

"Politehnica" University of Bucharest  
Faculty of Automatic Control and Computer Science

# **Personalized Search in P2P Networks**

*Diploma Thesis in Computer Science*

by *Oana Scurtu*

Supervisors:

Prof. Dr. Techn. *Wolfgang Nejd*  
Prof. Dr. Ing. *Valentin Cristea*  
Ing. *Paul - Alexandru Chirita*

September 2004

# Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Introduction</b>  | <b>2</b>  |
| 1.1      | L3S and Uni Hannover . . . . .                                       | 2         |
| 1.2      | Information Retrieval . . . . .                                      | 3         |
| <b>2</b> | <b>Generating the Input Data</b>                                     | <b>5</b>  |
| 2.1      | Web Simulator - Theoretical Approach . . . . .                       | 5         |
| 2.1.1    | Simulator - Detailed View . . . . .                                  | 10        |
| 2.1.2    | Experiments and Results . . . . .                                    | 12        |
| 2.2      | Data Distribution . . . . .  | 14        |
| <b>3</b> | <b>Distributed Personalized Page Rank</b>                            | <b>15</b> |
| 3.1      | Page Rank and Personalized Page Rank . . . . .                       | 15        |
| 3.1.1    | Background and Previous Work . . . . .                               | 16        |
| 3.1.2    | Widom Algorithm - Centralized . . . . .                              | 24        |
| 3.1.3    | Widom Algorithm - Distributed . . . . .                              | 25        |
| 3.1.4    | Experiments and Results . . . . .                                    | 28        |
| 3.2      | Application: Search Algorithm using Personalized Page Rank . . . . . | 31        |
| 3.2.1    | Background and Previous Work . . . . .                               | 31        |
| 3.2.2    | Our Algorithm - Detailed View . . . . .                              | 32        |
| 3.3      | Experiments and Results . . . . .                                    | 32        |
| <b>4</b> | <b>Trust and Personalized Trust in P2P Networks</b>                  | <b>39</b> |
| 4.1      | Trust Computation . . . . .  | 40        |
| 4.1.1    | Definition of Trust . . . . .  | 41        |
| 4.1.2    | How to Quantify Trust . . . . .                                      | 44        |
| 4.1.3    | How to Propagate Trust Values . . . . .                              | 46        |
| 4.1.4    | Previous Reputation Systems . . . . .                                | 49        |
| 4.1.5    | Our Algorithm . . . . .  | 50        |
| 4.1.6    | Distributed Widom Algorithm for Trust . . . . .                      | 52        |
| 4.2      | Application: Search Algorithm Using Trust Values . . . . .           | 57        |
| 4.2.1    | Background and Previous Work . . . . .                               | 57        |
| 4.2.2    | Our Algorithm - Detailed View . . . . .                              | 59        |
| 4.2.3    | Malicious Threats . . . . .  | 61        |
| 4.3      | Experiments and Results . . . . .                                    | 62        |
| <b>5</b> | <b>Conclusions and Further Work</b>                                  | <b>66</b> |

# **1 Introduction**

## **1.1 L3S and Uni Hannover**

Founded in 1831 as a Higher Trade School of Hannover and moved into the Guelph palace in 1879 as the Royal College of Technology (Koenigliche Technische Hochschule), the university of Hannover is the largest higher education institution in Lower Saxony. It has around 24.000 students, 2.000 academics and scientists working in 17 faculties with around 160 departments and institutes. The diversity of academic and scientific disciplines (science and engineering, economics, arts, humanities and social sciences) combines with high-quality research and excellent study conditions to make Hannover University an attractive place to study. Within the region, the university is a significant economic and innovative factor, while in the international field the university is seen as a highly proficient research and teaching institution.

The Learning Lab Lower Saxony (L3S) Research Center was created in 2001 as a coordinating institute for the University of Hanover, Technical University Carolo-Wilhelmina at Brunswick, and the Brunswick School of Art. Professors from the universities of Karlsruhe, Mannheim and Kassel are also actively involved in the L3S Research Center. L3S has focused on theoretical and applied research in the innovative areas of information, learning, and knowledge technologies and also on training and continuing education concepts for academia and industry.

In the short period of less than three years, the L3S Research Center has established itself as a nationally and internationally acknowledged center of research(as coordinator of the Network of Excellence PROLEARN, as part of the 6th EU research program; as core partner in the KnowledgeWeb and REWERSE networks working in the field of Semantic Web; as partner in the EURON (Robotics) network and in Wallenberg Global Learning Network (WGLN) and is active in the organization of the most important conferences in these fields).

## 1.2 Information Retrieval

Peer to peer networking is one of the most rapidly developing areas of modern computing. By utilising the exponentially increasing number of Internet nodes, which can be anything from powerful servers to mobile devices, the P2P paradigm attempts to create open and collaborative networks of the most diverse functionality nature.

Generally, a peer-to-peer (or P2P) computer network is any network that does not have fixed clients and servers, but a number of peer nodes that function as both clients and servers to the other nodes on the network. Any node is able to initiate or complete any supported transaction. Peer nodes may differ in local configuration, processing speed, network bandwidth, and storage quantity.[50] The peers are sharing/looking for different resources, like information (files) or services. In order to receive a file/service, a peer queries its neighbours and they query their neighbour and so on until a peer can provide the file/service and sends back a positive answer.

The most distinct characteristic of P2P computing are that there is no central point of control and that there is a symmetric communication between the peers: each peer has both a client and a server role (there are some networks with super peers). The advantages of the P2P systems are multi-dimensional: they are scalable by enabling direct and real-time sharing of services and information, enable knowledge sharing by aggregating information and resources from nodes that are located on geographically distributed and potentially heterogeneous platforms and provide high availability and robustness by eliminating the need for a single centralized component.

In the context of increasing the size of these networks, *knowing where to search* [42] becomes a real problem. Optimizing and focusing search is often solutionate by computing and then comparing *page rank* values for the available files. More accurate results can be obtained with *personalization* of this results. We have developed an algorithm to distributedly compute personalized page rank values (using the advantages of their distributed structure - we will avoid keeping in memory unnecessary data) and also an algorithm which uses this values to optimize and focus search in the P2P network. The paper also discusses how these algorithms improve current *distributed search* in *power law* networks and gives some simulation results.

Because of the open nature of the P2P networks, it is possible that *malicious users* introduce corrupt data and/or harmful services much easier than in centralized systems. Many kind of reputation schemes have been developed in order to reduce "malicious" activities. Such reputation algorithms are usually based on aggregating the trustworthiness information collected by each peer (based on interaction with other peers) thus generating a micro or macro *web of trust*.

As the size of current P2P networks is continuously increasing, recent research has concentrated on designing more *personalized trust management algorithms*, while also addressing their robustness against attacks of malicious peers.

In this paper we introduce a novel algorithm for computing *personalized reputation values* in P2P networks and test its robustness against several possible attacks in a simulated P2P environment. In our experiments, we exploit the power law distribution of peers observable in many current P2P networks.

We also developed a *Web Simulator*, to create test data for our programs. The papers we studied used existing P2P Networks (Gnutella [21]) or generated Internet like graphs. A very important search optimization has been made by considering the power laws that govern this networks: when performing a search, peers were selected with a probability proportional to their rank (which is in a power law distribution). So, highly ranked peers were selected as download source much more often than the other ones, thus guaranteeing obtaining a good answer in a few steps. Of course, overloading some very trusted peers in the Network has been avoided.

We start with a short introduction of the information retrieval problem in a distributed P2P environment.

The contributions of this papers include:

- an algorithm for computing (*personalized*) *Page Rank* in a *distributed* fashion 3.1
- new approach to search in P2P networks using Page Rank values 3.2
- an algorithm for computing (*personalized*) *Trust* values in a *distributed* environment 4.1
- a *reputation* system which incorporates both user-individual personalization and global experiences of peers in the network 4.2
- experimental results of running these algorithms in a simulated power law-based P2P environment 3.3,4.3

Section 5 concludes with a discussion of future work.

## 2 Generating the Input Data

If we consider P2P networks and Internet represented as directed graphs with nodes being web pages or collections of web pages and edges - hyperlinks between them, the two graphs have the same structure (P2P are subsets of Internet). We used for our tests graphs having Internet characteristics. We mean by that both the graph structure (gouverned by several power laws) and the data distribution. We will show below the assumption that we have made.

### 2.1 Web Simulator - Theoretical Approach

The measurments regarding the Internet [37, 8, 17] structure revealed that the network is not random, but gouverned by several *power laws*. This information is crucial in order to construct more effective search engines. We are also interested in the observed values of some parameters of this graph like *averadge edges for a node*, *diameter* of the *strongly connected component*.

If we consider the directed graph whose nodes correspond to static pages on the web, and whose arcs correspond to hyperlinks between these pages, we will use the terms *In* degree, *Out* degree for the number of *incoming* and *outgoing* arcs. Here are two of the power laws important by the point of view of a search algorithm:

**Rank exponent** The *Out* and *In* degree  $d_v$  of a node  $v$  is proportional to the rank of the node (if we sort all nodes by their degree),  $r_v$  to the power of a constant  $\gamma$  :

$$d_v = C \cdot r_v^\gamma$$

where C is a real constant.

The Internet graph has for the *In* neighbours the  $\gamma = -2,1$  and for the *Out*  $-2,7$ .

If we impose the condition that the last node (in the sorted set) has degree 1 (the last degree - actually most of the pages are having degree 1), we can compute a formula for the degree:

$$d_v = C \cdot r_v^\gamma \Rightarrow 1 = d_N = C \cdot r_N^\gamma \Rightarrow$$

$$C = \frac{1}{N^\gamma} \Rightarrow d_v = \frac{1}{N^\gamma} \cdot r_v^\gamma$$

**Degree frequency** The frequency  $f_d$  of a degree  $d$  is proportional to the degree of the node to the power of a constant  $\gamma$  :

$$f_d = C \cdot d^\gamma$$

where  $C$  is a real constant.

In the real Internet, the correlation coefficient of the fit is around 96 proc (according to Faloutsos [37]).

In order to obtain a  $N$ -nodes Internet-like graph, we applied the rank exponent formula, but the results were quite far from what we expected. So we had to do some adjustments, we had to study some other properties of the Internet graph and to combine them.

For better understanding, we will compare the results obtained by applying only one or two properties with the final results.

For a 1000 nodes graphs we obtain the following degree distribution (for  $\alpha = 2800$  and  $\gamma = -2.1$ ) (see table 1):

- 1 node with an indegree of 2800 (impossible as long as there is only maximum one inlink from a node to another one),
- 2 nodes with degree 635
- 3 nodes with degree 278
- 334 nodes with only 1 neighbour.

Obs: the constant  $\alpha$  was chosen to be 2800 because it provided the best fit of edges per nodes.

In fig. 1 we can see the power law structure of the degree distribution. A different  $\alpha$  would produce a slight movement of the line. The number of nodes with the same degree is also in a power law distribution: we chose to represent it under a pie format ( fig. 2), in order to observe the great amount of nodes with degree one.

We can have different graphs with the same power law distributions of nodes degrees, but with different average edges number, different connectivity structure.

| <b>Degree</b> | <b>Nodes Nr.</b> |
|---------------|------------------|
| 2800          | 1                |
| 653           | 2                |
| 279           | 3                |
| 152           | 4                |
| ...           | ...              |
| 4             | 69               |
| 3             | 106              |
| 2             | 260              |
| 1             | 334              |

Table 1: Theoretical degree distribution (1000 nodes graph)

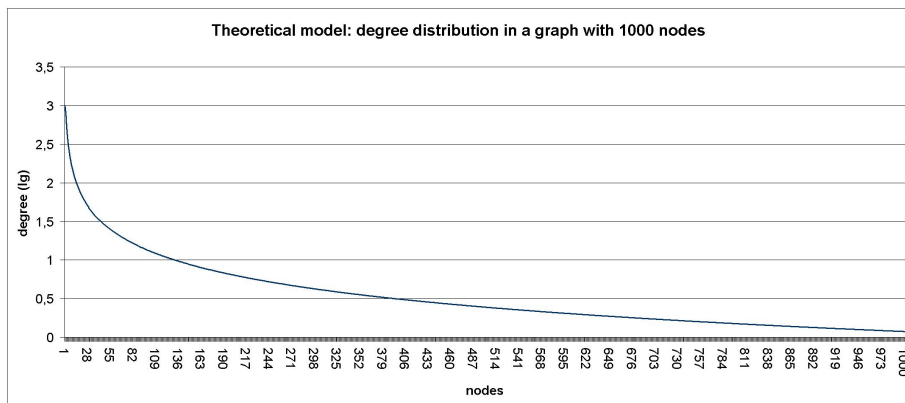


Figure 1: Theoretical degree distribution

We will present in the following the graph properties we considered when developing our web simulator.

Given a directed graph, a *strongly connected component* (SSC for brevity) of this graph is a set of nodes such that for any pair of nodes  $u$  and  $v$  in the set there is a path from  $u$  to  $v$ . In general, a directed graph may have one or many SSCs. The SSCs of a graph consist of disjoint sets of nodes.

A very dynamic network like Internet is expected to have an always changing structure, but the studies found some interesting properties that remain the same over time.

Internet is made of a huge *connected component* - if we consider the undirected graph. As a directed graph, there are four major components with about the

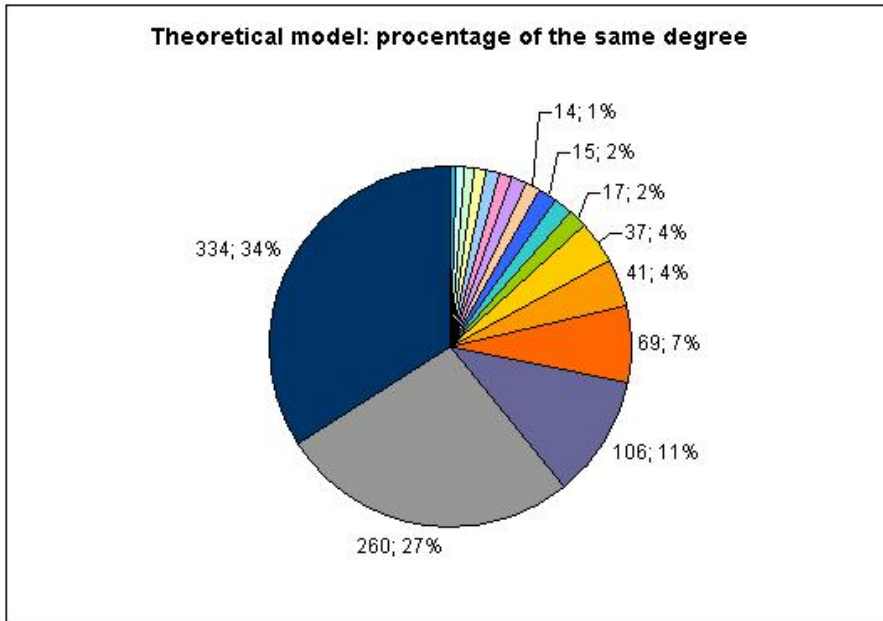


Figure 2: Theoretical degree frequency (percentage)

same size (number of nodes) [12]:

**SSC** a strongly connected component

**IN** a set that has the following property: there is a directed path from each node of IN to (all the nodes of) SCC,

**OUT** containing all starting points for which there is a directed path from any node in the SCC to any node in OUT and

**TENDRILS** containing nodes that are reachable from portions of IN, or that can reach portions of OUT, without passage through SCC (may also contain tubes).

*Connectivity of the web* [3]: one can pass from any node of IN through SCC to any node of OUT. Hanging off IN and OUT are TENDRILS containing nodes that are reachable from portions of IN, or that can reach portions of OUT, without passage through SCC. It is possible for a TENDRIL hanging off from IN to be hooked into a TENDRIL leading into OUT, forming a TUBE – a passage from a portion of IN to a portion of OUT without touching SCC.

Most of the studies on search algorithms are done only on SSC, so we consider that there is always a path between any randomly chosen peers. We will have to

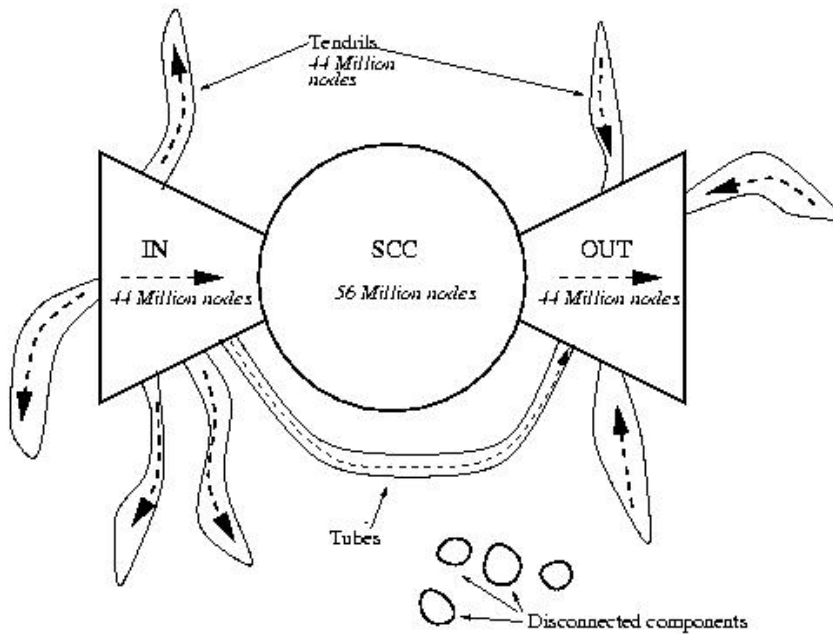


Figure 3: Macroscopic structure

impose this condition when creating the graph (or select from the generated graph the strong connected component).

The *diameter* of the graph is the maximum over all ordered pairs  $(u, v)$  of the shortest path from  $u$  to  $v$ .

IN 2001, the diameter was around 16 in 2001 (without counting the infinite values). [12].

The macroscopic structure, the high connectivity and the very distance between any two nodes (we are talking about a graph with 200 million nodes) are strong reasons for a search algorithm to finish very fast. One may need only a few steps to find the answer). [4]

The *average edges* for a node is in the real Internet around 8 [12]. In a power law network - like the one we study, that means that most of the nodes have 1 or 2 neighbours!

### 2.1.1 Simulator - Detailed View

We implemented a Simulator that generates a directed graph with N nodes having a power law distribution with different in and out degree.

We have also developed a program to compute  $\alpha$  for a given N, so that we obtain a certain the average number of edges (this value is obtained as the best fit out of several tries and is a little different from the theoretical one).

We chose for our tests  $\gamma = -2.1$  for in degree and  $\gamma = -2.7$  for out degree, like in the actual Internet [13], but these values can be specified at run time. The constant  $\alpha$  has to be given in the command line - in order to obtain a greater or lower edge average for a node.

Here is a table with some pairs N -  $\alpha$  (Table 2) - for 8, the average edges [13].

| N      | $\alpha$ | Theoretical edges average | Obtained edges average |
|--------|----------|---------------------------|------------------------|
| 500    | 3.1      | 7.87                      | 8.39                   |
| 1000   | 2.8      | 7.82                      | 8.36                   |
| 5000   | 2.4      | 7.71                      | 8.51                   |
| 10000  | 2.3      | 7.78                      | 8.67                   |
| 50000  | 2.1      | 7.86                      | 8.96                   |
| 100000 | 2.0      | 7.78                      | 8.96                   |
| 250000 | 1.7      | 6.99                      | 8.18                   |

Table 2: Alpha for different N

We represented the graph as two lists of nodes and neighbours, one for *In* neighbours and one for *out* neighbours. First, we generated the theoretical degree of each node and then connected the nodes.

We generated two sets of numbers obtained by applying the power law function ( $d_v = \alpha \cdot r_v^\gamma$ ), one with  $\gamma -2,1$  and one with  $\gamma -2,7$ .

For both *In* and *Out*, a node is a structure with the following fields:

- nodeSize ID;
- nodeSize edgesAllocated;
- nodeSize edgesNr;
- vector nodeSize neighbours;

where:

**ID** stands for the ID of the node

**edgesAllocate** stands for theoretical number of edges (initially allocated by applying the power law function) and

**edgesNr** for the real number of edges the node will have after running the program.

We allocated the numbers from the two sets as the *In* and *Out* degree for the nodes (the *In* values are sorted and the *Out* are randomly distributed, so that a node with high in degree won't necessary have a high out degree).

Then we connected them all in one single pass through (otherwise it would have taken too much time for graphs with millions nodes). The real number of edges for each node differs from the theoretical one in many cases (by a small amount), but the power laws and the average edges are kept, as you can see in the graphs.

We connect the nodes as follows: first, we connect the nodes with the highest node degree (we will eliminate from the beginning a large number of one neighbour nodes) and then add as many neighbours as possible from the nodes with not yet allocated inneighbours to be the out neighbours.

To run the program, type: `./simulate N  $\alpha$` , where

**N** - number of nodes

$\alpha$  - the coefficient in the Power Law function

$\alpha$  should be chosen so that the average edge number is 8  
(the exponent gamma is -2.1, for in and -2.8 for out )

Output:

**number of average edges** on the resulted graph

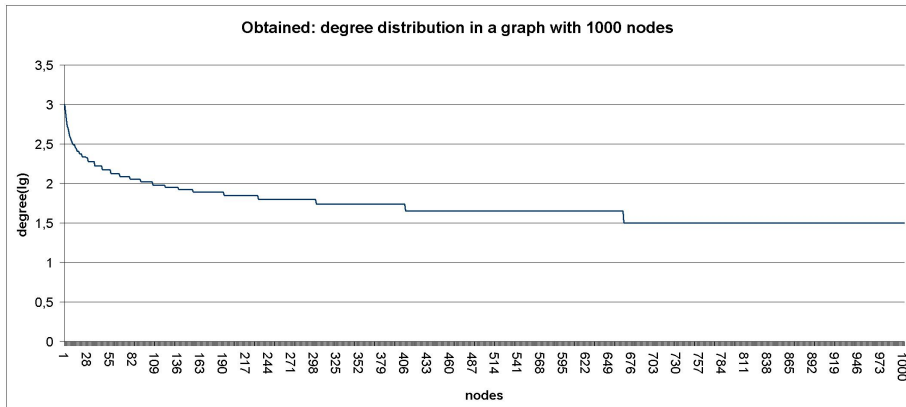


Figure 4: Real degree distribution

**theoretical number of average edges** on the graph with the given parameters

**files** (containing the generated graphs): in.txt out.txt

Each file contains N lines (one for each node) as follows:

n neighbours\_number neighbour1 neighbour2 . . .

### 2.1.2 Experiments and Results

For  $N = 1000$  and  $\alpha = 2.8$ , we obtained the following distribution of nodes table 3:

| Degree | Nodes Nr. |
|--------|-----------|
| 2800   | 1         |
| 653    | 1         |
| 419    | 1         |
| 279    | 1         |
| 252    | 1         |
| ...    | ...       |
| 4      | 68        |
| 3      | 105       |
| 2      | 259       |
| 1      | 334       |

Table 3: Obtained degree distribution (1000 nodes graph)

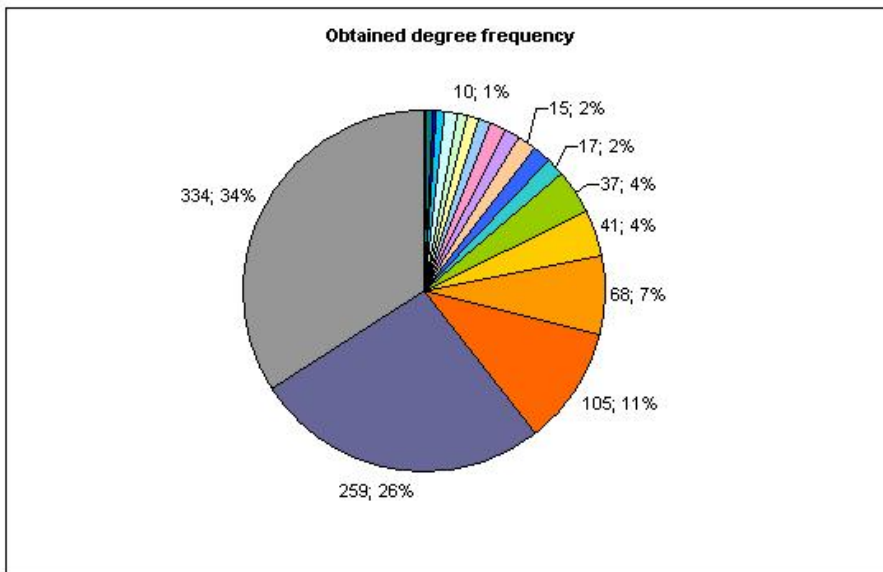


Figure 5: Real degree frequency (procentage)

## 2.2 Data Distribution

We can consider that each node in the graph corresponds to a single page on the Internet, with a set of words. In order to simplify the tests (we are interested only in showing how quick does our algorithm find the answer), we perform searches like this: each query contains a word and only the peers whose file contain the word answer the query.

More sophisticated models could contain some words calculus like: word frequency (in local documents, in all documents etc.)

We used a power law distribution of words: we created a *dictionary file* with words (represented by numbers between 1 and maximum\_word) in a *power law* distribution (1 1 1 1 1...1 1 1 2 2 2...2 2 2.....98 98 99 100 - for n=100 maximum "word") and we allocated to each peer a certain number of words, randomly chosen from the dictionary file.

The number of words for each peer follows also a power law: for example: for 5000 words: we choose

- 10% of total words between 4000 and 5000,
- 20% between 2000 and 4000
- 40% between 500 and 2000
- 50% lower than 5000 words/peer.

### **3 Distributed Personalized Page Rank**

Page Rank is the basis of the most modern search engines (for both Internet and P2P networks). Search engine computations involves crawling speed, index access speed and pagerank speed. The enlarging of the networked brought in front the scalability of these algorithms. Future solutions must focus on the distributed nature of these networks.

We studied some of the existent solutions (3.1) and developed an algorithm to compute personalized Page Rank values in a distributed manner (3.1.3), using the widom algorithm (3.1.2). As expected, the data transfer generated by the algorithm was very low (3.1.4)

We also developed a search algorithm (3.2) which improves current search algorithms by using the personalized Page Rank values (3.3).

#### **3.1 Page Rank and Personalized Page Rank**

Computing Page Rank was a quite researched issue in the past five years, after the papers of Kleinberg [33] and Brin [41, 11] appeared. At the beginning, research was focused only on building Google-like algorithms. However, in time, research has concentrated on two main aspects: search personalization and improving search speed. The former is mostly oriented on developing personalized pagerank algorithms [30, 24, 6, 38, 29]. These algorithms are extensions of the original Google Page Rank algorithm and exploit the personalization vector presented in the paper [41].

A very sensible subject is computational speed. As the size of the network grows, it is impossible to compute Page Rank values in a short period of time (it may take a few days). Some are already trying to build distributed indexes or to compute the pagerank in a distributed manner [31, 27]. The latter approach is proved to be quite effective. A local pagerank is first computed for each strongly connected component of the graph, and then these ranks are combined into an initial approximation of the Google pagerank. The possible parallelism of the first step is obvious.

Many other challenges appear when writing search engine software. Only the exponentially growth of the Web size can be enough for a reason. Every day about 7.3 millions pages are added to the Web and many others are modified or removed [26]. Also, according to [23], current Web graph contains more than 4 billion nodes.

### 3.1.1 Background and Previous Work

Here are the definitions of some terms often used in the literature:

*Forward Link* - a page A has a forward link to a page B if there is a reference (hyperlink) from page A to page B

*Backlink* - a page A has a backlink link from a page B if there is a reference (hyperlink) from page B to page A

*Hub* - a page pointing to many relevant pages (authorities)

*Authority* - a important page in a certain domain - a page pointed back by many hubs

*Page Rank* - the importance of a page, viewed as a "vote" from all the pages that link to it, weighted by their importance

A shortly presentation of most known page ranking algorithms:

**HITS** We can consider the HITS algorithm [33] the first algorithm on page ranking. The idea is to compute two scores for each page in a community, a *hub* score and an *authority* score.

The algorithm starts with a set  $R$  of pages with high pagerank (as if it were computed by a search engine). Then, this set is firstly extended using the following method (later in this paper we shall call it The Kleinberg Extension):

---

#### **HITS 1.** The Kleinberg extension

---

Let  $S_{\Gamma} = R$

For each page  $p$  in  $R$  do

    Let  $\Gamma^{+}(p)$  denote the set of all pages  $p$  points to.

    Let  $\Gamma^{-}(p)$  denote the set of all pages pointing to  $p$ .

    Add all pages in  $\Gamma^{+}(p)$  to  $S_{\Gamma}$

    If  $|\Gamma^{-}(p)| \leq d$  Then

        Add all pages in  $\Gamma^{-}(p)$  to  $S_{\Gamma}$

    Else

        Add an arbitrary set of  $d$  pages from  $\Gamma^{-}(p)$  to  $S_{\Gamma}$

End For

Return  $S_{\Gamma}$

---

After the targeted set of pages is generated, the hubs and authorities score are computed. Two weights are defined for each page  $p$ :

- An authority weight  $x^{<p>}$
- A hub weight  $y^{<p>}$

If  $p$  points to many pages with large  $x$ -values, then it should receive a large  $y$ -value; and if  $p$  is pointed to by many pages with large  $y$ -values, then it should receive a large  $x$ -value. This motivates the definition of two operations on the weights, which we denote by  $I$  and  $O$ . Given weights  $x^{<p>}$  and  $y^{<p>}$  the  $I$  operation updates the  $x$ -weights as follows:

$$x^{<p>} \leftarrow \sum_{q:(q,p) \in E} y^{<q>}$$

Here we considered  $E$  to be the set of existing links. The  $x$ -value of page  $p$  is the sum of all  $y$ -values of pages  $q$  pointing to  $p$ .

Analogously, the  $O$  operation updates the  $y$ -weights as follows:

$$y^{<q>} \leftarrow \sum_{q:(q,p) \in E} x^{<p>}$$

The  $y$ -value of page  $p$  is the sum of all  $x$ -values of pages  $q$  pointed to by  $p$ . The  $I$  and  $O$  operations practically mean that the hubs and authorities are reinforcing one another. The computation of hubs and authorities is now a simple iteration on these formulas, like below:

---

## HITS 2. Hub and authorities scores

---

Initialize  $x_0$  and  $y_0$  with  $(1,1,1,\dots,1)$  in  $R^n$ , where  $n$  is the number of pages.

Given a  $k$  number of steps, apply the following computation  $k$  times:

Apply the  $I$  operation to  $(x_{i-1}, y_{i-1})$  obtaining  $x'_i$

Apply the  $O$  operation to  $(x'_i, y_{i-1})$  obtaining  $y'_i$

Normalize  $x'_i$  obtaining  $x_i$

Normalize  $y'_i$  obtaining  $y_i$

End

Return  $(x_k, y_k)$  as the authority and hub scores required.

---

**Google Page Rank** PageRank is an algorithm for computing a Web page score based on the graph inferred from the link structure of the Web. It is based on the idea that "a page has high rank if the sum of the ranks of its backlinks is high".

So, the Page Rank of each page depends on the Page Rank of the pages pointing to it. But we will not know what Page Rank those pages have until the pages pointing to them have their Page Rank computed and so on.

We will compute after some iterations the vector of the whole Internet. We begin with  $x$ , an  $N$ -dimensional vector ( $N$  - the number of Internet pages) with 1 (or some other value between 0 and 1) on each position and compute the values as follows:

$$x^{n+1} \leftarrow c \cdot A \cdot x^{(n)}$$

where  $x$  is the page rank vector(for the whole Internet),  $A$  the web matrix.

If you consider that page links can form circles it seems impossible to do this calculation. The initial solution of Google authors was to compute the first eigenvector of the adjacency matrix of the Web graph. This raised some other problems, like how to store such a huge matrix, or how to compute the eigenvector in a reasonable amount of time. Google authors did not present a solution to this, but a lot of research in this direction followed.

This algorithm worked well, but experimentally some problems appeared:

1. *Dead ends*: A page that has no successors has nowhere to send its importance. Eventually, this page will induce a 0 pagerank to many pages that forward link to it.
2. *Spider traps*: A group of pages that has no outgoing links will eventually accumulate all the importance of the Web. While their pagerank will increase at each iteration, the pagerank of the other pages will be almost constant.

so, the new formula becomes:

$$x^{n+1} \leftarrow c \cdot A \cdot x^{(n)} + (1 - c) \cdot e$$

where  $x$  is the page rank vector,  $c$  is a constant, the dumping factor, and  $e$  is a vector.

The reasoning behind this formula is called *The Random Surfer Model*. It states that an imaginary random surfer will choose an outgoing link of the current page with probability  $c$ , or will get bored and choose another random page with probability  $(1 - c)$ . The  $e$  vector is called personalization vector, because it can be used to direct the random surfer only to some preferred pages (if the fields corresponding to them will be 1 and all the other fields will be 0, instead of having all fields equal to 1).

Quoting from the original Google paper, PageRank is defined like this:

*”We assume page  $A$  has pages  $T1...Tn$  which point to it (i.e., are citations). The parameter  $d$  is a dumping factor which can be set between 0 and 1. We usually set  $d$  to 0.85. There are more details about  $d$  in the next section. Also  $C(A)$  is defined as the number of links going out of page  $A$ . The PageRank of a page  $A$  is given as follows:*

$$PR(A) = (1 - d) + d(PR(T1)/C(T1) + \dots + PR(Tn)/C(Tn))$$

*Note that the PageRanks form a probability distribution over web pages, so the sum of all web pages PageRanks will be one. PageRank or  $PR(A)$  can be calculated using a simple iterative algorithm, and corresponds to the principal eigenvector of the normalized link matrix of the Web.”*

**Convergence Proof for Pagerank** Let us first state that the normalized adjacency matrix of the Web graph is a Markov matrix. A Markov matrix is the transition matrix for a finite Markov chain, also called a stochastic matrix. Elements of the matrix must be real numbers in the closed interval  $[0, 1]$ . A completely independent type of stochastic matrix is defined as a square matrix with entries in a field  $F$  such that the sum of elements in each column equals 1. This is the case for our normalized adjacency matrix.

The intuition behind the convergence of the power method is as follows. For simplicity, assume that the start vector  $x^{(0)}$  lies in the subspace spanned by the eigenvectors of  $A$ . Then  $x^{(0)}$  can be written as a linear combination of the eigenvectors of  $A$ :

$$x^{(0)} = u_1 + a_2 u_2 + \dots + a_m u_m$$

Since the first eigenvalue of a Markov matrix is  $1 = 1$ , we have:

$$x^{(1)} = Ax^{(0)} = u_1 + a_2 \lambda_2 u_2 + \dots + a_m \lambda_m u_m$$

and:

$$x^{(n)} = A^n x^{(0)} = u_1 + a_2 (\lambda_2)^n u_2 + \dots + a_m (\lambda_m)^n u_m$$

Since  $\lambda_m < \dots < \lambda_2 < \lambda_1$ ,  $A^n x^{(0)}$  approaches  $u_1$  as  $n$  grows large. Therefore, the Power Method converges to the principal eigenvector of the Markov matrix  $A$ .

**Theorem 1** Let  $P$  be an  $n \times n$  row-stochastic matrix. Let  $c$  be a real number such that  $0 < c < 1$ . Let  $E$  be the  $nn$  rank-one row-stochastic matrix  $E = ev^T$ , where  $e$  is the  $n$ -vector whose elements are all equal to 1, and  $v$  is an  $n$ -vector that represents a probability distribution.

Define the matrix  $A = [cP + (1 - c)E]^T$ . Its second eigenvalue is  $|\lambda_2| \leq c$ .

**Theorem 2** Further, if  $P$  has at least two irreducible closed subsets (which is the case for the Web hyperlink matrix), then the second eigenvalue of  $A$  is given by  $\lambda_2 = c$ . A proof of these theorems can be found in [40].

**Convergence speed of PageRank** The PageRank algorithm uses the power method to compute the principal eigenvector of  $A$ . The rate of convergence of the power method is given by  $|\lambda_2/\lambda_1|$ . For PageRank, the typical value of  $c$  has been given as 0.85; for this value of  $c$ , theorem 2 thus implies that the convergence rate of the power method  $|\lambda_2/\lambda_1|$  for any web link matrix  $A$  is 0.85 (note that the first eigenvalue for a Markov Matrix is always 1). Therefore, the convergence rate of PageRank will be fast, even as the web scales.

**Topic Sensitive Page Rank** Another important paper is [28]. It starts by computing offline a set of 16 pagerank vectors oriented on the 16 main topics of the Open Directory Project (ODP). This way, importance scores relative to topics are given. As a further artifact, artificial links may be introduced in the Web graph to further focus towards one or another topic.

Two search scenarios are imagined: normal search, search in context (a text from a Web page, e-mail, etc.). Vectors with number of occurrences of the terms in each class are computed and used to compute for a query  $q$  the class probabilities of each of the 16 top-level topics.

Furthermore, in place of the uniform dumping vector from the PR algorithm  $p = [1/N]$ , the author uses a non uniform dumping vector  $v_j$  for category  $j$ , where  $v_{ji}$  is:

$$v_{ij} = \begin{cases} T_0[p](q) & , i \in T_j \\ 0 & , otherwise \end{cases} \quad (1)$$

( $T_j$  is the set of URLs in the category  $j$ .)

Also an unbiased vector is computed for comparison. Finally the 16 page ranks are combined into one using the different class probabilities (depending on each user's query).

The experimental results of the execution of this algorithm show a 60% improved precision on Google pagerank for each topic searched.

**Sim Rank** In [29] the authors are computing a node-pairs graph  $G^2$  with SimRank similarity scores, starting from a graph  $G$ . SimRank is defined based on the theory that two objects are similar if they are referenced by similar objects. In the left side we have the initial graph  $G$  (which can be any - Web - graph), and in the right side there is the corresponding computed node-pairs graph ( $G^2$ ).

There are several possible similarity formulas presented in the paper. The graph above was computed using  $s(a, b)$  as 1 if  $a = b$  and otherwise:

$$s(a, b) = \frac{C}{|I(a)||I(b)|} \sum_{i=1}^{|I(a)|} \sum_{j=1}^{|I(b)|} s(I_i(a), I_j(b))$$

However, our interest went towards those formulas where Web links could be exploited, like in these:

$$s(a, b) = \frac{C_1}{|O(a)||O(b)|} \sum_{i=1}^{|O(a)|} \sum_{j=1}^{|O(b)|} s_2(O_i(a), O_j(b))$$

$$s(a, b) = \frac{C_2}{|I(a)||I(b)|} \sum_{i=1}^{|I(a)|} \sum_{j=1}^{|I(b)|} s_1(I_i(a), I_j(b))$$

Here, two similarity scores are computed, one for out-going links and one for in-going links. By  $O(a)$  and  $I(a)$  are denoted the sets of pages pointed to by page  $a$  and respectively pointing to  $a$ .  $C$ ,  $C_1$  and  $C_2$  are constants set to 0.800 in the original experiments and they represent degeneration factors (how much a similarity between two pages is degenerated by the distance between them).

These formulas represent generalizations of the bi-partite SimRank formulas for the Web. The first score is a hub similarity score, while the second is an authority similarity one.

The node-pairs graph can be used to compute similarities between either Web pages (using links) or between scientific papers (using citations), etc.

**Other approaches** [10] attempts to find the documents related to the topic of the query (not only those that conform precisely to the query match). Techniques for

tackling scenarios like "Mutually reinforcing relationships between hosts" (a set of documents on one host points to one document on a second host, or a document on one host points to many documents from a second host), "Automatically generated links" and "Non-relevant documents" are also presented.

The authors compute the relevance for the starting Web nodes (their similarity score) using a subset of the first 1000 words of each document  $D$  considered to be the query, and then the coefficient similarity  $(Q, D)$ . They modify the Kleinberg algorithm to consider also the relevance of a node, together with its hub and authority value. The algorithm has high computation time. A solution for that is the pruning of some documents from the computation. Two approaches are tested: Degree based pruning (based on the in\_degree and out\_degree of the nodes) and Iterative pruning (after each sequence of 10 iterations of the modified Kleinberg algorithm, fetch the most relevant documents until an allotted quota of documents has been fetched or enough top ranked documents have been found).

[43] presumes that users are interested in a set of latent topics which in turn generate both items and item content information. Model parameters are learned using expectation maximization, so the relative contributions of collaborative and content-based data are determined in a statistical manner. Users, together with the documents they access form observations. These observations are associated to topics. After the model is learned, documents can be ranked for a given user (how likely it is that the user will access the corresponding document). Documents not seen and with high probability are good candidates for recommendation. The model is then extended to consider the document words (content) when calculating the above mentioned probability.

[24] presents Open Rating Systems and the problems they arise: aggregation (the ratings of many sources must be aggregated into one ranking), and meta-ranking (the ratings may be serious or may have a poor quality; consequently, the ratings must also be ranked). The rating agents should also be rated by other agents.

Two goals are proposed: rating (complete the agents-resources rating schema, that is if Jane did not rate resource  $j$ , predict what her rating will be) and ranking (resources can be ranked from a global perspective and from the perspective of a given agent). The web graph  $G$  is defined as containing agents and objects as nodes and arcs from agents to agents when the former agent trusts the latter and from agents to objects when the agent has rated the object positively. Then, the Brin pagerank algorithm is applied to  $G$ . The negative ratings are considered afterwards by subtracting a value depending on them. Therefore, the rating of a page can be negative in the end.

Another proposed algorithm is to compute separately Trust Ranks and Distrust Ranks and then combine them in several modes. For augmenting the web of trust, an agent may believe the opinions of one or more globally trusted agents. Such

a rating based search-engine is the Romanian Top 100 ([www.top100.ro](http://www.top100.ro)). Simpler algorithms seem to be behind it, though. More complex such web sites are [5], [49], and [19] (this last one is also analyzed in the [24] article).

[9] is focusing on computing a query specific sub-graph of the main graph (ignoring pages not on the topic) and compute the page scores only for that sub-graph. The limitation is obvious: only popular topics can be covered because the ranking cannot be done in real time. The main algorithm proposed in the paper, called hilltop, has this same limitation. When computing the response to a query, a list of the most relevant experts on the query topic is first created (pages created with the specific purpose of directing people towards resources). Then, the relevant links within the selected set of experts are identified and followed, and finally the resulting pages are ranked using the number and relevance of the non-affiliated experts pointing to them. If such experts are not available, no results are returned. An expert page is a page about a certain topic and with links to many non-affiliated pages on that topic. Two pages are non-affiliated if they are authored by authors from non-affiliated organizations.

[14] provides incremental updates to the pagerank vector, once it has been calculated. They identify a small portion of the Web in the vicinity of the link changes and consider the rest of the Web as a single node. Then, computes a version of pagerank on this small graph and transfers the results to the original one.

The authors define the transition probabilities from the small graph  $G$  to/from the super-node  $W \setminus G$  (where  $W$  is the web graph). The graph  $G$  is build considering the pages whose pagerank is likely to be changed by the insertion of a link  $(i, j)$ . Assign the weight 1 to page  $i$  and dissipate it through its children, i.e. all children will have the weight of  $(1 - e)d_i$ , where  $d_i$  is the out-degree of  $i$  and  $e$  is the pagerank dumping factor. The chosen nodes must exceed a given weight. If more links are added/deleted, a BFS is performed around them and then the weighting method is applied. The small graph  $G$  will be the union of the nodes discovered in this process.

[7] analyses the Brin pagerank algorithm and two proposed improvements. The authors start from the equation  $x = aAx + (1 - a)e$ , where  $A$  is the Web graph matrix,  $a$  is the dumping factor and  $e$  is the personalization vector. The first improvement modifies the above equation into  $(I - aA)x = (1 - a)e$ . At this point, instead of using the Jacobi iteration, a similar but faster convergent Gauss-Seidel based solution is proposed. Furthermore, the latter equation can be decomposed into a sequence of smaller problems, which take fewer iterations individually to converge.

### 3.1.2 Widom Algorithm - Centralized

There have been several recent approaches to distributedly compute the Page Rank scores. Their motivation arises not only from the continuously increasing size of the Web graph [46, 47] but also from the need of reputation models in P2P networks [51, 32]. However, there is no algorithm which computes *personalized* page ranks in a distributed fashion.

In a P2P environment, each peer is computing the ranks of its documents, sending the intermediate results to all documents to which its documents are linked (out-links) [46]. The overall performance can be increased as in [47], where selected peers handle the computation for bigger groups of documents. High overall speed is obtained reducing the communication between peers to the minimum, provided that they are not overloaded by their local computations.

*Description.* [30] is the most recent investigation towards personalized page ranks. One Personalized PageRank Vector (PPV) is computed for each user. The personalization aspect of this algorithm stems from a *set of hubs* (H), each user having to select her *preferred pages* from it. PPVs can be expressed as a linear combination of basis vectors (PPVs for preference vectors with a single non-zero entry corresponding to each of the pages from P, the preference set), which could be selected from the precomputed basis hub vectors, one for each page from H. To avoid the massive storage resources the basis hub vectors would use, they are decomposed into partial vectors (which encode the part unique to each page, computed at run-time) and the hub skeleton (which captures the interrelationships among hub vectors, stored off-line).

In the paper we will use a notation similar to [30].  $G = (V, E)$  represents the *Web graph*, where  $V$  is the set of all Web pages and  $E$  is the set of directed edges  $\langle p, q \rangle$ .  $E$  contains an edge  $\langle p, q \rangle$  iff a page  $p$  links to page  $q$ .  $I(p)$  denotes the set of in-neighbors (pages pointing to  $p$ ) and  $O(p)$  the set of out-neighbors (pages pointed to by  $p$ ). For each in-neighbor we use  $I_i(p)$  ( $1 \leq i \leq |I(p)|$ ), the same applies to each out-neighbor. We refer  $v(p)$  to denote the  $p$ -th component of  $v$ . We will typeset vectors in boldface and scalars (e.g.,  $v(p)$ ) in normal font.

Let  $A$  be the adjacency matrix corresponding to the Web graph  $G$  with  $A_{ij} = \frac{1}{|O(j)|}$  if page  $j$  links to page  $i$  and  $A_{ij} = 0$  otherwise.

*Algorithm.* In the first part of the paper, the authors present three different algorithms for computing basis vectors: "Basic Dynamic Programming", "Selective Expansion" and "Repeated Squaring". In the second part, specializations of these algorithms are combined into a general algorithm for computing PPVs, as

depicted below.

---

**Algorithm 1.** Personalized PageRank in a centralized fashion.

---

Let  $D[p]$  be the approximation of a basis vector corresponding to page  $p$ , and  $E[p]$  the error of its computation.

**1.(Selective Expansion)** Compute the partial vectors using

$Q_0(p) = V$  and  $Q_k(p) = V \setminus H$ , for  $k > 0$ , in the formulas below:

$$\mathbf{D}_{k+1}[\mathbf{p}] = \mathbf{D}_k[\mathbf{p}] + \sum_{q \in Q_k(p)} c \cdot E_k[p](q) \mathbf{x}_q$$

$$\mathbf{E}_{k+1}[\mathbf{p}] = \mathbf{E}_k[\mathbf{p}] - \sum_{q \in Q_k(p)} E_k[p](q) \mathbf{x}_q + \sum_{q \in Q_k(p)} \frac{1-c}{|O(q)|} \sum_{i=1}^{|O(q)|} E_k[p](q) \mathbf{x}_{O_i(q)}$$

Under this choice,  $D_k[p] + c * E_k[p]$  will converge to  $\mathbf{r}_p - \mathbf{r}_p^H$ , the partial vector corresponding to  $p$ .

---

**2.(Repeated squaring)** Having the results from the first step as input, one can now compute the hubs skeleton ( $r_p(H)$ ). This is represented by the final  $D[p]$  vectors calculated using  $Q_k(p) = H$  into:

$$\mathbf{D}_{2k}[\mathbf{p}] = \mathbf{D}_k[\mathbf{p}] + \sum_{q \in Q_k(p)} E_k[p](q) * D_k[q]$$

$$\mathbf{E}_{2k}[\mathbf{p}] = \mathbf{E}_k[\mathbf{p}] - \sum_{q \in Q_k(p)} E_k[p](q) \mathbf{x}_q + \sum_{q \in Q_k(p)} E_k[p](q) E_k[q]$$


---

**3.** Let  $u = \alpha_1 p_1 + \dots + \alpha_z p_z$  be a preferred vector,

where  $p_i$  are from  $H$  and  $i$  is between 1 and  $z$ , and let:

$$r_u(h) = \sum_{i=1}^z \alpha_i (r_{p_i}(h) - c * x_{p_i}(h)), \quad h \in H$$

which can be computed from the hubs skeleton.

The PPV  $v$  for  $u$  can then be constructed as:

$$v = \sum_{i=1}^z \alpha_i (r_{p_i} - r_{p_i}^H) + \frac{1}{c} \sum_{h \in H} r_u(h) \cdot [(\mathbf{r}_h - \mathbf{r}_h^H) - c \cdot x_h]$$


---

### 3.1.3 Widom Algorithm - Distributed

In the distributed algorithm for computing personalized reputation values we start from a set  $H$  of pre-trusted peers (called *hub peers* hereafter). Each peer will then have its own preference set  $P \subset H$ .

We divide the algorithm in three parts, as presented in section 3.1.2: One part focuses mostly on peers outside the set of pre-trusted peers,  $V \setminus H$  (Algorithms 3.1.1 and 3.1.2), one on peers within the set of pre-trusted peers  $H$  (Algorithm 3.2), and the final algorithmic step ties everything together (Algorithm 3.3).

**Partial Vectors** This part of the algorithm consists of one special initialization step and several succeeding steps. Even though its focus is on peers from  $V \setminus H$ ,

peers  $p \in H$  will also gather their components of  $\mathbf{D}$  and  $\mathbf{E}$ . In the first step, each peer  $q \in V$  will compute  $E[p](q)$ . As peers  $p \in H$  are known, each peer  $q \in V$  can set its initial values  $D_0[p](q)$  and  $E_0[p](q)$  by itself, as below:

$$D_0[p](q) = \begin{cases} c & , q \in H \\ 0 & , otherwise \end{cases} \quad (2)$$

$$E_0[p](q) = T_0[p](q) = \begin{cases} 1 & , q \in H \\ 0 & , otherwise \end{cases} \quad (3)$$

**Non-hub peers.** After this initialization, the following operations will be executed in parallel by each peer  $q \in V \setminus H$  for each  $p \in H$ :

---

**Algorithm 3.1.1.** Distributed computation of partial vectors by peers in  $V \setminus H$ .

---

- 1: Send  $\frac{1-c}{|\mathbf{O}(q)|} \cdot T_k[p](q)$  to all peers from which  $q$  has downloaded files, including those from  $H$ .
  - 2: Wait from all peers which have downloaded files from  $q$  their  $T_k[p](*)$  at this step ( $k$ )
  - 3: After all  $T_k[p](*)$  values have been received, compute  $T_{k+1}[p](q)$  as:  

$$T_{k+1}[p](q) = \sum_{v \in \mathbf{I}(q)} T_k[p](v)$$
  - 4: Compute:  

$$D_{k+1}[p](q) = D_k[p](q) + c \cdot T_k[p](q)$$
 $N$  being the number of steps we apply the Selective Expansion algorithm.
  - 5: If there are more iterations left, go to 1
  - 6: Each peer  $q \in V \setminus H$  computes  $(r_p - r_p^H)(q) = D_N[p](q) + c \cdot T_N[p](q)$ , its component of the partial vector corresponding to  $p$ .
- 

In the distributed version of the Selective Expansion algorithm, for  $p \in H$  each peer  $q \in V \setminus H$  has to compute:

$$T_{k+1}[p](q) = \sum_{v \in \mathbf{I}(q)} \frac{1-c}{|\mathbf{O}(v)|} \cdot E_k[p](v) \quad (4)$$

$$\begin{aligned}
E_{k+1}[p](q) &= \left( E_k[p] - \sum_{v \in V \setminus H} E_k[p](v) \cdot x_v \right) (q) + \\
&\quad + \left( \sum_{v \in V \setminus H} \frac{1-c}{|\mathbf{O}(v)|} \sum_{i=1}^{|\mathbf{O}(v)|} E_k[p](v) \cdot x_{O_i(v)} \right) (q) = \\
&= E_k[p](q) - (E_k[p](q) \cdot x_q) (q) + \\
&\quad + \left( \sum_{v \in I(q)} \frac{1-c}{|\mathbf{O}(v)|} \sum_{i=1}^{|\mathbf{O}(v)|} E_k[p](v) \cdot x_{O_i(v)} \right) (q) = \\
&= \left( \sum_{v \in I(q)} \frac{1-c}{|\mathbf{O}(v)|} E_k[p](v) \cdot x_q \right) (q) = \\
&= \sum_{v \in I(q)} \frac{1-c}{|\mathbf{O}(v)|} \cdot E_k[p](v)
\end{aligned}$$

**Hub peers.** In the special first step, a peer  $p \in H$  will send  $\frac{1-c}{|\mathbf{O}(q)|} \cdot T_k[p](p)$  to its all peers from which it has downloaded files, including those from  $H$ . After that, it will execute the following operations:

---

**Algorithm 3.1.2.** Distributed computation of partial vectors by peers in  $H$ .

---

- 1: Wait from all peers which have downloaded files from  $p$  their  $T_k[p](*)$  at this step ( $k$ )
  - 2: After all  $T_k[p](*)$  values have been received, compute  $T_{k+1}[p](p)$  as  $T_{k+1}[p](p) = \sum_{v \in I(q)} T_k[p](v)$
  - 3: If there are more iterations left, go to 1
  - 4: Compute:  $E[p](p) = \sum_{k=1}^N T_k[p](p)$
  - 5: Set  $D_N[p](p)$  to  $c$
  - 6: Each peer  $p \in H$  computes  $(r_p - r_p^H)(p)$ , its component of its partial vector.
- 

Algorithms 3.1.1 and 3.1.2 perform a power iteration, and they would therefore converge also in the presence of loops in  $G$  (see [30, 41] for details), whereas the high dinamicity of a P2P network would only result in some additional iterations until convergence.

**Hub Skeleton** In the second phase of the algorithm, each peer  $p \in H$  (pre-trusted, or hub peer) has to calculate its hub skeleton ( $\mathbf{r}_p(\mathbf{H})$ ) using as input the results from the previous stage. The result is stored in the values  $\mathbf{D}_{2k}[\mathbf{p}]$  obtained after the last iteration of the following operations, performed by each  $p \in H$ :

---

**Algorithm 3.2.** Distributed computation of hub skeleton (only in  $H$ ).

---

- 1: Calculate  $\mathbf{D}_{2k}[\mathbf{p}]$  and  $\mathbf{E}_{2k}[\mathbf{p}]$ , using the same formulas as the centralized version.
  - 2: Multicast the results to all other peers  $q \in H$ , possibly using a minimum spanning tree for that.
  - 3: If there are more iterations left, go to 1.
  - 4: Every hub peer broadcasts its  $\mathbf{D}_{2N}[\mathbf{p}]$  sub-vector (only the components regarding pages from  $H$ ).
- 

As this step refers to hub-peers only, the computation of  $\mathbf{D}_{2k}[\mathbf{p}]$  and  $\mathbf{E}_{2k}[\mathbf{p}]$  can consider *only* the components regarding pages from  $H$ .

**PPV** As the  $\mathbf{D}_{2N}[\mathbf{p}]$  sub-vectors ( $p \in H$ ) have been broadcast, *any* peer  $v \in V$  can now determine its  $\mathbf{r}_v(\mathbf{P})$  locally, using the original formula .

*Any* peer  $v \in V$  can also calculate its partial PPV containing its reputation and the reputation of any other peers from its own point of view. If we denote  $NB$  the set of peers whose rank  $v$  wants to compute, it must do the following:

---

**Algorithm 3.3.** Computation of the Personalized Reputation Vector.

---

- 1: Request the components of  $\mathbf{r}_p - \mathbf{r}_p^H$  for all  $p \in H$  from all peers from  $NB$ .
  - 2: Compute the components of the PPV using the original formula.
- 

Of course, if later  $v$  wants to compute the reputation value of another peer, it would only need to ask the new peer about its components of  $\mathbf{r}_p - \mathbf{r}_p^H$ .

### 3.1.4 Experiments and Results

Unless stated differently, the set of pre-trusted peers contained 5 (hub) peers, and the preference set of each ordinary peer contained 2-3 hub peers (out of the

5 from above) randomly selected. Different numbers of ordinary good and malicious peers were present, depending on the experiment. In order to assess the amount of network traffic required by the computation of reputation values. The set of pre-trusted peers contained 150 (hub) peers, while the preference set of each ordinary peer contained 30 hub peers (out of the 150 from above) randomly selected. Results are summarized in tables 4 and 5.

|                 | <b>Max. Size</b> | <b>Avg. Size</b> |
|-----------------|------------------|------------------|
| <b>All Data</b> | 1,989,072        | 1,979,695        |
| <b>≠ 0 Data</b> | 1,944,764        | 1,938,796        |

Table 4: Data transferred by hub peers (nb. of real values sent)

|                              | <b>Max. Size</b> | <b>Avg. Size</b> |
|------------------------------|------------------|------------------|
| <b>All Data</b>              | 101,232          | 4,231.3          |
| <b>≠ 0 Data</b>              | 5,360            | 84.69            |
| <b>All Data, 1 iteration</b> | 665              | 27.91            |
| <b>≠ 0 Data, 1 iteration</b> | 171              | 2.043            |

Table 5: Data transferred by ordinary peers (nb. of real values sent)

In our statistics, we distinguish between ordinary peers and hub peers. Both peer types perform different operations, and the amount of data sent differs quite a lot. During the simulation, we discovered many peers often send the value "0" through the network (as a correct intermediate value). We decided also to count these "special" 0 values in order to verify whether a further modification of the algorithm which would avoid sending them is indeed useful or not. The current experimental results indicate that this is the case.

The hub peers generate significantly more traffic because of the second phase of the computation (algorithm 3.2), in which they need to multicast their intermediate results to all other hub peers in the network. However, there is usually a small number of hub peers compared to the size of the network, and they also control more resources (e.g. bigger bandwidth or processing power). Finally, we distinguish the reduced communications of ordinary peers, which means that for the majority of the network, the computation would need only small bandwidth (per iteration, an ordinary peer needs to send about two non-zero values, for example).

Figures 6 and 7 present the traffic generated by each peer in the network. As we are dealing with a power-law network, one can observe the power-law distri-

bution of data size among the ordinary peers. This is because the more neighbors a peer has, the more data it needs to exchange.

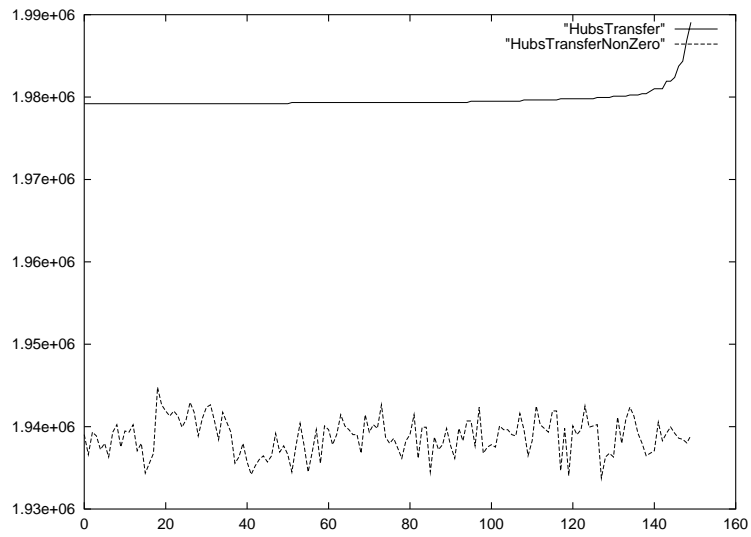


Figure 6: Data transfered by hub peers (nb. of real values sent)

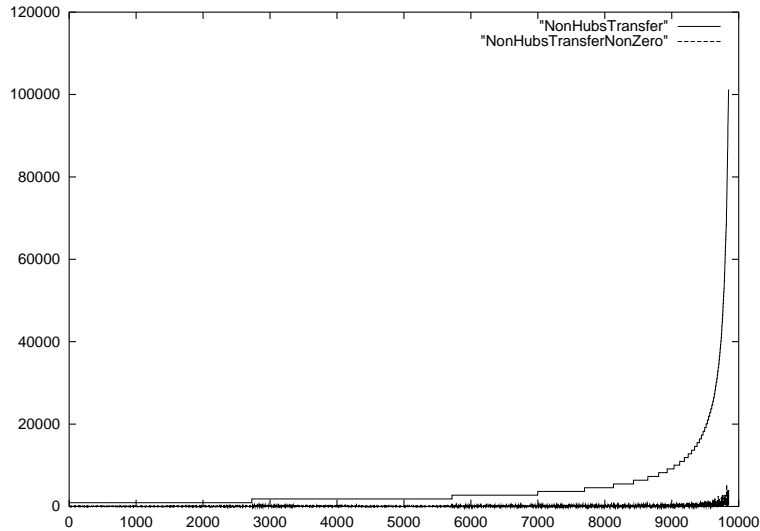


Figure 7: Data transfered by ordinary peers (nb. of real values sent)

## 3.2 Application: Search Algorithm using Personalized Page Rank

### 3.2.1 Background and Previous Work

Searching P2P networks depends on the architecture used (e.g. super-peer based or pure P2P networks), on the expected availability, etc. Although techniques as those of Chord [48] or CAN [44] guarantee location of content within a limited number of hops, they exhibit a tight control over the network (e.g. topology, placement, etc.). Similar to [52] our work is focused on file-sharing systems with less tight constraints, such as Gnutella [21] or Freenet [20].

[52] explores three techniques, called "Iterative Deepening", "Directed BFS" and "Local Indices". The first one is about initiating multiple breadth-first searches with successively larger depth limits until either enough answers to the query have been obtained or the maximum depth limit has been reached. It does not necessarily visit the best peers which can answer the query, but it generates lower network traffic than the other approaches. "Directed BFS" sends the query only to the "best" neighbors of a peer, which are computed by judging the previous experiences achieved when querying. In this paper, we use PPR values, which do not include only the local experience, but also the experiences of all other peers.

The "Local Indices" technique deals with building descriptions of the content of all neighboring peers, thus allowing fast identification of peers able to answer the query. This is orthogonal to our method, but we intend to investigate how they could be combined to provide better results.

Two other search techniques are discussed in [3], namely one in which queried neighbors are randomly selected, and another in which the unvisited neighbor with the highest degree is queried. Although eventually the former method gravitates around high-degree nodes, the latter one reaches them faster. However, using PageRank scores as criterion for determining the queried node should provide even better results than simply using the node-degree.

### 3.2.2 Our Algorithm - Detailed View

One can introduce page ranks into several of the existing P2P searching techniques. We present an approach similar to [3], and focus the forwarding of queries by using rank values we have computed in a distributed fashion:

---

**Algorithm 4.** Searching P2P Networks using Distributed Personalized PageRanks.

---

- 1: Upon issuing a query, a peer  $p$  will send it to its neighbor having the highest rank from its perspective (or the neighbor containing the highest ranked page). It will attach to the query its own ID (or IP address), the number of results desired and a TTL (time to live).
  - 2: Upon initial receiving of a query, a peer  $q$  will:
    - 2.1: Add its own ID (or IP address) to the query and decrease the TTL.
    - 2.2: Check if the local page matches the query. If yes, then send it to  $p$  and decrease the number of results desired.
    - 2.3: If the number of results desired is more than zero, select the neighbor with the highest rank whose ID is not already added to the query (highest ranked unvisited neighbor) and forward the query to it.
    - 2.4: If there is no unvisited neighbor, send a special message back to the peer from which the query was obtained. Put in the message the latest instance of the query (i.e. with latest list of visited peers and most recent number of desired results).
  - 3: Upon receiving a query back from a neighboring peer, go to step (2.3).
- 

## 3.3 Experiments and Results

We analyzed our algorithm against several similar approaches:

1. Best-neighbor search (as in [3]: at each step, the query is forwarded to the unvisited neighbor having the biggest number of neighbors).
2. PageRank search: forward the query to the neighbor with the highest PageRank (i.e. without personalization).

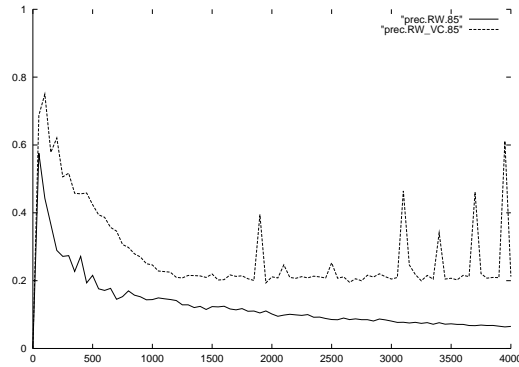


Figure 8: Precision of random walk search, with and without visiting same peers again

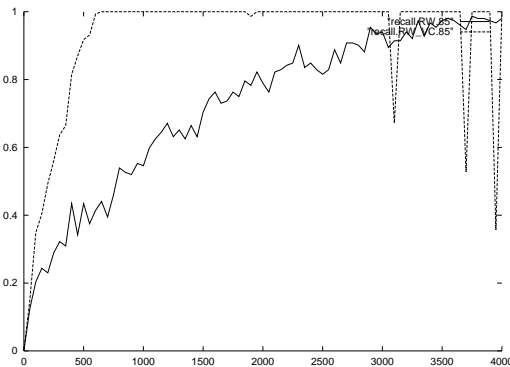


Figure 9: Recall of random walk search, with and without visiting same peers again

The latter solution is new. Even though there are several articles building PageRank in a distributed manner, they do not use our search approach.

We use a simulated power-law network with the same exponents as in the previous sub-section, but with 10,000 peers. Each of them has one document

with a size varying from 100 to 5000 again after a power-law distribution. The random words we used also exhibit a probability of appearance in a document which follows a power-law.

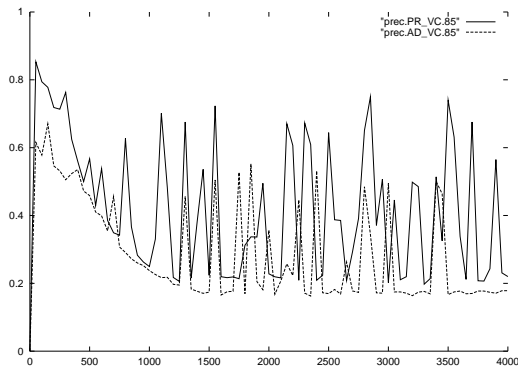


Figure 10: Precision of Best Neighbor and PageRank-based search

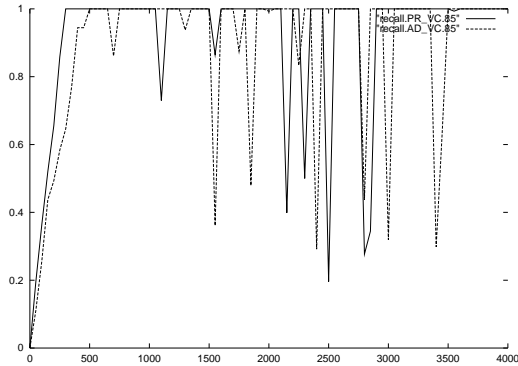


Figure 11: Recall of Best Neighbor and PageRank-based search

In order to evaluate the results, we used precision and recall with respect to the top PageRank results, as well as an estimation of the necessary TTL to get some specific results. When not specified, all our graphs contain the TTL on the x-axis.

$$Precision = \frac{\text{Nb. of Top 200 matches found}}{\text{Total nb. of matches found}}$$

$$Recall = \frac{\text{Nb. of Top 200 matches found}}{\text{Nb. of Top 200 matches}}$$

We started with some experiments inspired from the walk of a random surfer [41], i.e. in which a peer forwards a query using some algorithm to determine the next target with probability  $c < 1$ , and to any randomly selected peer with probability  $1 - c$ . Note that jumping to a peer in a P2P network is not straightforward, but we included these initial searches for comparison purposes.

In the first two experiments (figures 8 and 9), the peer randomly selects one of his neighbors with probability  $c = 0.85$  and jumps to any peer with probability 0.15. The difference in precision and recall is given by the fact that in RW\_VC we ensured that the query will not visit the same peer again unless absolutely necessary (there is no other neighbor to visit). The small irregularities in the graphs are given by the random jumps which sometimes lead to "poor" peers.

We continued in the same manner, testing a neighbor-based and a PageRank-based search (figures 10 and 11). As predicted, the latter approach performs slightly better. Although selecting the neighbor with the largest number of neighbors finds best peers relatively fast, it usually cannot find them faster than when the actual best neighbor is selected.

We also tested these last two experiments without random jumps. We found both better precision and better recall when using PageRank to select the neighbor where to forward the query, especially with a  $TTL > 150$ . Again we observed some small irregularities, but they are normal considering the fact that all documents were randomly generated at the beginning of each run (in order not to have them stored in a file).

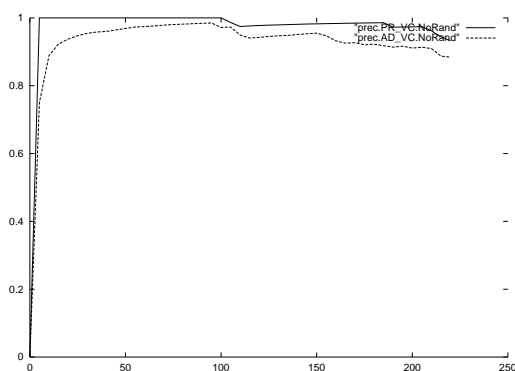


Figure 12: Precision of Best Neighbor and PageRank-based search

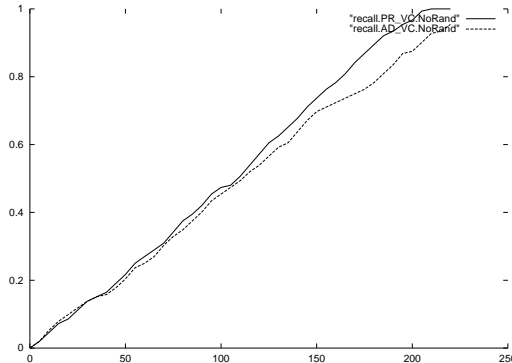


Figure 13: Recall of Best Neighbor and PageRank-based search

Finally, we wanted to simulate the PPR search. As all our documents are randomly generated, we were forced to use the following heuristic: in PPR the documents from my preference set will contain the keywords of my interest with a higher probability ( $prob' = \alpha \cdot prob$ ) than in PageRank, because top PPR documents will be on the same topic as my topic of interest. However, we cannot estimate how much higher this will be (considering our experience from [15],  $\alpha$  varies from one preference set/topic to another). Therefore, we decided to test using  $\alpha = \{2, 3\}$ , being sure that  $\alpha > 1$ . The results are depicted in figures 14 and 15. Even though PPR (PR\_VC\_2, PR\_VC\_3) performed better than PageRank, increasing  $\alpha$  over 2 does not seem to bring much improvements.

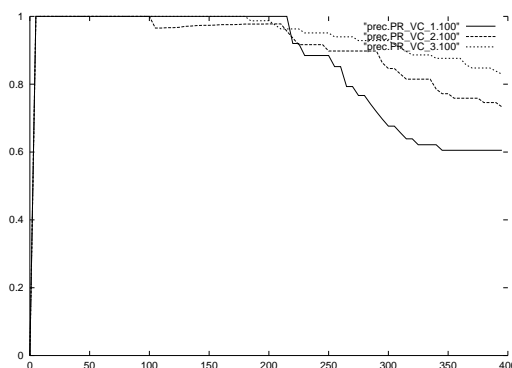


Figure 14: Precision of simulations of the PPR

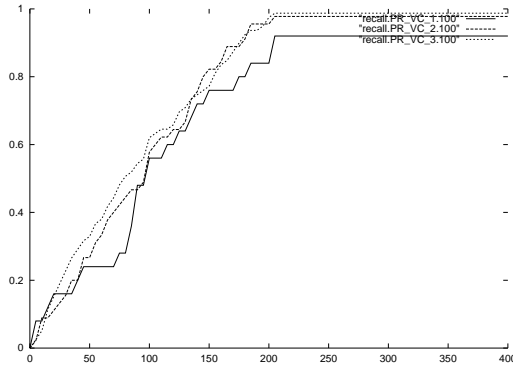


Figure 15: Recall of simulations of the PPR

In the last experiment we measured the TTL needed to get the top 30 (figure 16) and top 50 (figure 16) PageRank documents using different algorithms: neighbor-based search (AD), PageRank, Personalized PageRank  $\alpha = 2$  (PR\_2) and Personalized PageRank  $\alpha = 3$  (PR\_3). On the x-axis there are different words, "10000" being the most popular of them and "0" the least popular. We can see that especially for unpopular words, PPR-based search finds the best results much faster (e.g. getting the top 30 results querying for the most popular word needs TTL 38 for both AD and PPR, while for say word 6000, we need TTL 200 for AD, but only 100 for PPR).

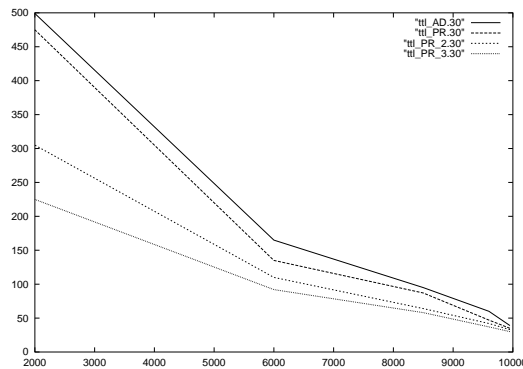


Figure 16: TTL needed to get top 30 PageRank pages

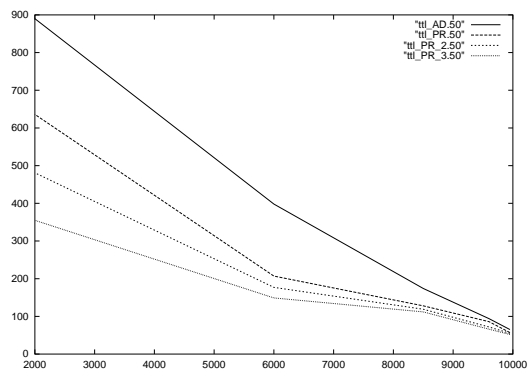


Figure 17: TTL needed to get top 50 PageRank pages

## 4 Trust and Personalized Trust in P2P Networks

Trust and how to model trust has been subject of discussion in different domains like: sociology, economy, justice. In computer domain and especially Internet, it was first used for public key certification and rating in online communities. We are interested to compute trust values in order to minimize unsuccessful trades between peers (in the sense that it implies predictability).

Two main problems must be solved:

- *accuracy* of the computed values
- computation *speed*

We shall first study different trust representation models (4.1). Among various reputational systems (4.1.4), we chose Eigentrust's (4.1.5) point of view for our Algorithm (4.2). The trust propagation values was done using a distributed wisdom-like algorithm (4.1.6). In order to compare the search results with an other system, we used the same network configuration (number of peers, number of pretrusted peers, malicious peers) as in [32]. The malicious attacks scenarios are from the same paper. (experiment results in section: 4.3).

We propose a search algorithm that based on trust and even personalized trust values. This values are computed in a distributed manner, in a way that a peer can obtain its trust value in any other peer in a very short period of time (at run time). The key lies in the fact that a peer only interacts with a small set of peers in the network and will need the trust values only for those.

The notion of personalized trust is based on selecting a personalized set of pre-trusted from the general set of pretrusted peers (hubs).

What is the intuition behind personalized trust values in a P2P network, and, even more importantly, in which way does personalization of trust values improve on global trust rating systems, where only one trust value is established per peer, encompassing all other peers' interactions with this peer?

When a peer in the network selects a subset of pre-trusted peers which it trusts most, it does not necessarily consider these peers as more *trustworthy* than other pre-trusted peers - in fact, all pre-trusted peers should be entirely trustworthy, i.e., always strive to provide authentic file uploads or non- malicious services. Rather, a peer selects those pre-trusted peers whose trust ratings and experiences with peers in the network are most *relevant* to this peer's operations within the P2P network. The peer may operate in a content domain in which certain peers have provided seamless and perfectly trustworthy service - even though they would

have a low global trust rating due to other peers being much more active in responding to popular queries. Hence, personalization of trust values does not only establish a *web of trust*, it creates *personal webs of trust* in which peers with similar interests cooperate to choose the most trustworthy peers in their peer group.

## 4.1 Trust Computation

We can imagine a simple scenario to prove *the need of a reputation system*: in a P2P network a peer is looking for a file. It launches a query to some neighbours who also forward the query to their neighbours and eventually receives several answers from peers who claim to own the file (a peer provides only one answer to a query). It starts downloading and analysing every received file until it finds the expected file (we assume that the search has been done using some key words and that the peer has a mechanism to evaluate the received file and stops after receiving one correct answer).

It is possible that a file provided from a peer in the network is:

1. a file that matches the search criteria, but it is not what the peer wanted
2. the right file, but it is not complete
3. an other file (a malicious peer may answer to all the queries, but always provide the same file)

so, the peer that initiated the query will have to do many file downloads and run the evaluation program a few times until it reaches the expected answer.

These are very *costly* operations for both the network and peer. If the peer defines some trust values based on previous downloads (for example high value for the peers that provided good answers in most of the cases) and use them when choosing the peer to download from, it is very probable that the right file is found quicker (see section 4.3). We will thus *reduce traffic* and *save computational time* for peer (will not have to run the validation program so often).

As discussed above, there can be several approaches for *defining trust* - the value can regard the *quality* of the information contained in the file or the *integrity* of the file. We can define different trust values, for each of this aspects, or we can *aggregate* them by saying like in [32] that a answer is good (the source *trustworthy*) when it provides authentic files (it is the file we are interested in and it is complete) and not trustworthy in all other cases.

In section 4.1.1 we will provide an overview over several different approaches. Section 4.1.2 refers to different mathematical representations and the problems that arise.

It is possible that a peer receives files also from peers from which it never downloaded before. It is necessary to somehow compute trust values for these peers. If not, peers will quickly build a small set of peers to download from and rarely extending this set when these peers do not provide the desired file. But how to compute trust values between two peers that never interacted?

From the *human society model*, if we know that A has trust  $u$  in B and B has trust  $v$  in C, then A should have some trust  $t$  in C that is a function of  $u$  and  $v$ . So, we consider that trust *propagates* from A and B to C. If we could somehow aggregate all these values (see section 4.1.3) - thus building a *web of trust* and compute some global reputation scores, the search in P2P will be much optimized.

How to *aggregate* these trust values depends on how we *represent them mathematically*. This leads us back to the *definition* of trust (how to differentiate between peers that never interacted and those with a low trust value, how to integrate distrust). So, we can say that there are three main problems when dealing with a reputation system:

1. *Definition* of trust
2. How to *quantify* trust
3. How to *propagate* trust values

In order to make the right decision when choosing the reputation system, we studied several approaches concerning trust definition and propagation.

#### **4.1.1 Definition of Trust**

When we say we trust someone, we know, with a large amount of certainty, exactly which aspects of trust we are referring to. There are also instances when we trust one entity more than another. We can talk about different *levels of trust*. To be able to capture this potentially large amount of trust information we have to organize this through different approaches of trust information, from the point of view of a P2P with file sharing.

[1] use trust categories to represent which aspect of trust users are referring, and trust values for the different levels of trust within each category.

In one of the first surveys of the literature on trust and reputation models across diverse disciplines [39], Mui gives definitions to trust and reputation:

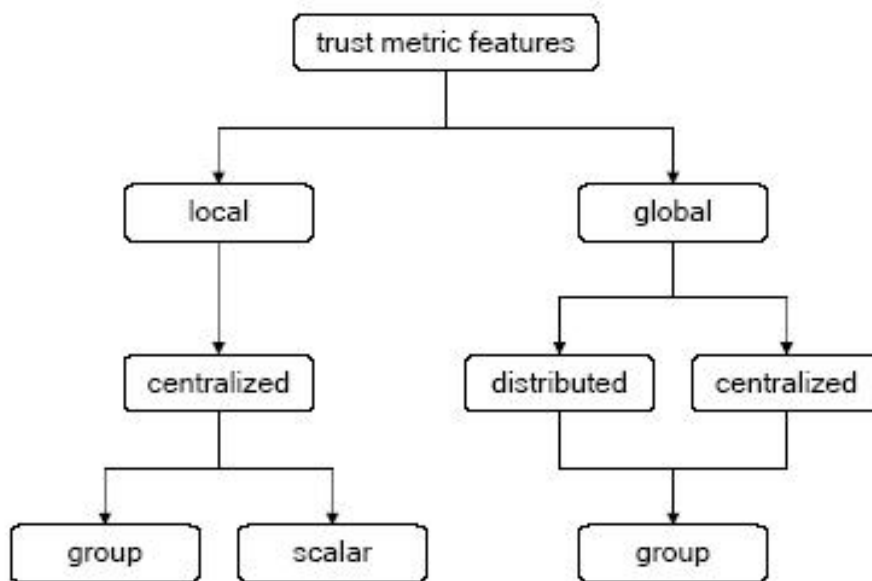


Figure 18: Trust metrics classification

*Trust* - a subjective expectation an agent has about another's future behavior based on the history of their encounters - (a value regarding two agents)

*Reputation* (or *prestige* or *image*)- the perception that an agent creates through past actions about its intentions (a global value)

We notice the clear difference between trust and reputation (or image of a peer in the others eyes).

In a more generalized description, [45] talks about *users* and *statements*, a user had made. From here, two other definitions:

*Beliefs* Any user may assert his personal belief in the statement, which is taken from  $[0,1]$ . A high value means that the statement is accurate, credible, and/or relevant. If user  $i$  one has not provided one, we set it to 0.

*Trust* User  $i$  may specify a personal trust,  $t_{ij}$ , for any user  $j$ . Trust is also a value taken from  $[0, 1]$ , where a high value means that the user is credible, trustworthy, and/or shares similar interests. If unspecified, we set  $t_{ij}$  to be 0.

[53] gives a classification schemes of trust metrics: fig. 18

## 1. Network Perspective

- (a) *Global* trust ranks are assigned to an individual based upon complete trust graph information. Many global trust metrics, such as those presented in [32, 24, 45], borrow their ideas from the renowned PageRank algorithm [41] to compute web page reputation. The basic intuition behind the approach is that nodes should be ranked higher the better the rank of nodes pointing to them. Obviously, latter approach works for trust and page reputation likewise. [25]
- (b) Trust metrics with *local* scope take into account personal bias. The rationale behind local trust metrics is that persons an agent  $a$  trusts may be completely different from the range of individuals that agent  $b$  deems trustworthy. Local trust metrics exploit structural information defined by personalized webs of trust. Hereby, the personal web of trust for an individual  $a$  is given through the set of trust relationships emanating from  $a$ . Merging all webs of trust engenders the global trust graph. An example of local trust metrics is Leviens Advogato trust metric [35].

## 2. Computation Locus

- (a) *Centralized* approaches perform all computations in one single machine and hence need to be granted full access to relevant trust information. There are problems regarding
  - security: one central point of failure
  - ethics: peer  $a$  may find out how it was scored by peer  $b$
- (b) *Distributed* metrics for computation of trust and reputation, such as those described in [32, 45, 46] equally deploy the load of computation on every trust node in the network. Upon receiving trust information from its predecessor nodes in the trust graph, an agent merges the data with its own trust assertions and propagates synthesized values to its successor nodes. The entire process of trust computation is necessarily asynchronous and its convergence depends on the eagerness or laziness of nodes to propagate information.

## 3. Link Evaluation

- (a) According to Levien [34] *group* trust metrics evaluate groups of assertions in tandem, for the reputation of one page depends on the ranks

of referring pages, thus entailing parallel evaluation of relevant nodes thanks to mutual dependencies, like in PageRank algorithm [41]. Advogato [35] represents an example for local group trust metrics.

- (b) *Scalar* metrics analyze trust assertions independently [34], by tracking trust paths from sources to targets and not performing parallel evaluations of groups of trust assertions [25]. So, in general, scalar metrics compute trust between two given individuals  $a$  and  $b$  taken from the set of agents  $V$ .

This are very general points of view. It is important to notice that all the authors agree that trust:

- refers to a *1 to 1* relation
- is *subjective*
- is very *dynamic* (trust values change in time)
- depends by the *context* (local, global, direct or recommended)
- is *not transitive* (if  $a$  trusts  $b$  and  $b$  trusts  $c$ ,  $a$  may not trust  $c$ ) or conditionally transitive
- is *not symmetric* ( $t_{ab}$  need not equal  $t_{ba}$ )

#### 4.1.2 How to Quantify Trust

There are several approaches to quantify trust, beginning from "word-of-mouth" values to real numbers.

Abdul-Rahman talks about different contexts of trust (direct, experience and recommendet) [2] and represent the direct trust using terms from the english language in a 5 level scale. (see table 6)

| Numerical value | Meaning                  |
|-----------------|--------------------------|
| 2               | Very Trustworthy         |
| 1               | Trustworthy              |
| 0               | Moderate Trustworthiness |
| -1              | Untrustworthy            |
| -2              | Very Untrustworthy       |

Table 6: Direct trust

Golbeck [22] uses trust assertions with values ranging from 1 to 9, where 1 denotes complete distrust and 9 absolute trust towards the individual for which the assertion has been issued.

Most of the authors chose a real representation with numbers between 0 and 1, like in [53], which is similar with our approach:

For a set of agents

$$V = \{a_1, \dots, a_n\}$$

there is a partial trust function set:

$$T = W_{a_1}, \dots, W_{a_n}.$$

Every agent  $a$  is associated with one partial trust function  $W_a : V \rightarrow [0, 1]$ , which corresponds to the set of trust assertions that agent  $a$  has stated in other agents.

In most cases, these functions will be very sparse as the number of individuals for which an agent is able to assign explicit trust ratings is much smaller than the total number of agents on the SemanticWeb:

$$W_{a_i}(a_j) = \begin{cases} p & , \text{if } trust(a_i, a_j) = p \in [0, 1] \\ * & , \text{if no rating for } a_j \text{ from } a_i \end{cases}$$

Note that the higher the value of  $W_{a_i}(a_j)$ , the more trustworthy  $a_i$  deems  $a_j$ . Conversely,  $W_{a_i}(a_j) = 0$  means that  $a_i$  considers  $a_j$  to be not trustworthy at all. The value for *no rating* (\*) should be chosen conform to the current policy of the systems regarding new comers: they can receive a higher or a lower value depending of the willing to encourage them to enter or not.

In all these systems, the authors have to point out on:

- the meaning of *0 value* (zero trust and distrust are not the same)
- the value for *not interacted*

In general, we are talking about *trusting* and *not trusting* peers. In a system with trust values between [0,1], we can choose a *threshold* and say that all the values above it mean trust and the other, distrust. We will generally need this only in the final stage of computation.

Some of the authors chose to use values between  $[-1, 1]$  for representing trust. But they only capture the notion of trust and lack of trust, instead of trust and distrust. Explicit modelling of *distrust* [25] has some serious implications for trust metrics (working with negative trust rises both algorithmic and philosophical challenges) and still needs to become subject of intense study.

### 4.1.3 How to Propagate Trust Values

There are several approaches regarding the aggregation of trust values, in many cases particular solutions to some problems, differing on how they defined trust (you can deal only with several values or with an infinity).

There are two main issues when trying to mathematically represent a trust aggregation mechanism:

- *Direction* of propagation
- Mathematical *operator/operation* (add, minimum, etc)

[25] presents four types of *atomic propagation* for trust values. (table 7).

The mathematical representation of these aggregation cases is encoded as the matrix  $M$ : given any matrix  $C$  in which  $C_{ij}$  represents current inferences about  $i$ 's trust in  $j$ , we can replace  $C$  with a new inference matrix  $C \cdot M$  representing one step of direct propagation.

Thus,  $M$  is the operator that encodes the direct propagation basis element.

| Title              | Transformation | Description                               |
|--------------------|----------------|---|
| Direct propagation | $M$            | A trusts B, so trust(A) propagates to B   |
| Co-citation        | $M^T M$        | A trusts B,C, so trust(B) propagates to C |
| Transpose trust    | $M^T$          | A trusts B, so trust(B) propagates to A   |
| Trust coupling     | $M M^T$        | A,B trust C, so trust(A) propagates to B  |

Table 7: Atomic propagation

In fig. 19 we represent direct propagation: if  $i$  trusts  $j$ , and  $j$  trusts  $k$ , then  $i$  trusts  $k$  and co-citation: suppose  $i_1$  trusts  $j_1$  and  $j_2$ , and  $i_2$  trusts  $j_2$ . Under co-citation, we would conclude that  $i_2$  should also trust  $j_1$ .

When dealing with peers and statements, trusts and beliefs [45], the *merging* is done by *concatenating* the trust values along the paths of trust between users and *aggregating* the beliefs.

The *combination* functions may vary from *average* to *noisy OR* and *logistic functions*.

In order to have a well formed merging solution, the concatenation function needs to be *commutative* and *associative* (there are paths with more than one node and it is possible that there are several paths between two nodes) and the aggregating function is *associative* and *distributes* over the first operation.

One example is to use *maximum* and *multiplication*.

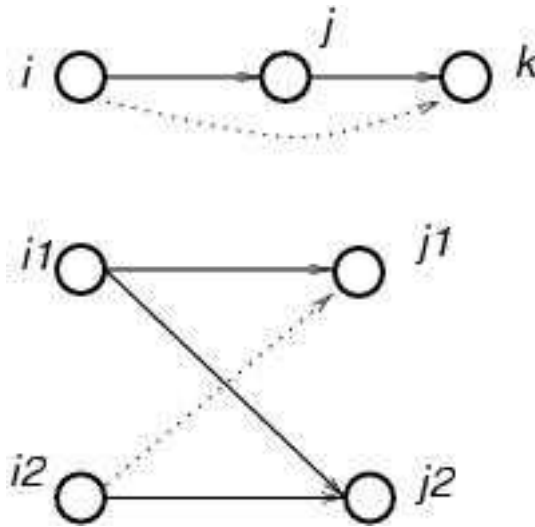


Figure 19: Direct propagation and co-citation

The belief combination function may have a *weak global invariance* - see fig. 20. If a node is removed from the web of trust, and the edges to it are redirected to its trusted nodes (combining the trusts), the merged beliefs of the remaining users remain unchanged.

On the other hand, if we add an arc of trust directly from A to C, and the trust between A and C is unchanged, we say that the belief combination function has *strong global invariance*.

Thus far, we have assumed the graph is *acyclic*.

On *cyclic graphs*, a merging function may have the questionable property that a user may be able to affect others trusts in him by modifying his own personal trusts.

A combination of function is *cycle-indifferent* if it is not affected by introducing a cycle in the path between two users. With cycle indifference, the aggregation over infinite paths will converge, since only the (finite number of) paths without cycles affect its calculation.

However, it is improbable that a web of trust will be acyclic. For example, the Epinions web of trust [45] is highly connected and cyclic.

[45] uses *multiplication* for *concatenation* and compares some possible belief aggregation functions:

*Maximum Value* Using maximum to combine beliefs is consistent with fuzzy

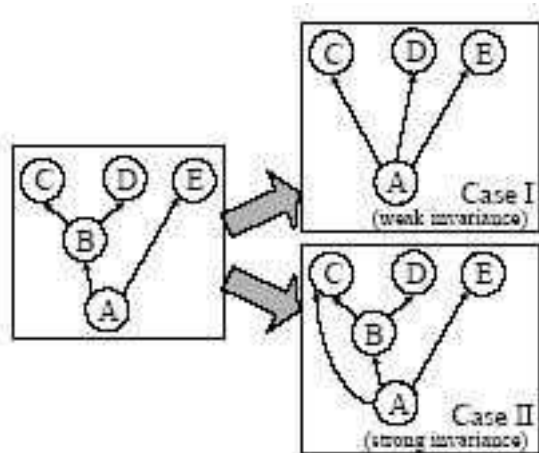


Figure 20: Strong and weak invariance

logic, in which it has been shown to be the most reasonable function for performing a generalized OR operation over  $[0,1]$  valued beliefs. Maximum also has the advantages that it is *cycle-indifferent*, strongly *consistent*, and naturally handles missing values (by letting them be 0). With maximum, the user will believe anything believed by at least one of the users she trusts a reasonable, if not overly optimistic, behavior.

*Minimum Value* Minimum is *not cycle-indifferent*. In fuzzy logic, minimum value is used to perform the *AND operation*. With minimum, the user will only believe a statement if it is believed by all of the users she trusts.

*Average* Average does not satisfy the requirements for a well-formed path algebra outlined above (average is *not associative*). However, average can still be computed by using two aggregation functions: sum and count (count simply returns the number of paths by summing 1s). By passing along these two values, each node can locally compute averages. Average is *not cycle-indifferent*.

We cannot say which choice is the best, in fact, it is hard to define a ranking for a propagation mechanism. We can just compare the results, and concentrate on the specific of the problem.

In our approach, for example, we considered the same values for trust and beliefs and only one propagation function: *multiplication* (logical *AND*).

**Probabilistic interpretation** Imagine a *random knowledge-surfer* hopping from user to user in search of beliefs. At each step, the surfer probabilistically selects

a neighbor to jump to according to the current users distribution of trusts. Then, with probability equal to the current users belief, it says *yes, I believe in the statement*. Otherwise, it says *no*.

#### 4.1.4 Previous Reputation Systems

In only a few years many reputational systems were developed, each solving some of the puzzles regarding a trust management in a computational system. We developed our own system in order to improve search in a P2P file sharing oriented network.

[53]s main contribution is *Appleseed*, a fixed-point personalized trust algorithm inspired by spreading activation models. An important aspect for us is the introduction of "backward trust propagation" (i.e. virtual edges from every visited node  $x$  to the computation seed node  $s$ ), which solves several rank normalization problems (e.g. distinguish between a peer with whom some peer  $i$  did not interact and a peer with whom  $i$  had bad experiences). We intend to investigate the integration of this approach into our algorithm.

[45] builds a web of trust in a way similar to ours, with each user having to maintain trust values on a small number of other users. The algorithm presented is designed for an application within the context of the Semantic Web, composed of logical assertions. This helps introducing personalization, as each user would have a level of local belief in statements and a level of trust in other users, all of which could then be merged to reflect a global meaning.

[36] presents a similar, but simpler method than ours: rather than computing a fixed-point algorithm, a quorum of other peers is asked about their opinion on some peer  $p$ . The advantage is the more reduced network traffic due to the computation of required trust values, yet this comes at the cost of not achieving a global perspective on the trustworthiness of peer  $p$ . A related approach is also presented in [22], where the FOAF [16] schema is extended to contain trust assertions between peers. The inferred rating from a source peer to a sink one is (recursively) computed using the weighted average of the neighbors' reputation ratings of the sink.

Just as [30] improves [41], our algorithm extends the capabilities of [32]. The latter is a fixed-point PageRank-like distributed computation of reputation values in a P2P network. We used its model in designing our algorithm, but also the investigations about possible attacks from malicious peers. Finally, [25] introduces the notion of "distrust" and analyses several trust propagation approaches, as well

as the rounding of trust values to boolean decisions.

In all cases, input data can be represented as a directed graph with the targets of the algorithm as nodes. For ranking web pages or documents, graph edges are the hyperlinks, whereas for building reputation values they resemble peers' experiences with other peers. These approaches are summarized in table 8.

| Outcome   | Node in graph      | Edge in graph                   |
|---|--------------------|---------------------------------|
| Web page ranks [41]                                       | Web page           | Hyperlink                       |
| Personalized Web page ranks [30]                          | Web page           | Hyperlink                       |
| Document ranks (distributed) [46, 51]                     | Document on a peer | Hyperlink between two documents |
| Personalized document ranks (distributed) [42]            | Document on a peer | Hyperlink between two documents |
| Reputation values (distributed) [32]                      | Peer               | Download experience of peers    |
| Personalized reputation values (distributed) - this paper | Peer               | Download experience of peers    |

Table 8: Hyperlink structures used by diferent ranking algorithms

#### 4.1.5 Our Algorithm

We chose the Eigentrust algorithm [32] as model for our reputational system. We will present below the assumption that we made.

Eigentrust is an algorithm developed at Stanford in 2003 for search in P2P Networks in order to decrease the number of downloads of *inauthentic files* (up to 70% in some cases).

Inauthentic files can be obtained when downloading from a source wich

- has a bad connection to the internet or a small bandwidth or
- deliberately provide inauthentic files

This peers should be avoided in the future when selectind as download source. It is easy for a peer to count down authentic and inauthentic downloads for a small number of peers.

Generaly speaking, attempting to identify malicious peers that provide inauthentic files is superior to attempting to identify inauthentic files themselves, since *malicious peers* can easily generate a virtually unlimited number of inauthentic files if they are not banned from participating in the network.

**A definition for trust** In Eigentrust, the global reputation of each peer  $i$  is given by the local trust values assigned to peer  $i$  by other peers, weighted by the global reputations of the assigning peers.

In a distributed environment, peers may still rate each other after each transaction, as in the eBay system.

For example, each time peer  $i$  downloads a file from peer  $j$ , it may rate the transaction as positive ( $tr_{i,j} = 1$ ) or negative ( $tr_{i,j} = -1$ ). Peer  $i$  may rate a download as negative, for example, if the file downloaded is inauthentic or if the download was interrupted.

Like in the eBay model [18], we may define a local trust value  $s_{i,j}$  as the sum of the ratings of the individual transactions that peer  $i$  has downloaded from peer  $j$ :  $s_{i,j} = \sum tr_{i,j}$ .

Equivalently, each peer  $i$  can store the number satisfactory transactions it has had with peer  $j$ ,  $sat(i, j)$  and the number of unsatisfactory transactions it has had with peer  $j$ ,  $unsat(i, j)$ . Then,  $s_{i,j}$  is defined:  $s_{i,j} = sat(i, j) - unsat(i, j)$ .

It is necessary to *normalize* these values in some manner. Otherwise, malicious peers can assign arbitrarily high local trust values to other malicious peers, and arbitrarily low local trust values to good peers, trust can be computed like this:

$$c_{ij} = \frac{\max(s_{i,j}, 0)}{\sum_j \max(s_{i,j}, 0)}$$

This ensures that all values will be between 0 and 1.

Notice that if  $\sum_j \max(s_{i,j}) = 0$ , then  $c_{ij}$  is undefined. We will choose a set of pretrusted peers with some trust values, so that every peer trusts at least some peers.

There are some drawbacks to normalizing in this manner: for one, the normalized trust values do not distinguish between a peer with whom peer  $i$  did not interact and a peer with whom peer  $i$  has had poor experience. Also, these  $c_{ij}$  values are relative, and there is no absolute interpretation.

That is, if  $c_{ij} = c_{ik}$ , we know that peer  $j$  has the same reputation as peer  $k$  in the eyes of peer  $i$ , but we don't know if both of them are very reputable, or if both of them are mediocre.

However, we are still able to achieve substantially good results despite the drawbacks mentioned above (see section 4.3).

We choose to normalize the local trust values in this manner because it allows us to perform the computation that we describe below, without renormalizing the global trust values at each iteration (which is prohibitively costly in a large distributed environment).

**Aggregating local trust values** We wish to aggregate the normalized local trust values. A natural way to do this in a distributed environment is for peer  $i$  to ask its acquaintances about their opinions about other peers. It would make sense to weight their opinions by the trust peer  $i$  places in them.

This is a useful way to have each peer gain a view of the network that is wider than his own experience. However, the trust values stored by peer  $i$  still reflects only the experience of peer  $i$  and his acquaintances.

In order to get a wider view, peer  $i$  may wish to ask his friends' friends ( $t = (C^T)^2 c_i$ ). If he continues in this manner, ( $t = (C^T)^n c_i$ ), he will have a complete view of the network after some iterations (under the assumptions that  $C$  is irreducible and aperiodic).

Fortunately, if  $n$  is large, the trust vector  $t_i$  will converge to the same vector for every peer  $i$ . Namely, it will converge to the left principal eigenvector of  $C$ . (please refer to [32] for the proofs)

For computing the final values, we chose the *widom algorithm* (see section 3.1.2) which scales well with the size of the network and number of hubs.

We created a version for trust computation and distributed it. We will present in the next section the complete Algorithm and the proofs for convergence.

#### 4.1.6 Distributed Widom Algorithm for Trust

We divide the algorithm in three parts, as presented in section 3.1.3:

- one part focuses mostly on peers outside the set of pre-trusted peers,  $V \setminus H$  (Algorithms 3.1.1 and 3.1.2),
- one on peers within the set of pre-trusted peers  $H$  (Algorithm 3.2),
- the final algorithmic step ties everything together (Algorithm 3.3).

The tranfered data between peers is the same as for the Page Rank computation algorithm (see section 3.1.4).

**Partial Vectors** This part of the algorithm consists of one special initialization step and several succeeding steps. Even though its focus is on peers from  $V \setminus H$ , peers  $p \in H$  also gather their components of  $\mathbf{D}$  and  $\mathbf{E}$ .

All peers normalize their trust values as

$$\gamma_q(i) = \frac{\gamma_q(i)}{\sum_i \gamma_q(i)}$$

In the first step, each peer  $q \in V$  computes  $E[p](q)$ .

As peers  $p \in H$  are known, each peer  $q \in V$  can set its initial values  $D_0[p](q)$  and  $E_0[p](q)$  by itself to:

$$D_0[p](q) = \begin{cases} c & , q \in H \\ 0 & , otherwise \end{cases} ; E_0[p](q) = T_0[p](q) = \begin{cases} 1 & , q \in H \\ 0 & , otherwise \end{cases} \quad (5)$$

**Non-hub peers.** After this initialization, the following operations will be executed in parallel by each peer  $q \in V \setminus H$  for each  $p \in H$ :

---

**Algorithm 3.1.1.** Distributed computation of partial vectors by peers in  $V \setminus H$ .

---

- 1: Send  $\frac{1-c}{|\mathbf{O}(q)|} \cdot T_k[p](q) \cdot \gamma_q(i)$  to all peers  $i$  from which  $q$  has downloaded files, including those from  $H$ .
  - 2: Wait from all peers which downloaded files from  $p$  their  $T_k[p](*)$  (at this step  $k$ )
  - 3: After all  $T_k[p](*)$  values have been received, compute  $T_{k+1}[p](q)$  as:  

$$T_{k+1}[p](q) = \sum_{v \in \mathbf{I}(q)} T_k[p](v)$$
  - 4: Compute:  

$$D_{k+1}[p](q) = D_k[p](q) + c \cdot T_k[p](q)$$

$$N$$
 being the number of steps we apply the Selective Expansion algorithm.
  - 5: If there are more iterations left, go to 1
  - 6: Each peer  $q \in V \setminus H$  computes  $(r_p - r_p^H)(q) = D_N[p](q) + c \cdot T_N[p](q)$ , its component of the partial vector corresponding to  $p$ .
- 

In the distributed version of the Selective Expansion algorithm, for  $p \in H$  each peer  $q \in V \setminus H$  has to compute:

$$T_{k+1}[p](q) = \sum_{v \in \mathbf{I}(q)} \frac{1-c}{|\mathbf{O}(v)|} \cdot E_k[p](v) \cdot \gamma_v(q) \quad (6)$$

Due to space limitations, we refer the reader to [42] for the proof of this theorem.

We will only present here the proof for the convergence when using trust ( $\gamma$ ) values:

**Proof**

We will proof that for all  $k \geq 0$  and for all  $p \in V$

$$\mathbf{D}_k[\mathbf{p}] + \sum_{q \in V} E_k[p](q) \mathbf{r}_q = \mathbf{r}_p \quad (7)$$

and that the sequence  $\mathbf{E}_k[\mathbf{p}]$  converges to 0 as  $k$  tends to infinity.

The proof is by induction on  $k$ .

The case for  $k=0$  is obvious, so suppose that claim is true for  $k$ , for some  $k \geq 0$ .

We will show that

$$\mathbf{D}_{k+1}[\mathbf{p}] + \sum_{q \in V} E_{k+1}[p](q) \mathbf{r}_q = \mathbf{r}_p \quad (8)$$

for an arbitrary  $Q_k(p) \in V$ .

$$\begin{aligned} & \mathbf{D}_{k+1}[\mathbf{p}] + \sum_{q \in V} E_{k+1}[p](q) \mathbf{r}_q = \left( \mathbf{D}_k[\mathbf{p}] + \sum_{q \in Q_k(p)} c \cdot E_k[p](q) \mathbf{x}_q \right) + \\ & + \sum_{q' \in V} \left( \mathbf{E}_k[\mathbf{p}] - \sum_{q \in Q_k(p)} E_k[p](q) \mathbf{x}_q + \sum_{q \in Q_k(p)} \frac{1-c}{|\mathbf{O}(q)|} \sum_{i=1}^{|\mathbf{O}(q)|} E_k[p](q) \mathbf{x}_{O_i(q)} \cdot \gamma_{O_i(q)}(q) \right) (q') \mathbf{r}_{q'} = \\ & = \left( \mathbf{D}_k[\mathbf{p}] + \sum_{q' \in V} E_k[p](q') \mathbf{r}_{q'} \right) + \sum_{q \in Q_k(p)} c \cdot E_k[p](q) \mathbf{x}_q - \sum_{q' \in V} \sum_{q \in Q_k(p)} E_k[p](q) x_q(q') \mathbf{r}_{q'} + \\ & + \sum_{q' \in V} \sum_{q \in Q_k(p)} \frac{1-c}{|\mathbf{O}(q)|} \sum_{i=1}^{|\mathbf{O}(q)|} E_k[p](q) (x_{O_i(q)}(q') \cdot \gamma_{O_i(q)}(q)) \mathbf{r}_{q'} \end{aligned}$$

By the inductive hypothesis,

$$\mathbf{D}_k[\mathbf{p}] + \sum_{q' \in V} E_k[p](q') \mathbf{r}_{q'} = \mathbf{r}_p \quad (9)$$

so we need only to show that the latter terms cancel.

Since  $x_q(q') = 1$  if  $q = q'$  and 0 otherwise, and similarly for  $x_{O_i(q)}(q')$  and  $\gamma_{O_i(q)}(q')$ , we have

$$\sum_{q' \in V} \sum_{q \in Q_k(p)} E_k[p](q) x_q(q') \mathbf{r}_{q'} = \sum_{q \in Q_k(p)} E_k[p](q) \mathbf{r}_q \quad (10)$$

and

$$\sum_{q' \in V} \sum_{q \in Q_k(p)} \frac{1-c}{|\mathbf{O}(\mathbf{q})|} \sum_{i=1}^{|\mathbf{O}(\mathbf{q})|} E_k[p](q) x_{O_i(q)} \cdot \gamma_{O_i(q)}(q') (q') \mathbf{r}_{q'} = \sum_{q \in Q_k(p)} \frac{1-c}{|\mathbf{O}(\mathbf{q})|} \sum_{i=1}^{|\mathbf{O}(\mathbf{q})|} E_k[p](q) r_{O_i(q)} \quad (11)$$

By the Decomposition Theorem,

$$c \cdot E_k[p](q) \mathbf{x}_q + \frac{1-c}{|\mathbf{O}(\mathbf{q})|} \sum_{i=1}^{|\mathbf{O}(\mathbf{q})|} E_k[p](q) \mathbf{r}_{O_i(q)} = E_k[p](q) \mathbf{r}_q \quad (12)$$

for all  $q \in Q_k(p)$ , which shows that the terms cancel.

**Hub peers.** In the special first step, a peer  $p \in H$  will send the value

$$\frac{1-c}{|\mathbf{O}(\mathbf{q})|} \cdot T_k[p](p) \cdot \gamma_p(i)$$

to all peers  $i$  from which it has downloaded files, including those from  $H$ .

After that, it will execute the following operations:

---

**Algorithm 3.1.2.** Distributed computation of partial vectors by peers in  $H$ .

---

- 1: Wait from all peers which downloaded files from  $p$  their  $T_k[p](*)$  (at this step  $k$ )
  - 2: After all  $T_k[p](*)$  values have been received, do  $T_{k+1}[p](p) = \sum_{v \in \mathbf{I}(\mathbf{q})} T_k[p](v)$
  - 3: If there are more iterations left, go to 1
  - 4: Compute:  $E[p](p) = \sum_{k=1}^N T_k[p](p)$
  - 5: Set  $D_N[p](p)$  to  $c$
  - 6: Each peer  $p \in H$  computes  $(r_p - r_p^H)(p)$ , its component of its partial vector.
-

Algorithms 3.1.1 and 3.1.2 perform a power-iteration, and they would therefore converge also in the presence of loops in  $G$  (see [30, 41] for details), whereas the high level of dynamics of a P2P network would only result in some additional iterations until convergence.

**Hub Skeleton** In the second phase of the algorithm, each peer  $p \in H$  (pre-trusted, or hub peer) has to calculate its hub skeleton ( $\mathbf{r}_p(\mathbf{H})$ ) using as input the results from the previous stage.

The result is stored in the values  $\mathbf{D}_{2k}[\mathbf{p}]$  obtained after the last iteration of the following operations, performed by each  $p \in H$ :

---

**Algorithm 3.2.** Distributed computation of hub skeleton (only in  $H$ ).

---

- 1: Calculate  $\mathbf{D}_{2k}[\mathbf{p}]$  and  $\mathbf{E}_{2k}[\mathbf{p}]$ , using the centralized version formulas
  - 2: Multicast the results to all other peers  $q \in H$
  - 3: If there are more iterations left, go to 1.
  - 4: Every hub peer broadcasts its  $\mathbf{D}_{2N}[\mathbf{p}]$  sub-vector (only the components regarding pages from  $H$ ).
- 

As this step refers to hub-peers only, the computation of  $\mathbf{D}_{2k}[\mathbf{p}]$  and  $\mathbf{E}_{2k}[\mathbf{p}]$  can consider *only* the components regarding pages from  $H$ .

**PPV** As the  $\mathbf{D}_{2N}[\mathbf{p}]$  sub-vectors ( $p \in H$ ) have been broadcast, *any* peer  $v \in V$  can now determine its  $\mathbf{r}_v(\mathbf{P})$  locally.

*Any* peer  $v \in V$  can also calculate its partial PPV containing its reputation and the reputation of any other peers from its own point of view. If we denote  $NB$  the set of peers whose rank  $v$  wants to compute, it must do the following:

---

**Algorithm 3.3.** Computation of the Personalized Reputation Vector.

---

- 1: Request the components of  $\mathbf{r}_p - \mathbf{r}_p^H$  for all  $p \in H$  from all peers from  $NB$ .
  - 2: Compute the components of the PPV.
- 

Of course, if later  $v$  wants to compute the reputation value of another peer, it would only need to ask the new peer about its components of  $\mathbf{r}_p - \mathbf{r}_p^H$ .

## 4.2 Application: Search Algorithm Using Trust Values

We developed and implemented an algorithm for search in P2P using trust values (4.2.2). We tested our search algorithm in the same conditions as in [32], we mean by that distribution of peers and different malicious threats (4.2.1).

Like in this paper, we simulated some *malicious attacks* (4.2.3) and we run the same type of experiments (4.3).

### 4.2.1 Background and Previous Work

We described some reputational systems in (4.1.4). We are interested on how robust such a system can be. The authors of [32] made a research on Edutella and presented a list of some of the most known malicious attacks in P2P networks and the results obtained when using Eigentrust.

#### A) Individual Malicious Peers

- *malicious peers*
  - always provide inauthentic files when selected as source for download
  - set their local trust values to be  $s_{i,j} = \text{authentic\_download}(i, j) - \text{inauthentic\_download}(i, j)$
- *results*
  - inauthentic file download reduced to 10% (even by a rate of 70% malicious peers)

#### B) Malicious Collectives

- *malicious peers*
  - always provide inauthentic files when selected as source for download
  - form malicious collective by assigning a single trust value of 1 to an other malicious peer in the network
- *results*
  - inauthentic file download reduced to 10
  - the system breaks up malicious collectives by the presence of pretrusted peers

### *C) Malicious Collectives with Camouflage*

- *malicious peers*
  - provide an inauthentic files in  $f\%$  of all cases
  - form malicious collective by assigning a single trust value of 1 to an other malicious peer in the network
- *results*
  - they have maximum impact when providing in 50% of cases authentic files (up to 20% inauthentic files downloaded/ total nr of downloads)

### *D) Malicious Spies*

- *malicious peers*
  - answer 0.05% of the most popular queries
  - provide a good response when selected as download source
  - assign trust values of 1 to all malicious files of type 1 in the network
  - they will gain trust value and contribute on increasing of typeB peers trust values
  - reduces the ammount of inauthentic files downloaded

### *E) Sybil Attack*

- *malicious peers*
  - initiates thousands of peers in the network wich provides an inauthentic file and then disconnects, entering the network again under an other name
  - will prevent good peers to gain some reputation and they are also not interested in gaining good reputation Such an attack can be avoided by imposing a cost in enteing the network, or passing a test to prove that it is no automated login

### *F) Virus Disseminators*

- *malicious peers*
  - sends a virus every 100th request
- such an attck can be avoided by other methods

### 4.2.2 Our Algorithm - Detailed View

The algorithm is a generalization of the search algorithm presented in (3.2) in a *reputational context*.

In this section,

- *peer* stands for a station in the network - a peer may own several files
- *neighbours* of a peer are the peers to whom a peer first connect and then all the peers from whom it downloads file in a certain amount of time.
- *time* represents the interval from launching a query until it is done (TTL expires or the desired number of responses were received)

The queries are forwarded from a peer to an other like this: a peer sends the query to its high trusted neighbour. Every time a peer forwards a query, it checks to see if that neighbour was not visited and if it was, it sends the query to the neighbour with the second trust value and so on. If all the neighbours were visited, it sends the query back.

In our experiments, there were very few cases of a query "returning" to the originating peer. It only happened in small and not very connected networks - under 1000 nodes).

OBS: from the set of neighbours of a peer, the peer to forward the query to, were not selected strictly by their trust value, but with a probability proportional to their trust value, in order to avoid *overloading* some high trusted peers.

We will *aggregate* the trust and belief in one trust value, but we will deal with two trust values: *direct* and *indirect*.

The *direct* one results from history of files exchanges (peers download files one from another and count the authentic and inauthentic downloads, each peer will compute the trust values to the peers it interacted with. This are values between 0 and 1 and the sum of all trust values expressed by a peer must be normalized.

The *indirect* values are obtained by aggregating the *direct* values via *trust paths*.

The activity in the network is divided into *cycles*. At the beginning of the cycle, the direct trust values are aggregated over the network, so the new trust values can be computed. A peer memorizes a set of trust values - at the beginning, only for its neighbours - and uses this values when forwarding a query. So, the

answers will come at the beginning from its high trusted neighbours. If the close neighbours do not provide the right answer, the peer will download from the peers trusted by its high trusted neighbours and so on. The new download sources will be added to the set of trusted peers - with their corresponding trust value. One cycle contains a limited number of queries. After that, new trust values are again propagated and so on.

We imagine an environment more closer to the real world by introduction of different states of a peer: at a certain moment (issued query) a peer may be up or down. If it is down, it will neither answer the query, nor forward it. Peers may provide authentic or inauthentic files.

We also introduce some malicious peers, imagine malicious attack scenarios and study the robustness of our system.

The system consists of several parts:

- *Personalized Trust* - the program that computes the personalized trust or just trust using the neighbours file and the direct trust values
  - reads the Out and In neighbours, hubs and preferenced hubs (different values for different peers).
  - reads the direct trust values from the trust file (a value for each neighbour of each peer)
  - simulates the distributed computation of trust values (the aggregation of trust values through the network and the final computations for both global trust and personalized trust values)
  - writes the new trust values in files (indirect trust values)
- *Search Simulator* - the program that simulates the P2P environment (file distribution over the peers, peers state, peers type - malicious or not, allows query cycles)
  - reads the peers neighbours and the trust values for each neighbour
  - allows a number of queries to pass through the network
  - during the query cycles, each peer memorizes the number of authentic and inauthentic file downloads from all the peers that answered
  - computes the new trust values - direct values - each peer  $i$  computes a value for each neighbour  $j$  based on the download history:

$$s_{i,j} = \text{authentic\_download}(i, j) - \text{inauthentic\_download}(i, j)$$

$$\text{trust}_{i,j} = \frac{\max(s_{i,j}, 0)}{\sum_j \max(s_{i,j}, 0)}$$

– saves the direct trust values and the new neighbours  $j$

- *Scripts for network configuration* - model different malicious attacks.

### 4.2.3 Malicious Threats

From a security point of view, the algorithm as described above, contains two critical components. First, pre-trusted peers play an important role in computing trust values. Hence an implementation of the algorithm has to make sure that they behave correctly. Fortunately, only very few pre-trusted peers are required in a network (see also section 4.3) such that the administrative task of ensuring this behavior will present little overhead. Second, non-pre-trusted peers calculate trust values for themselves, or at least contribute to their calculation. This gives each peer major power over his own trust rating.

A secure version of the algorithm thus has to ensure that malicious peers in the network have limited or no impact on the computation of their own trust ratings.

- *Threat Model A.* Malicious peers always provide an inauthentic file when selected as download source. They set their local trust values to  $1 - s_{i,j}$ , i.e. the opposite of their real trust value.
- *Threat Model B.* Malicious peers of type A, but forming a malicious collective: they set their local trust values to  $1 - s_{i,j}$  for good peers, and to 1 for any other malicious peers
- *Threat Model C.* Malicious peers of type B, but provide an inauthentic file in  $f\%$  of all cases when selected as download source ( $f$  very low), in order to gain trust.
- *Threat Model D.* A first group of malicious peers of type D provide authentic files with 95% probability (just as the good peers), but additionally assign a trust of 1 to all malicious peers of the second type, B.

### 4.3 Experiments and Results

**Robustness.** A very important aspect of a reputation algorithm is to be robust against attacks of malicious peers, which try to subvert the network by uploading inauthentic files. For these tests, we implemented the possible attacks of malicious peers presented in [32]. As our algorithm is also based on a power iteration, the percentages of inauthentic files malicious peers are able to upload in the presence of the reputation system should be similar to those from [32] (we target the production of results more tailored to user’s preferences, not a better resistance against attacks, as these are very well handled by the power iteration already).

In all experiments, we considered the network to have an initial structure, i.e. peers have some initial trusts in other peers, generated randomly following a power law distribution. We ran 30 query cycles, each of them consisting of sending 50 queries through the network. After each query cycle one more iteration of the selective expansion took place, as well as an update of the reputation vectors at each peer. The results of repeated squaring were updated only once in 5 cycles, as they need more computation resources.

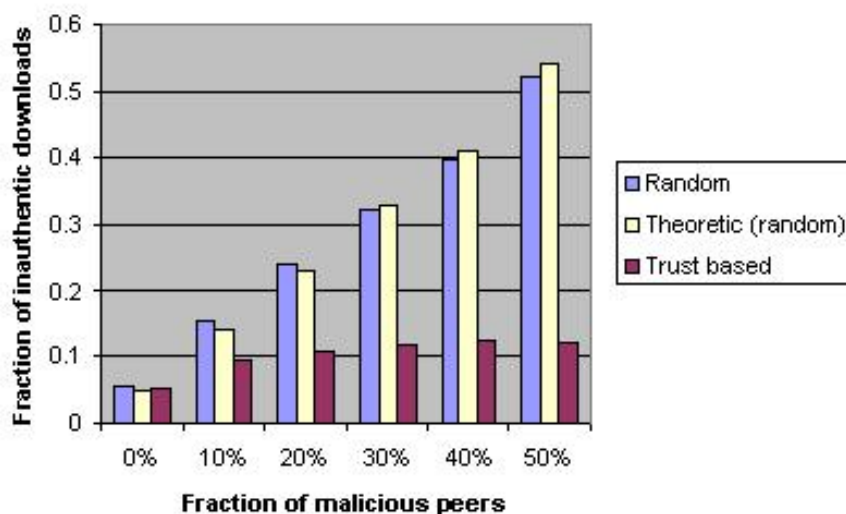


Figure 21: Threat Model A: Malicious peers provide inauthentic files and set their local trust opposite to the real values.

For the tests, we used three Threat Models like those described in Eigentrust [32]:

*Threat Model A.* Malicious peers always provide an inauthentic file when se-

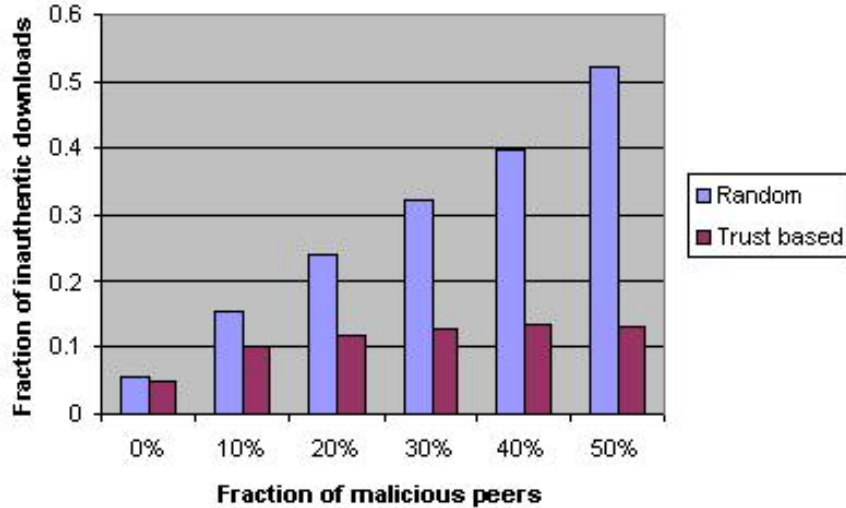


Figure 22: Threat Model B: Malicious peers of type A also form a collective.

lected as download source. They set their local trust values to  $1 - s_{i,j}$ , i.e. the opposite of their real trust value. The network consists of 63 good peers and 0, 7, 14, 25, 37, and 60 malicious peers respectively. Each good peer answers a corrupt file with 5% probability.

*Threat Model B.* Malicious peers of type A form a malicious collective. They set their local trust values to  $1 - s_{i,j}$  for good peers, and to 1 for any other malicious peers (i.e. complete trust in the other malicious peers). The peer distribution has the same characteristics as in threat model A.

*Discussion.* In both cases, the average fraction of inauthentic downloads is about 11% and does not exceed 13.5%, which represents a significant decrease against a network without a reputation system in place. Moreover, malicious collectives are broken by our algorithm, as they only add 1-2% extra inauthentic downloads.

*Threat Model C.* Malicious peers of type B provide an inauthentic file in  $f\%$  of all cases when selected as download source, with an  $f$  varying from 0 to 100 in steps of 10. The network consists of 53 good peers and 20 malicious ones.

*Threat Model D.* A first group of malicious peers of type D provide authentic files with 95% probability (just as the good peers), but additionally assign a trust

of 1 to all malicious peers of the second type, B. There are 63 good peers and 40 malicious ones, the latter ones having different roles in different experiments, as depicted in figure 24.

*Discussion.* Here, malicious peers try to increase their rating by also providing good files. In model C, the maximum amount of inauthentic files they insert in the network is 17.6%, achieved when providing 50% good answers. For  $f=70\%$  (e.g. 365 good answers and 1215 bad ones), there were only 7.3% corrupt downloads in the entire network. Finally, the increase of bad downloads is also small when some peers acting correctly are trying to boost the reputation of the malicious ones. Although results in the right side of figure 24 are comparable to the random case, they require too much effort from malicious peers. For example, with 15 B peers and 25 D peers, they need to upload 1420 good files altogether in order to distribute 1197 inauthentic ones.

Generally, we can conclude that the robustness of our algorithm against malicious peers is very similar to that of EigenTrust [32] and thus the addition of personalization into a fixed-point reputation algorithm does not make it more vulnerable.

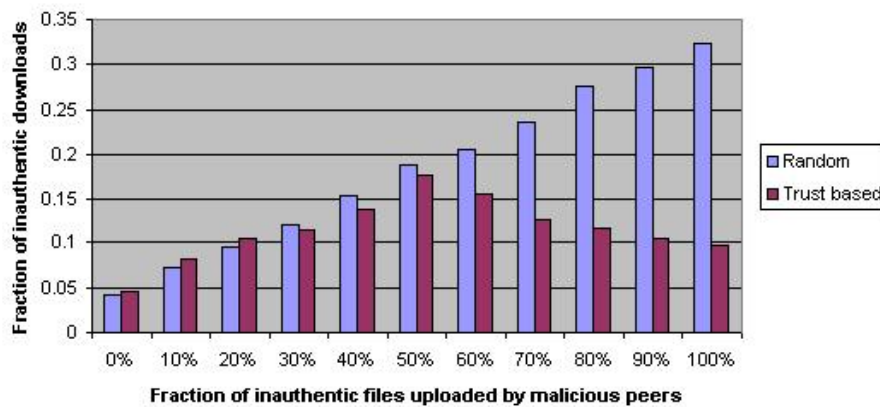


Figure 23: Threat Model C: Malicious collective providing  $f\%$  correct answers.

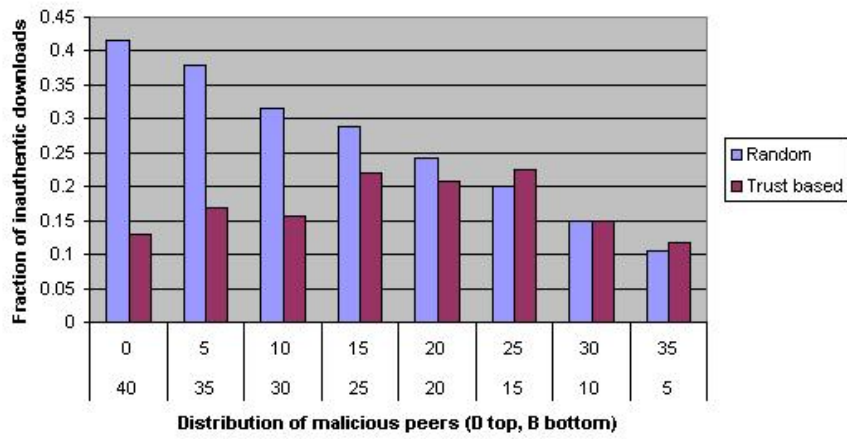


Figure 24: Threat Model D: Malicious group boosting the reputation of a malicious collective of type B.

## 5 Conclusions and Further Work

We implemented and presented in this paper algorithms to distributedly compute *personalized page ranks for files* in P2P networks and for computing *trust values in peers* and the corresponding *search* simulators. The sources can be found at [www.l3s.de/~scurtu/Diplo/Sources](http://www.l3s.de/~scurtu/Diplo/Sources)

The algorithms *scale* well with the graph dimension (they depend only on the number of hubs and the number of neighbours a peer has).

We showed in our experiments that search is improved by adding a *personalization* component.

As in *personalized page rank*, the *personalized trust* value stands for similarity. We though derived from the notion of *web of trust* the one of *personalized web of trust* with the meaning that peers with similar interests cooperate more via the *trust paths*.

In order to obtaine reliable results, we created a *WebSimulator*. The search algorithms model a real P2P environment, with peers being up or down, with presence of inauthentic files and even malicious attacks.

We managed to adress some of the main problems in a P2P network:

- The system has to be *self-policing*.
- The system has to *maintain anonymity*.
- The system should have *minimal overhead* in terms of computation, infrastructure, storage, and message complexity.
- The system should be *robust* to malicious collectives

Some issues are still left open, mostly in concerning *security* issues. It is not safe to have the peers computing their own values.

Solutions can be found by:

- having peers computing values one for another
- adding redundancy to the calculations

How will this solutions work and interfere with the above enumerated issues will be the subject of further work.

## References

- [1] Alfarez Abdul-Rahman and Stephen Hailes. A distributed trust model.
- [2] Alfarez Abdul-Rahman and Stephen Hailes. Learning to rely on others opinions on your own. In *Invited seminar, Computer Lab (Com Sci Dept), University of Cambridge*.
- [3] L.A. Adamic, R. Lukose, A. Puniyani, and B. Huberman. Search in power-law networks. *Phys. Rev.*, E 64, 046135, 2001.
- [4] Barabasi Albert. *Linked. How Everything is Connected to Everything Else and What it Means for Science, Business and Everyday Life*. Perseus Publishing, 2002.
- [5] Amazon on-line market [www.amazon.com](http://www.amazon.com).
- [6] Corin Anderson. A machine learning approach to web personalization, 2002.
- [7] Arvind Arasu, Junghoo Cho, Hector Garcia-Molina, Andreas Paepcke, and Sriram Raghavan. Searching the web. Technical report, Stanford Digital Library Technologies Project, 2000.
- [8] Albert Reka Barabasi Albert and Jeong Hawoong. Diameter of the world-wide web. *Nature*, 401.
- [9] K. Bharat and G. Mihaila. When experts agree: Using non-affiliated experts to rank popular topics. In *In Proceedings of the WWW2001 Conference, Hong Kong*, 2001.
- [10] Krishna Bharat and Monika Henzinger. Improved algorithms for topic distillation in a hyperlinked environment. In *In Proceedings of the ACM SIGIR Conference, New Orleans, Louisiana, USA, September*, 1998.
- [11] 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.
- [12] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, and R. Stata. Graph structure in the web. In *In Proceedings of the 9th International World Wide Web Conference*, 2000.
- [13] Andrei Broder. A taxonomy of web search. Technical report, IBM Research, 2002.

- [14] Steve Chien, Cynthia Dwork, Ravi Kumar, and D. Sivakumar. Towards exploiting link evolution, 2001.
- [15] Paul-Alexandru Chirita, Daniel Olmedilla, and Wolfgang Nejdl. Pros: A personalized ranking platform for web search. In *Proceedings of the International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems*, Aug 2004.
- [16] E. Dumbill. Finding friends with xml and rdf.
- [17] Growth dynamics of the World-Wide Web. Bernardo a. huberman and lada a. adamic. *Nature*, 401.
- [18] Ebay <http://www.ebay.com>.
- [19] Epinions: [www.epinions.com](http://www.epinions.com).
- [20] The freenet project: <http://freenet.sourceforge.net/>.
- [21] Gnutella web page: <http://www.gnutella.com/>.
- [22] J. Golbeck, B. Parsia, and J. Hendler. Trust networks on the semantic web. In *Proceedings of Cooperative Intelligent Agents*, 2003.
- [23] Google search engine. <http://www.google.com>.
- [24] R. Guha. Open rating systems, 2004.
- [25] R. Guha, R. Kumar, P. Raghavan, and A. Tomkins. Propagation of trust and distrust. In *Proceedings of the 13th International WWW Conference (WWW 2004)*, 2004.
- [26] Jean-Loup Guillaume and Matthieu Latapy. The web graph: an overview.
- [27] T. Haveliwala. Efficient computation of pagerank. Technical report, Stanford University, 1999.
- [28] T. Haveliwala. Topic-sensitive pagerank. In *In Proceedings of the Eleventh International World Wide Web Conference, Honolulu, Hawaii*, May 2002.
- [29] I.~Stoica, R.~Morris, D.~Karger, M.~F.~Kaashoek, and H.~Balakrishnan. Chord: A scalable peer-to-peer lookup lervice for internet applications. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, San Diego, USA, August 2001.

- [30] G. Jeh and J. Widom. Simrank: A measure of structural-context similarity. In *Proceedings of the 8th ACM International Conference on Knowledge Discovery and Data Mining*, 2002.
- [31] G. Jeh and J. Widom. Scaling personalized web search. In *Proceedings of the 12th International World Wide Web Conference*, 2003.
- [32] S. Kamvar, T. Haveliwala, C. Manning, and G. Golub. Exploiting the block structure of the web for computing pagerank. Technical report, Stanford University, 2003.
- [33] S. Kamvar, M. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the 12th International World Wide Web Conference*, 2003.
- [34] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.
- [35] R. Levien. Attack resistant trust metrics.
- [36] R. Levien and A. Aiken. An attack-resistant, scalable name service. In *Draft submission to the Fourth International Conference on Financial Cryptography*.
- [37] S. Marti and H. Garcia-Molina. Limited reputation sharing in p2p systems. In *Proceedings of ACM Conference on Electronic Commerce (EC04)*, 2004.
- [38] Christos Faloutsos Michalis Faloutsos, Petros Faloutsos. On power-law relationships of the internet topology. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication Cambridge, Massachusetts, United States*, 1999.
- [39] B. Mobasher, H. Dai, T. Luo, Y. Sun, and J. Zhu. Integrating web usage and content mining for more effective personalization, 2000.
- [40] Lik Mui, Mojdeh Mohtshemi, and Ari Halberstadt. A computational model of trust and reputation. In *Proceedings of the 35th Hawaii International Conference*, 2002.
- [41] The Second Eigenvalue of the Google Matrix. The second eigenvalue of the google matrix. Technical report, Stanford University, 2003.
- [42] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford University, 1998.

- [43] Wolfgang Nejdl, Paul-Alexandru Chirita and Oana Scurtu. Knowing where to search: Personalized search strategies for peers in p2p networks. In *Proceedings of the P2P Information Retrieval Workshop held at the 27th International ACM SIGIR Conference*, 2004.
- [44] Alexandrin Popescul, Lyle Ungar, David Pennock, and Steve Lawrence. Probabilistic models for unified collaborative and content-based recommendation in sparse-data environments. In *Proceedings of the UAI-2001 Conference, Seattle, USA*, 2001.
- [45] M. Richardson, R. Agrawal, and P. Domingos. Trust management for the semantic web. In *Proceedings of the 2nd International Semantic Web Conference*, 2003.
- [46] K. Sankaralingam, S. Sethumadhavan, and J. C. Browne. Distributed pagerank for p2p systems. In *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, 2003.
- [47] S. Shi, J. Yu, G. Yang, and D. Wang. Distributed page ranking in structured p2p networks. In *Proceedings of the 2003 International Conference on Parallel Processing*, 2003.
- [48] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 161–172. ACM Press, 2001.
- [49] The slashdot rating based engine <http://slashdot.org>.
- [50] Web encyclopedia <http://en.wikipedia.org/>.
- [51] A. Yamamoto, D. Asahara, T. Ito, S. Tanaka, and T. Suda. Distributed pagerank: A distributed reputation model for open peer-to-peer networks. In *Proceedings of the 2004 Symposium on Applications and the Internet-Workshops*, 2004.
- [52] B. Yang and H. Garcia-Molina. Efficient search in p2p networks. In *Proceedings 22nd International Conference on Distributed Computing Systems*, Austria, 2002.
- [53] C. Ziegler and G. Lausen. Spreading activation models for trust propagation. In *Proceedings of the IEEE International Conference on e-Technology, e-Commerce, and e-Service*, 2004.