of the GUI

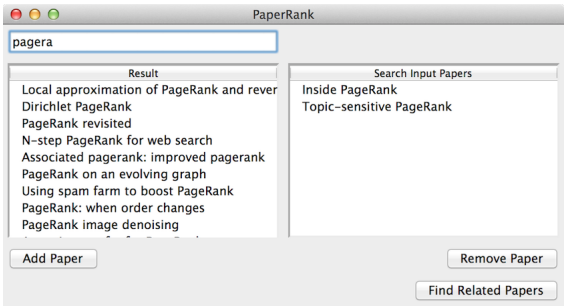

Figure A.1: Screenshot of the Search Window

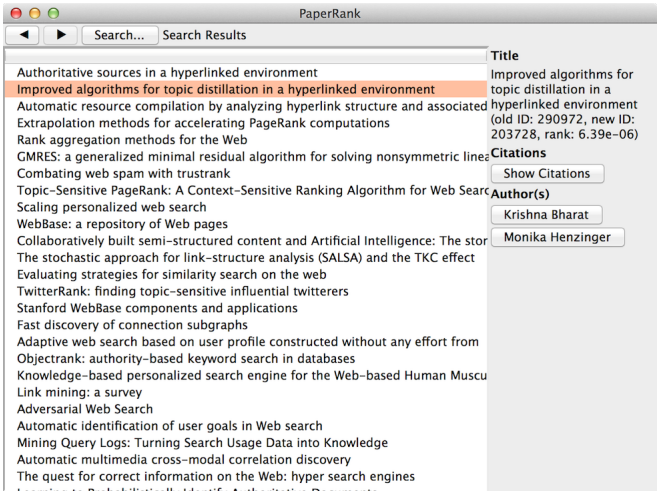

Figure A.2: Screenshot of a List of Papers

A.2 Additional Data

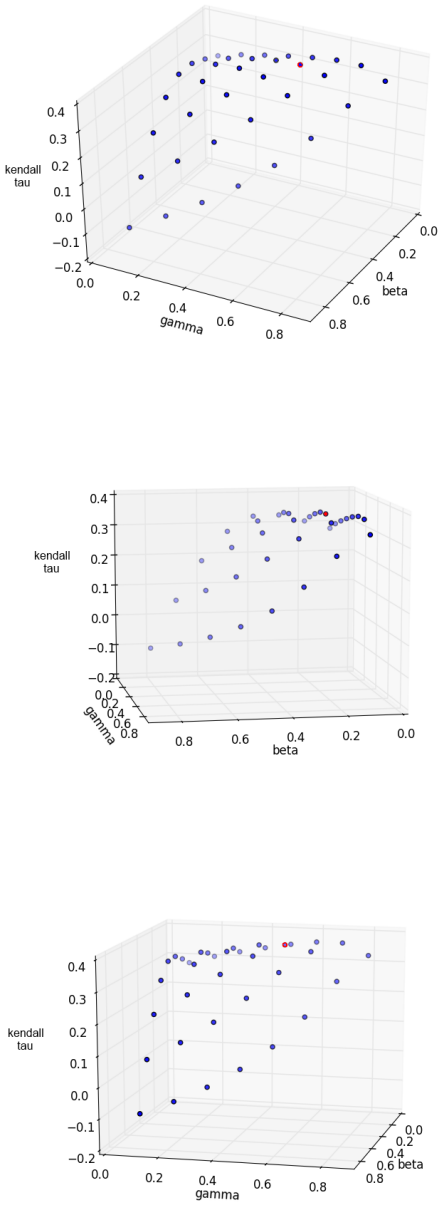

Figure A.3: This figure shows the Kendall rank correlation coefficient between our conference rank (see Section 3.6.1) and the CORE ranking [COR] from three different angles, plotted as a function of β and γ, using Mode 1 and $\alpha = 0.1$.

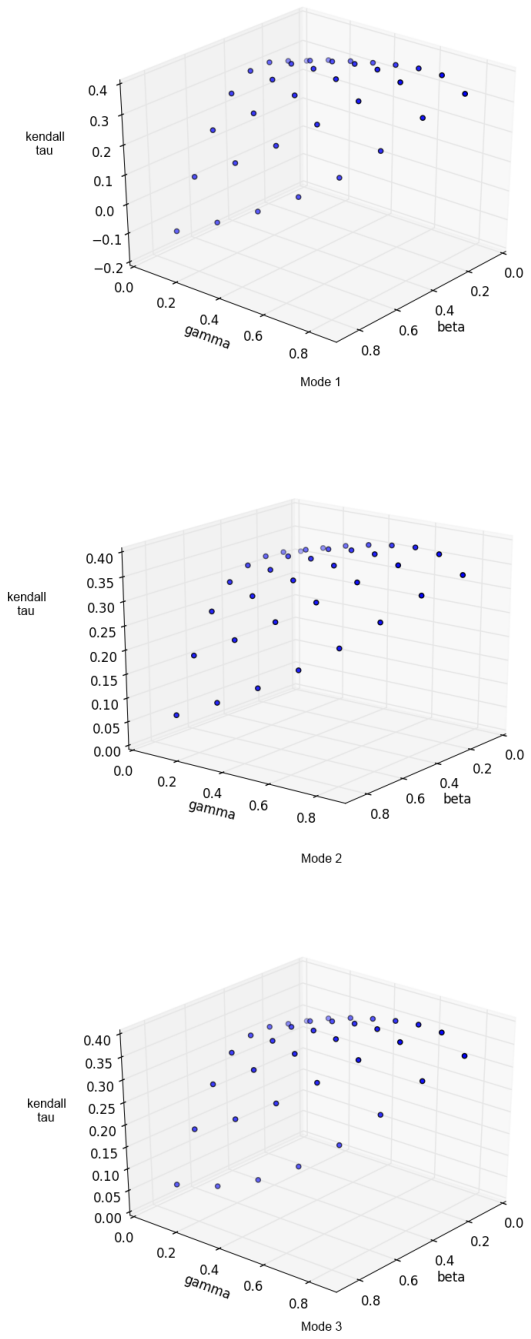

Figure A.4: This figure shows the Kendall rank correlation coefficient between our conference rank (see Section 3.6.1) and the CORE ranking [COR], plotted as a function of β and γ, for all three modes using $\alpha = 0.1$.

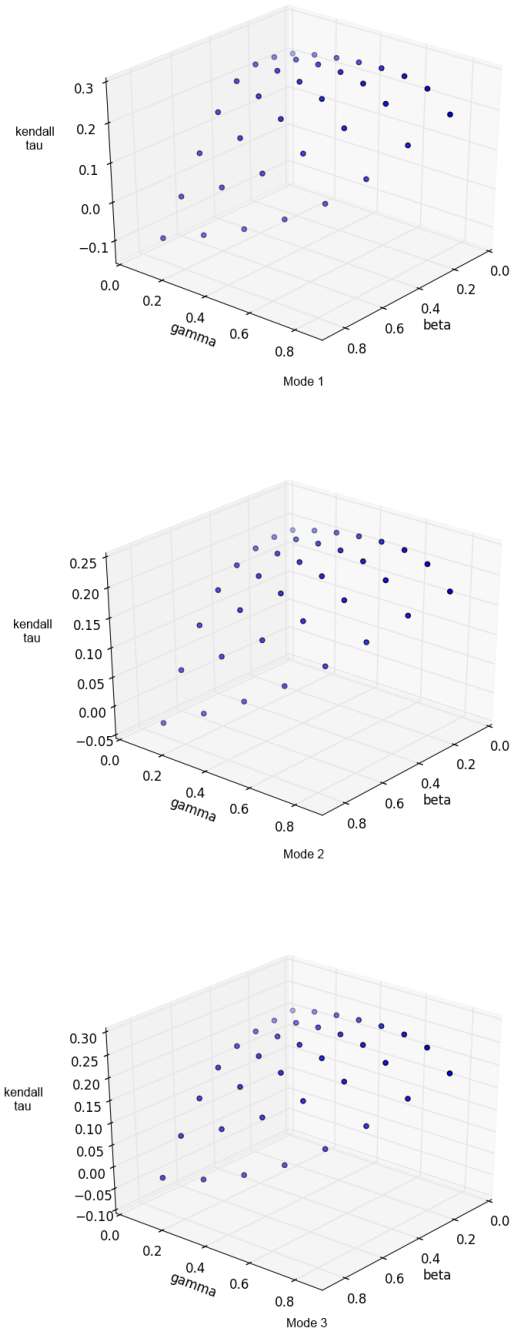

Figure A.5: This figure shows the Kendall rank correlation coefficient between our conference rank (see Section 3.6.1) and the Microsoft ranking [Mic], plotted as a function of β and γ, for all three modes using $\alpha = 0.1$.

Modes	Kendall's $\bar{\tau}$	$\sigma_{\text{Kendall's } \bar{\tau}}$
Mode 1 vs Mode 2	0.4313	0.1452
Mode 1 vs Mode 3	0.5404	0.1352
Mode 2 vs Mode 3	0.7060	0.0671

Table A.1: The plot shows the average Kendall rank correlation coefficient and its standard deviation respectively computed for two PaperRanks that only differ in the Mode used. The two respective PaperRanks have been calculated for $\alpha = 0.1, 0.2 \ldots, 0.8$, $\beta = 0.1, 0.2 \ldots, 0.8$ and $\gamma = 0.1, 0.2, \ldots, 0.8$.