Finding, Extracting and Exploiting Structure in Text and Hypertext

Ola Ågren

PhD Thesis, 2009

Department of Computing Science
Umeå University

UMINF 09.12 ISSN 0348–0542 ISBN 978-91-7264-799-2

To my family

Till min familj

Abstract

Data mining is a fast-developing field of study, using computations to either predict or describe large amounts of data. The increase in data produced each year goes hand in hand with this, requiring algorithms that are more and more efficient in order to find interesting information within a given time.

In this thesis, we study methods for extracting information from semi-structured data, for finding structure within large sets of discrete data, and to efficiently rank web pages in a topic-sensitive way.

The information extraction research focuses on support for keeping both documentation and source code up to date at the same time. Our approach to this problem is to embed parts of the documentation within strategic comments of the source code and then extracting them by using a specific tool.

The structures that our structure mining algorithms are able to find among crisp data (such as keywords) is in the form of subsumptions, i.e. one keyword is a more general form of the other. We can use these subsumptions to build larger structures in the form of hierarchies or lattices, since subsumptions are transitive. Our tool has been used mainly as input to data mining systems and for visualisation of data-sets.

The main part of the research has been on ranking web pages in a such a way that both the link structure between pages and also the content of each page matters. We have created a number of algorithms and compared them to other algorithms in use today. Our focus in these comparisons have been on convergence rate, algorithm stability and how relevant the answer sets from the algorithms are according to real-world users.

The research has focused on the development of efficient algorithms for gathering and handling large data-sets of discrete and textual data. A proposed system of tools is described, all operating on a common database containing “fingerprints” and meta-data about items. This data could be searched by various algorithms to increase its usefulness or to find the real data more efficiently.

All of the methods described handle data in a crisp manner, i.e. a word or a hyper-link either is or is not a part of a record or web page. This means that we can model their existence in a very efficient way. The methods and algorithms that we describe all make use of this fact.

Keywords

Automatic propagation; CHiC; data mining; discrete data; extraction; hierarchies; ProT; rank distribution; S^2ProT; spatial linking; web mining; web searching

Sammanfattning

Informationsutvinning (som ofta kallas data mining även på svenska) är ett forskningsområde som hela tiden utvecklas. Det handlar om att använda datorer för att hitta mönster i stora mängder data, alternativt förutsäga framtida data utifrån redan tillgänglig data. Eftersom det samtidigt produceras mer och mer data varje år ställer detta högre och högre krav på effektiviteten hos de algoritmer som används för att hitta eller använda informationen inom rimlig tid.

Denna avhandling handlar om att extrahera information från semi-strukturerad data, att hitta strukturer i stora diskreta datamängder och att på ett effektivt sätt rangordna webbsidor utifrån ett ämnesbaserat perspektiv.

Den informationsextraktion som beskrivs handlar om stöd för att hålla både dokumentationen och källkoden uppdaterad samtidigt. Vår lösning på detta problem är att låta delar av dokumentationen (främst algoritmbeskrivningen) ligga som blockkommentarer i källkoden och extrahera dessa automatiskt med ett verktyg.

De strukturer som hittas av våra algoritmer för strukturextraktion är i form av underordnanden, exempelvis att ett visst nyckelord är mer generellt än ett annat. Dessa samband kan utnyttjas för att skapa större strukturer i form av hierarkier eller riktade grafer, eftersom underordnandena är transitiva. Det verktyg som vi har tagit fram har främst använts för att skapa indata till ett informationsutvinningssystem samt för att kunna visualisera indatan.

Huvuddelen av den forskning som beskrivs i denna avhandling har dock handlat om att kunna rangordna webbsidor utifrån både deras innehåll och länkarna som finns mellan dem. Vi har skapat ett antal algoritmer och visat hur de beter sig i jämförelse med andra algoritmer som används idag. Dessa jämförelser har huvudsakligen handlat om konvergenshastighet, algoritmernas stabilitet givet osäker data och slutligen hur relevant algoritmernas svarsmängder har ansetts vara utifrån användarnas perspektiv.

Forskningen har varit inriktad på effektiva algoritmer för att hämta in och hantera stora datamängder med diskreta eller textbaserade data. I avhandlingen presenterar vi även ett förslag till ett system av verktyg som arbetar tillsammans på en databas bestående av “fingeravtryck” och annan meta-data om de saker som indexerats i databasen. Denna data kan sedan användas av diverse algoritmer för att utöka värdet hos det som finns i databasen eller för att effektivt kunna hitta rätt information.

Preface

The thesis consists of the six papers listed below and an introductory part. In the introductory part, a general background on data mining is presented, as well as more in depth coverage of the areas that are more closely related to our research. The main findings of our research are described, as well as a proposed system for handling large amounts of discrete and semi-structured data. The main parts of this thesis are followed by an appendix containing a Users' Guide for CHIC (see Paper III).

List of Papers

I ÅGREN, O. ALGEXT — *an* ALG*orithm* EXT*ractor for C Programs*, Technical Report UMINF 01.11, Department of Computing Science, Umeå University, 2001.

II ÅGREN, O. Automatic Generation of Concept Hierarchies for a Discrete Data Mining System, in *Proceedings of the International Conference on Information and Knowledge Engineering (IKE '02)* (Las Vegas, Nevada, USA, June 24–27, 2002), pp. 287–293.

III ÅGREN, O. CHIC: A Fast Concept HIerarchy Constructor for Discrete or Mixed Mode Databases, in *Proceedings of the Fifteenth International Conference on Software Engineering and Knowledge Engineering (SEKE'03)* (San Francisco, California, USA, July 1–3, 2003), pp. 250–258.

IV ÅGREN, O. Propagation of Meta Data over the World Wide Web, in *Proceedings of the International Conference on Internet Computing (IC '03)* (Las Vegas, Nevada, USA, June 23–26, 2003), pp. 670–676.

V ÅGREN, O. Assessment of WWW-Based Ranking Systems for Smaller Web Sites, *INFOCOMP Journal of Computer Science* vol. 5, no. 2 (June 2006), pp. 45–55.

VI ÅGREN, O. S^2ProT: Rank Allocation by Superpositioned Propagation of Topic-Relevance, *International Journal of Web Information Systems* vol. 4, no. 4 (2008), pp. 416-440.

Other Publications

Outside the thesis work, and in addition to the papers listed above, Ola Ågren has (co-)authored the following publications:

- ÅGREN, O. Teaching Computer Concepts Using Virtual Machines, *SIGCSE Bulletin* vol. 31, no. 2 (June 1999), pp. 84–85.
- ÅGREN, O. *The DARK-Series of Virtual Machines*, Technical Report UMINF 00.15, Department of Computing Science, Umeå University, 2000.
- ÅGREN, O. Virtual Machines as an Aid in Teaching Computer Concepts, *IEEE TCCA Newsletter* (September 2000), pp. 72–76.
- ÅGREN, O. *BitSet: Implementing Sets of Natural Numbers Using Packed Bits*, Technical Report UMINF 02.10, Department of Computing Science, Umeå University, 2002.
- BÖRSTLER, J., JOHANSSON, O., LARYD, A., ORCI, T., SEGERBERG, H., AND ÅGREN, O. *Quality Management for Small Enterprises*, Technical Report UMINF 02.20, Department of Computing Science, Umeå University, 2002.
- Editor for the proceedings of Umeå's student workshop in Computer Architecture, 2000–2006.

Acknowledgements

A thesis is not something that can be done in isolation and there has been a lot of input from various sources that I am extremely grateful for.

My thesis supervisor, Associate Professor Jürgen Börstler, for giving me more or less free hands to pursue my own personal interests, for commenting on the almost endless sets of drafts, and long discussions on various parts of what research is and Computing Science (especially Software Engineering).

My thesis co-supervisor, Associate Professor Frank Drewes, for discussing the more technical aspects of what I have been working on, for creative revisions, and for being there as a friend and a former (and future?) Table Tennis team mate.

The staff at the Department of Computing Science, especially Steven Hegner, Michael Minock, Lena Kallin Westin, Per-Åke Wedin, and *all* of the administration and support staff. This includes all those that inspired me but have moved on in life: Peter Jacobson, Olof Johansson, Krister Dackland, and those that have worked as teaching assistants in my courses over the years.

The staff at the Department of Interactive Media and Learning, you made me feel welcome and gave me support.

On a more personal level I must say that I'm unable to thank my friends enough. You already know who you are, but a short list of your names[1] includes: Anders, Anna, Anne, Annelie, Annika, Anton, Bertil, Björn, Britta, Cecilia, Claes, Clas, Claudia, Daniel, David, Elin, Elina, Emelie, Emilott, Eric, Erik, Erika, Eva, Fredrik, Frida, Gunnar, Göran, Hanna, Hans, Helena, Henrik, Ingemar, Ingrid, Jan, Jannek, Jenni, Jennie, Jennifer, Jenny, Jens, Jeroen, Joakim, Johan, Johanna, Jonas, Jörgen, Katarina, Klas, Krister, Kristina, Lars, Leif, LenaMaria, Lennart, Lina, Linda, Linus, Lisa, Lotta, Lovisa, Magnus, Malin, Marcus, Maria, Marianne, Martin, Mattias, Melker, Mikael, Mona, Mårten, Niclas, Niklas, Nikoletta, Nils, Nina, Ola, Olov, Oskar, Palle, Per, Per-Olof, Peter, Petter, Pär Anders, Rickard, Rikard, Robert, Roger, Runa, Sabina, Sandra, Sara, Sigrid, Simon, Sofi, Stefan, Stephan, Teresa, Therese, Thomas, Tomas, Tommy, Ulrika, Valentin, Viktoria, Viveka, Wenche, Åke, Örjan, and probably some more that I missed. You are the best!

[1] Name appears once, even if I know more than one with that name.

Finally, and most importantly, I thank my family; My mother Solveig, father Sten and former wife Anneli for supporting me and allowing me to follow my own paths in life. My son Simon for being the sunshine of my life. My brother Bo, his wife Maria and their fantastic daughters Sanna and Emma for being there for me. My cousins and their families for being great sources of inspiration.

Live Long and Prosper!

Contents

Chapter 1

Introduction

Each year more and more data is generated in various forms, currently in the multiple Exabytes per year range. Most of this data is in some type of raw format that must be refined before it can be used and understood. Moreover, a lot of this data is stored on magnetic media of some sort for later retrieval.

To give some sense of scale, we will look at the Internet as an example of how much information is available. Figure 1.1 on the following page displays the number of hosts (registered host names) available on the Internet over the last 25 years. Moreover, the figure shows that the data traffic over the Internet has increased even faster. In fact, the sheer volume of data added each week is so large that no one would be able to read all the information within a lifetime. This means that tools must be used to help find interesting information, or even go so far as to draw conclusions from the available data.

The tools might be simple or complex, but they can only work with the data they are given. The simplest tools available can only search for records containing exact matches of the keywords given in the query. Having more information about each document means that we can use more complex tools for that data-set. Two very important attributes here are how structured the data-set is and whether there is additional meta-data for each record.

If the data exist in a database or other forms of formally defined data-set, usually called *structured data*, it can easily be searched and used by software. The so called Deep Web is built up of rich information and databases accessible using forms or other software, and is usually seen as being much larger than the static web [100].

A bigger problem exists if the data has no apparent structure or has only a minimal inherent structure. General text files tend to have no structure outside of those on the syntactical level, and are often seen as a type of *unstructured data*.

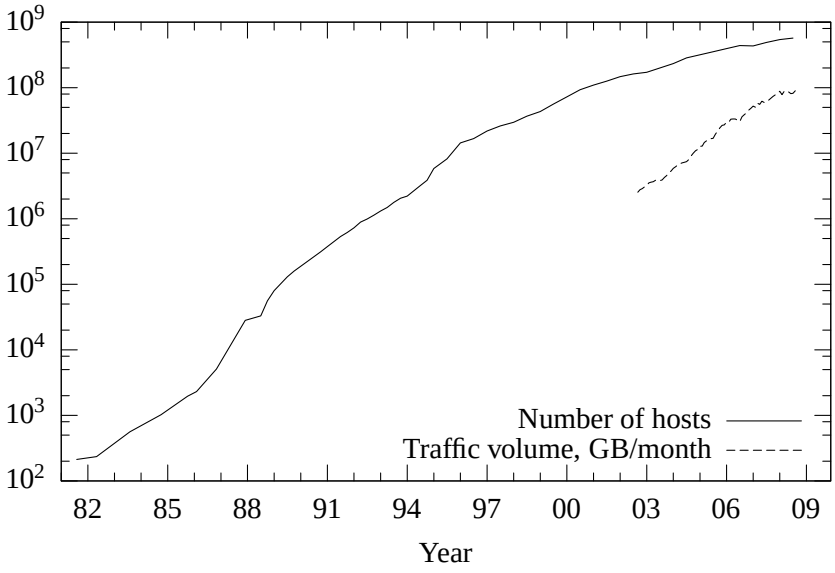


Figure 1.1: The number of hosts on the Internet 1982–2008 [73] and traffic through AMS-IX (the largest Internet exchange point) during 2002–2008 [9].

Some form of natural language processing is usually required to use such data-sets efficiently as soon as something more complex than a pure keyword search has to be done.

There is also the middle ground between structured and unstructured data called *semi-structured data*. A lot of the file types currently found on the Internet (such as HTML and PDF) allow a number of structural parts such as headings and hyperlinks pointing to other documents. This is the type of data that we have focused on, even though our indexing works on unstructured data as well.

It is also possible to use descriptive information about the data rather than the actual contents of the data. Such information is called *meta-data* and usually contains information about elements and attributes, records and structures, and provenance of the data. Typical examples of meta-data include the name, size, data type and length of each field available in the data-set, as well as where it is located, its association to other data, ownership of the data, etc. Meta-data can sometimes be seen as a model of the original data, thereby allowing applications and users to search and browse the available meta-data rather than the original data-set. As an example, the abstract of a book is together with the CIP record[1] supposed to give a

[1] A Cataloging in Publication record is a bibliographic record prepared by the American Library of Congress for a book that has not yet been published. When the book is published, the publisher

selling but objective sampling of the content of a book without going into details. This can be seen as a case where meta-data is used to enhance the usefulness of the data it describes [101]. An analogue would be to use thumbnails to provide an overview of all available pictures in a gallery rather than using the full pictures directly.

This thesis collects the results from three different projects dealing with semi-structured data or meta-data. The first one, called ALGEXT, extracts meta-data from source code. The second one, called CHIC, finds structure within large sets of discrete meta-data. The last, and most important, one is the PROT project. It uses structural as well as textual elements from semi-structured documents in order to rank them, and is by far the most complex of the three projects.

1.1 Research Questions

The main questions that the research in this thesis tries to answer are:

- How can we find and extract structural information or embedded data from discrete data-sets?
- How can we find and rank web-pages in a topic-sensitive way by algorithms that are more efficient (at least in practice) than the currently known ones?

includes the CIP data on the copyright page. This makes book processing easier for both libraries and book dealers [8].

Chapter 2

Data Mining

Data mining is a collective term that stands for a number of different procedures and methods for finding interesting information in large amounts of data. Other names used for the concept include deductive learning, exploratory data analysis, and data driven discovery [47].

While data mining can be applied to any type of data-set, it has been used extensively in business systems. Data mining tools usually do not work directly on a "live" database that contains day to day transactions, but operate on a modified and summarised database called a *data warehouse*. A data warehouse contains aggregated and cleaned information from the "live" databases, i.e. ideally no false or extraneous data [47, 71, 133].

Another difference between a regular database system and a data mining system is in their operation. The user of a database system expects a crisp answer to each query, for example, is a seat available on a certain flight. The answer given by a data mining system could be in the form of possibly interesting patterns or meta-data describing something in the database, for example, every user that bought article *a* also bought article *b* [47].

Data mining approaches are usually grouped into either predictive or descriptive systems, according to the taxonomy in Figure 2.1.

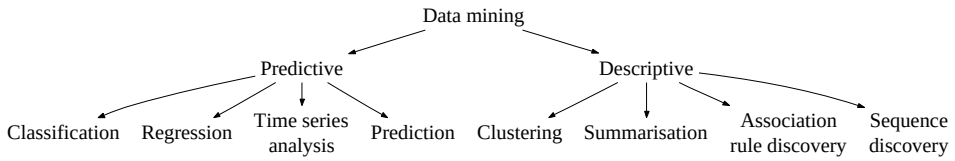


Figure 2.1: Data mining taxonomy [47].

A *predictive* system makes some sort of a prediction for new values based on already known values in the data-set. Predictive approaches include classification, regression, time series analysis and prediction systems.

- A *classification* system maps each object into one of a number of predefined classes.
- A *regression* system finds some sort of function that as closely as possible matches the given data. One of the the attributes of the data is modelled by the other attributes using the function.
- *Time series analysis* examines the behaviour of a value over time.
- A *prediction* system tries to foresee a future state given current and previous states. These systems are often used to give early warning for natural phenomena, like earthquakes and flooding, as well as other semi-unpredictable systems like speech recognition and pattern recognition [47].

A *descriptive* system tries to find relationships in data, e.g. patterns. Most of the time it is not used to predict future values, but can be used to analyse different attributes in the data. Descriptive approaches include clustering, summarisation, association rule discovery and sequence discovery.

- *Clustering* is related to classification, but creates the classes by using the existing data.
- *Summarisation* diminishes the amount of data, while retaining as much significant information as possible about the initial data set.
- *Association rule discovery* tries to find associations between different items in the database.
- *Sequence discovery* looks for patterns where one event leads to another event.

Most of our work has been on different descriptive approaches, including information extraction (see Section 2.1), clustering (see Section 2.2), mining for association rules (see Section 2.3) and web-based data mining (see Chapter 3).

2.1 Information Extraction

Information extraction is the process of extracting, computing and compiling information from the text of a large corpus using machine learning [61]. Information extraction systems are generally used to extract information about a page, storing it in a form that makes queries and retrieval of the data as easy and efficient as possible. Typical examples of information extraction systems include RSV [53] and WHISK/CRYSTAL [125, 126].

2.2 Clustering

One widely used form of descriptive data mining is *clustering*. Clustering is related to classification, but uses the existing data to automatically generate the classes. Clustering is also called *unsupervised learning* or *segmentation*. A very important notion in clustering is *similarity*, i.e. how closely related two (or more) items are to each other. Clustering works by automatically grouping together smaller clusters (i.e. data points with similar values) until either each cluster is "sufficiently" large or a certain (predefined) number of clusters have been reached.

Clustering can be seen as finding groups of facts not previously known in large data. There are numerous ways of clustering data-sets, depending on, for example, type and distribution of data. An excellent review of the state of the art in data clustering was published by Jain et al. in 1999 [76].

2.3 Mining for Association Rules

Mining for association rules is looking for patterns according to which one item is connected to another item. There are many different applications available supporting mining for association rules. Most of them fall into two different categories, unsupervised and supervised mining (see Sections 2.3.1 and 2.3.2, respectively).

The rules found are usually in the form of an implication $X \Rightarrow Y$. Each rule found is marked with the quality attributes called support and confidence. A formal definition is included in order to introduce the notation and terminology used later in this text.

DEFINITION 2.1 Let $\mathcal{I} = \{I_1, I_2, \ldots, I_m\}$ be a set of *items* or an *itemset*. Let $\mathcal{D}$ be a set of *transactions*, where each transaction T is a set of items, $T \subseteq \mathcal{I}$. We say that a transaction T *contains* X if $X \in T$. The fraction of transactions containing X is called the *frequency* of X. An *association rule* is an implication in the form $Y \Rightarrow Z$, where $Y, Z \subseteq \mathcal{I}$, and $Y \cap Z = \varnothing$. This rule holds in the transaction set $\mathcal{D}$ with *confidence* α if the fraction of the transactions containing Y that also contain Z is at least α. The rule has *support* s in the transaction set $\mathcal{D}$ if the fraction of the transactions in $\mathcal{D}$ that contain $Y \cup Z$ is at least s [47].

EXAMPLE 2.1 Given a database with three items $I = \{i_1, i_2, i_3\}$ and five transactions $D = \{\{i_1\}, \{i_1, i_2\}, \{i_1, i_2, i_3\}, \{i_2, i_3\}, \{i_2\}\}$, we can say that the support for $i_1 \Rightarrow i_2$ is $s(i_1 \Rightarrow i_2) = \frac{2}{5} = 40\%$ and the confidence is $\alpha(i_1 \Rightarrow i_2) = \frac{2}{3} \approx 67\%$. We can also see that $s(i_2 \Rightarrow i_3) = \frac{2}{5} = 40\%$ and $\alpha(i_2 \Rightarrow i_3) = \frac{2}{4} = 50\%$, so $i_1 \Rightarrow i_2$ has more confidence than $i_2 \Rightarrow i_3$ while they have equal support.

2.3.1 Unsupervised Mining for Association Rules

Unsupervised association rule mining systems automatically search $\mathcal{D}$ to discover association rules having high confidence and support, without being guided by input from the user. The most commonly used algorithm is called *Apriori* [47, 129].

Apriori builds upon the fact that only subsets of large sets can be large, and the assumption that only large subsets can give new information that is potentially important. This means that the possible solution space can be pruned quickly while checking the combination of all itemsets that differ in only one member. A support parameter s is used in Apriori to decide which itemsets are considered large. This means that the algorithm will ignore rules with high confidence if the support is too small [47, 64, 129].

Example 2.2 on the facing page shows how Apriori would prune the solution space using already found information. Unsupervised data mining will not be discussed further in this thesis, since it is not the focus of the work described here.

EXAMPLE 2.2 Suppose we are given a data-set with three items (A, B and C), and two of these (A and B) appear frequently while the last one (C) does not. Combining item C with either A or B results in an itemset that is not frequent enough, thus indicating that these can safely be ignored from further calculations by Apriori. This implies that only the intersection of A and B (denoted $\{A, B\}$) can be a frequent itemset when combining these itemsets. This is illustrated in the lattice in Figure 2.2.

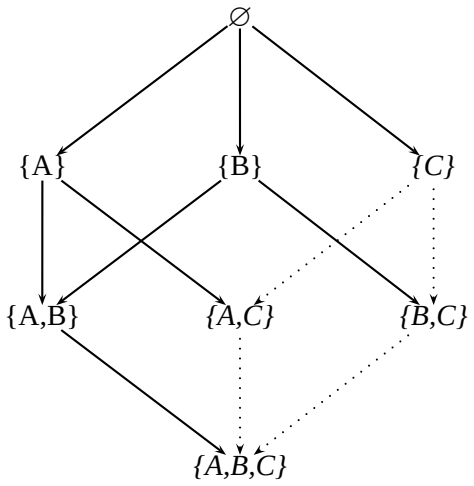


Figure 2.2: Lattice of itemsets for Example 2.2, with frequent and *infrequent* itemsets. Dotted lines can safely be ignored by Apriori, since at least one of its parents are infrequent.

2.3.2 Supervised Mining for Association Rules

Supervised mining for association rules is often performed on a data warehouse, where the available data is already somewhat summarised and cleaned. The data that is operated upon is usually described not only by a value, but also by a number of attributes that describe from whom and where it came [47, 64].

EXAMPLE 2.3 Assume that we have a company that has two branches in Umeå and one in Stockholm (both cities are in Sweden) as well as one in London, England. The company sells electronics (television sets and portable CD players, among other products) and mirror shades. To improve its operations the company wants to use the sales data gathered from each branch to make estimates of the number of units of each product type which need to be preordered. Each branch of the company has sent in the sales figures for each product type for each day to the central data warehouse that houses this data in a database. This means that each data value is marked with a number of attributes, such as date, location, and product type.

Each attribute of Example 2.3 can be seen as a hierarchy on its own, as illustrated in Figure 2.3. Such hierarchies are often referred to as *concept hierarchies* [47, 64]. We have explored automatic generation of concept hierarchies using CHiC [140, 141, Papers II–III].

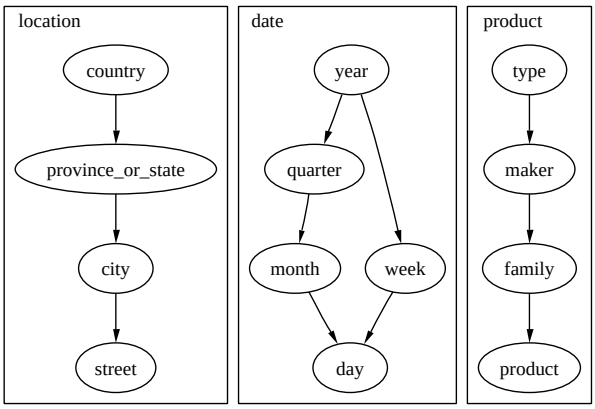


Figure 2.3: Examples of concept hierarchies for Example 2.3.

The usual way of looking at the data in a supervised system is in the form of an *n*-dimensional *data cube*, such as the one in Figure 2.4. Each side of the data cube corresponds to the current view level in the hierarchy of the attribute, also called *facet*. Another name for a data cube is *On-Line Analytical Processing (OLAP) cube* [47, 64, 129].

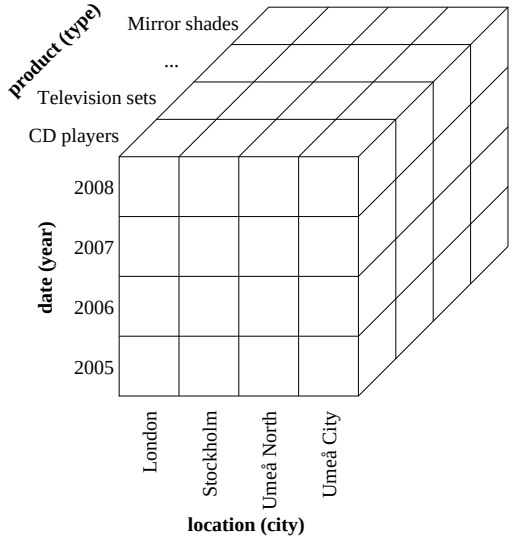


Figure 2.4: Data cube corresponding to Example 2.3 and the hierarchies (facets) shown in Figure 2.3 on the facing page.

Each block in the cube corresponds to the chosen hierarchy level of each facet. It is marked with the aggregated values of all underlying levels in the hierarchy as well as its support (see Definition 2.1 on page 8). This cube can be changed and examined by the user to find interesting patterns.

It is up to the user to choose operations (see Table 2.1 on the following page) in such a way that new information can be deduced from the database. This means that the output from using supervised mining for association rules will depend on the user expertise in both the domain and the tools used.

A typical starting point would be to look at highly aggregated values over either time (e.g. sales data per year) or per product, and then drilling down to find patterns in the data. This would show variations due to season, locale, or product in our example.

Table 2.1: Typical operations that can be performed on a data cube [47, 64].

Operation	*Result*
Transpose	Changes the positions of facets with respect to the others
Slice/dice	Choose a specific slice in one (or more) dimension
Drill-down	Goes to a lower level in the hierarchy of one facet[a]
Roll-up	The opposite of drill-down

[a] The data cube in Figure 2.4 on the previous page has already been drilled down to the "city" and "type" level in the location and product facets, respectively.

2.4 Thesis Contributions

There are two separate subareas within data mining that have been studied in the present thesis. Each of them will be handled in its own subsection below.

2.4.1 Algorithm Extraction

A lot of research has been done to ensure that documentation of software is as up-to-date as possible, but there are still some open problems. There is usually a semantic gap between the source code and the documentation for several reasons; the source code and the documentation are not always written at the same time, different programs are probably used to edit them, etc. This means that whenever a change is done in one, it needs to be transferred to the other.

Our approach is to extract some parts of the documentation from the software. This means that the documentation and source share the same file, implying that there is an increased likelihood that the documentation will be updated whenever a change is made to the source statements and vice versa; cf. the simplified version of literate programming seen in c~~web~~ [52].

While truly automatic extraction of algorithms has not yet been mastered, it is at least possible to use comments in order to add to the source code whatever information is required. ALGEXT [139, Paper I] is a proof of concept implementation that extracts all *strategic* comments from ANSI C (see Example 2.4 on the facing page), retaining the indentation of the source code in the extracted comments.

The main idea is to allow a textual description of the algorithms to be embedded within the source code, and extract it when required. This works in a similar way to `cextract`[1], doxygen [134] and Javadoc [54] to extract (part of) the function comments of the source files, but with less requirements on the comment mark-up from the tool's viewpoint; Example 2.4 shows that the comments can be quite

[1] Source code available from `http://dev.w3.org/cvsweb/Amaya/tools/cextract-1.7/`.

elaborate because of the requirements from other tools, in this case LaTeX.

EXAMPLE 2.4 Given the following source code:

```
/* Function f(x) = x*x*x-x */
int f(int x)
{
        /* \begin{equation} \mathcal{Z \rightarrow Z}, f(x) = x^3-x */
        return x*x*x-x; /* will not work if x is too large */
        /* \label{eq:f} \end{equation} */
}
```

The LaTeX embedded in the tactical comments of the source code above generates Eq. (2.1).

f:

$$\mathcal{Z} \rightarrow \mathcal{Z}, f(x) = x^3 - x \tag{2.1}$$

2.4.2 Structure Extraction

The concept hierarchies used when doing supervised mining for association rules (see Section 2.3.2) are normally defined at the same time as the database or by using predefined hierarchies, such as Dublin Core [46] or LOM [72]. This will not work that well when the data-set consists of free-text terms or free-text meta-data describing each record. The main problem is that changing the set of records to be included may yield a different set of terms to use as well. This means that there has to be an automated process to find the concept hierarchies given by the terms.

The process finds subsumptions, i.e. terms that exist in a record only if another term exists there as well but not vice versa. These subsumptions are then used to build up hierarchies that can be used either for semantic searches for documents or for doing supervised mining for association rules among the records.

The use of concept hierarchies to increase the number of documents found has been very successful in information retrieval. A larger set of documents can often be found by enriching the queries with terms that subsume the original terms in the hierarchy [30, 120].

At the turn of the century there were no tools available that could generate a concept hierarchy specifically made for supervised mining for association rules. It was possible to use decision tree inducers that could generate binary trees by checking one attribute at a time, using algorithms that were not optimised for crisp data-sets.

This was the main motivation for creating the CHiC tool, that is able to induce a concept hierarchy of terms given keyword based data [140, 141, Papers II–III]. It

was originally designed to work together with a proprietary data mining system on the IRIX platform, but it is easy to adapt the output to most data mining systems.

There have been two upgrades of functionality in the tool from Paper II [140] to Paper III [141], both being driven by an upgrade of the functionality of the data mining system. The first upgrade was to allow generation of concept lattices rather than hierarchies. This means that more than one path to a subsumed keyword may exist in the resulting data-set. The second upgrade was to allow terms to be reused in different facets, as long as there is no overlap between the keywords of the facets. Reusing terms increases the coverage in the later facets. Turning these options on means an increased amount of work required to generate the results (see Paper III [141, Sections 8.6.1 to 8.6.2 on pages 96–98]).

The clustering generated by CHIC is not guaranteed to be optimal, since the algorithm uses local minima to select decision points. Experience shows that it generates appropriate results in almost all practical cases; We tested thousands of data-sets and found only three hierarchies that did not quite make sense.

Figure 2.5: Problems with applying Data Mining on sales data.
With permission from PIB Copenhagen A/S 3/2004.

Chapter 3

Web Search Engines

What is currently known as Internet started out as a research network with packet switched data called ARPANET. It grew larger and larger as more computers and networks were added to it over time, and some of the older protocols were replaced to get more stability and/or throughput.

It was first and foremost used for transmitting text messages and text files, until Tim Berners-Lee from CERN in Switzerland created the first working prototype of what is now known as the World Wide Web. It consisted of a web server, a combined browser and editor, and a number of pages that described the project. Some of the technologies that we now take for granted were first introduced in this project, e.g. globally unique identifiers (Uniform Resource Identifier).

There were originally very few servers up and running, so it was possible to keep track of all of them and then manually browse to find the wanted material. It did, however, not take long until the number of servers was too large to keep track of manually (see Figure 1.1 on page 2). This meant that some sort of look-up service was required.

Along came the first generations of *web search engines* [20, Section 4.72], e.g. AltaVista.[1] They indexed all pages they could reach and provided their users with an easy way of doing searches. They usually had no way of ranking the pages, instead they gave their answers in an unspecified (albeit usually deterministic) order. The key to using these search engines was to add enough search terms (both positive and negative) to a query to get the right number of pages.

Over the years more and more advanced search engines appeared. These search engines used various techniques to give better search results. Among the most successful and prominent ones is the idea to use the links between web pages to

[1] Their original search engine became operational in 1995 and was located at `http://www.altavista.com`. They have later created far more advanced search engines.

derive rankings indicating the relative importance of pages. Approaches based on this idea will be discussed in this chapter.

3.1 Web Mining

Web mining is data mining using data from the web. Within this field, there are the following five major research areas:

Information extraction Finding, extracting and compiling information from a large corpus, see Section 2.1 on page 7.

Wrapper induction The process of finding general structural information about a set of web pages, and with this in mind extract only the relevant information from each page [36, 108, 109].

Vector space modelling and (latent) semantic indexing A method for extracting and representing the similarity between documents and the meaning of words from the contexts, by applying statistical computations to a large corpus of text [94, 119, 129].

Web link mining Mining the spatial link structure of the web for information, see Section 3.2.

Web log mining Mining for knowledge in web logs, otherwise known as *click-stream* data [25, 48, 103, 137].

3.2 Web Link Mining

The main part of our work concerns web link mining. A lot of research has been done by exploring the link structure between pages[2], especially about algorithms for very large data-sets such as the entire world wide web. Pages on a specific subject tend to have links to other pages on the same subject [42, 51, 81]. Neighbouring web pages (when using hyper-links to define distance) can be used to either deduce or corroborate the contents of a web page. Web link mining systems usually look at both the quantity and type of links, often removing or decreasing the effect of local links since these tend to be navigational rather than referential.

The web can be seen as a graph (V,E), where each vertex corresponds to a web page and each edge corresponds to a hyper-link. By using a predefined order among the vertices we can find a unique adjacency matrix corresponding to the web. Almost all web link mining algorithms use such an adjacency matrix, returning one or more eigenvectors corresponding to the eigenvalues of the adjacency matrix. Such eigenvectors can be seen as rating functions, giving a ranking or retrieval order for the corresponding pages.

Most of the research in web link mining has focused on variants of two algorithms called PageRank (see Section 3.2.1) and HITS (see Section 3.2.2).

3.2.1 PageRank

The general idea behind PageRank [26] is that of a *random surfer* browsing the web, at each time following a random link on the current web page. Given a sufficiently large number of simultaneous surfers, it would be possible to stop them at any given time and look at the number of surfers currently looking at each page and use that number as the relative probability that it is an important page.

There were some problems with this approach[3], i.e. what to do when there are no outgoing links from a page and when two (or more) pages point to each other without outgoing links from the group (*rank sink*). The answer to the first problem was to recursively remove all pages lacking outgoing links from the calculations. The latter problem was countered by adding the possibility of jumping to any page on the web at a certain probability called a *damping factor* $(1-\mu)$. The damping factor corresponds to the likelihood that a random surfer would jump to a random page rather than follow one of the links on current page. This value

[2]This can be seen in the proceedings from IJCAI Text-Mining & Link-Analysis workshop 2003 [62], LinkAnalysis-2005 [63], LinkKDD [3, 4, 5, 44], SIAM Workshop on Link Analysis, Counterterrorism and Security [13, 41, 124, 130], as well as papers published in other venues [56, 90, 110, 128, 131].

[3]Besides getting enough surfers to click at random.

must be between 0 (inclusive) and 1, and a value of 0.15 was used by the original authors [112]. The original PageRank algorithm gives a value for each page $j \in V$, which is obtained by solving Eq. (3.1) with $n = |V|$, using iteration to find a fixed point.

$$PR(j) = \frac{1-\mu}{n} + (\mu) \times \sum_{(i,j)\in E} PR(i)/\text{outdegree}(i) \tag{3.1}$$

This can also be described by using a matrix P obtained from the column-normalised adjacency matrix M (with all pages without links removed) of the graph (V,E) by adding the damping factor:

$$P = \left[\frac{1-\mu}{n}\right]_{n\times n} + \mu M \tag{3.2}$$

The rating returned, which is called PageRank, is the dominant eigenvector of P: $P\pi = \pi, \pi \geqslant 0, ||\pi||_1 = 1$. This means that the i-th entry of π is the probability that a surfer visits page i, or the PageRank of page i.

Today n is between 15-20 billions and computing the eigenvector π was already in 2002 called the largest matrix computation problem in the world [104].

3.2.1.1 Rate of Convergence

It has been proved that the second largest eigenvalue of P will never be larger than μ [68], leading to fast convergence when using power iteration to find the PageRanks.[4] It has also been shown that PageRank can achieve a stable state in $O(\log n)$ iterations, where n is the number of pages in the data-set. While this is sufficient for most applications, there have been a number of proposals for speeding up the calculations so it can be used for ranking large data-sets such as the entire Internet [10, 27, 39, 65, 66, 74, 82, 83, 84, 95, 114]. Typical examples of methods used for efficiency improvement include Arnoldi [121], Lanczos [60], Jacobi [23] and Gauss-Seidel [10].

[4] Because the power method converges at a rate proportional to $|\lambda_1/\lambda_2|$ [60] and P is an irreducible n-state Markov chain, which means that power iteration will always converge to a stable value [75, Theorem 5.2].

3.2.1.2 Problems and Variants

There are two main problems with the basic PageRank algorithm. The first problem is that there are huge computational costs involved in calculating the PageRank values once (described in the previous section). The second problem is that the values calculated represent an average random surfer rather than someone interested in one specific subject, thus potentially leading to an answer set that is not of interest for all users.

Two variations of PageRank have been widely used to counter the "randomness" problem. These are Personalized PageRank [112] and Topic-sensitive PageRank [67]. They both use the same general ideas and algorithm as the original PageRank, except that the damping factor is not added uniformly. Instead, a damping is scaled and added to either one starting page (for Personalized PageRank) or to a set of pages (for Topic-sensitive PageRank) assumed to be about that particular subject, which indicates that PageRank will have a preference for these pages over other pages. Personalized PageRank will thus give a view of the Internet from the viewpoint of one specific starting page.

Topic-sensitive PageRank has been used quite extensively, but suffers from a major problem when it comes to rate of convergence: Adding the damping factor to just some entries in the matrix makes it reducible. This means that several eigenvalues of the same magnitude might show up, thereby making the convergence of power iteration very slow [60]. This can partly be offset by using the approach of Jeh and Widom [77, 78], namely by creating base vectors for important pages. This corresponds to a partial view of the Internet according to each important page, by using Personalized PageRank with an extreme damping factor. The base vectors are scaled according to the corresponding eigenvalues and those that belong to the required set are aggregated and normalised in order to form the final answer vector. The rather small dampening factor used in PageRank still means that many iterations are required before a stable answer can be found for each base vector. Topic-sensitive PageRank is thus better for broader topics, so that each use of the vector can be seen as amortising the cost to generate it.

3.2.2 HITS

The basic idea behind HITS is that important pages about a specific subject have pages with links pointing to them, and pages with good links tend to point out important pages [88]. The algorithm gives two separate values for each page; how valuable the contained information is according to the algorithm (called *authority*) and also how good it is as a link page (called *hub*).

Rather than addressing the entire Internet directly it uses a bootstrap data-set, consisting of pages that are initially assumed to be about a specific subject. This set is further extended with all pages pointed to by the bootstrap set as well as pages that point to the bootstrap set. Each page in the entire set is given a start value in the two categories. These values are adjusted by simultaneous iteration over the equations given in Eq. (3.3), where η_i denotes the hub value for page i and α_j the authority value for page j.

$$\eta_i = \sum_{(i,j)\in E} \alpha_j \qquad \alpha_j = \sum_{(i,j)\in E} \eta_i \tag{3.3}$$

Eq. (3.3) can also be described in terms of operations on the corresponding adjacency matrix A:

$$\eta = A^T\alpha = A^TA\eta \qquad \alpha = A\eta = AA^T\alpha. \tag{3.4}$$

We remark that, in practice, the matrix products in Eq. (3.4) are never computed explicitly. All eigenvector-based methods only perform matrix-vector multiplications that make use of the sparse structure of the adjacency matrix A.

One thing to note here is that even though the required results are obtained in the relative differences between individual values in η and α, it is necessary to keep these values within $(0,1)$ by using normalisation after each iteration of Eq. (3.4). These values can otherwise become so large as to cause overflows in calculations.

3.2.2.1 Rate of Convergence

The basic HITS algorithm usually has a very good convergence rate, since it could be seen as two simultaneous power iterations on symmetric non-negative matrices [60]. Using a bootstrap set also creates a data-set (and corresponding adjacency matrix A) that is *much* smaller than the entire Internet, leading to much faster evaluation of the hub and authority values.

3.2.2.2 Problems and Variants

HITS suffers from a problem called *topic drift*. Topic drift occurs when pages that are barely on-topic receive high hub and authority ratings, since these pages are a part of another close-knit society of pages linking to each other. This means that if more than one topic can be found within the extended data-set the one with the largest eigenvalue will be given. Possible solutions to this problem were given in the CLEVER project [32, 34] as well as the work of Bharat and Henzinger [21]. CLEVER uses different weights on the links depending on the the number of links and whether they reside on the same server, while Bharat and Henzinger used either outline filtering or dividing the weight of each link with the total number of links between same two servers.

The numerical stability of the calculations can sometimes be less than adequate, meaning that small changes (such as missing links) in the input data can change the focus from one cluster of pages to another. Possible solutions to this problem were given by Miller et al. [103] and Ng et al. [111]. Miller et al. used web logs and up to two link steps to generate the adjacency matrix, while Ng et al. used random walks in a manner similar to PageRank.

Another problem is that many different meanings of the same word can appear within the data-set. It is often the case that these meanings can be found by checking more than the first eigenvalues for the combination, using what is called spectral graph theory [92, 111].

3.3 Thesis Contributions

We have used two major approaches to obtain a web search system that is stable, fast and, according to the users, returns good answer vectors.

3.3.1 Monotone Data Flow System

The first approach is to put *trust levels*[5] on the meta-data belonging to a page and then propagating it along links. The propagation is controlled by

- hyper-links (either given explicitly in the web pages or implied by the paths of the URLs),
- the trust level given to each page,
- whether the data was perceived as pervasive, i.e. should propagate more than one link, and
- a function that calculates the resulting meta-data using the incoming values from each link.

This corresponds to a weighted Topic-sensitive PageRank where each non-pervasive value can be propagated just one step along the links, and all links have weight. The approach builds on the work done by Kam and Ullman [80], with an updated propagation step to fit the requirements of our model.

The prototype is quite slow and requires inside knowledge to be used successfully; well-defined trust rules as well as a relatively small input data-set are essential. Experiments with the prototype gave very positive results, even though both the model and the resulting search engine are more of a theoretical and academic, rather than a practical, nature [142, Paper IV].

[5] How much trust we put in that page regarding each piece of meta-data.

3.3.2 Propagation of Topic-Relevance

The Propagation of Topic-Relevance[6] (ProT) algorithm is a close relative to Topic-sensitive PageRank, but with a major change. All links used in the calculations potentially have the same strength; the value to propagate is divided by the *decay factor* (ξ) rather than dividing the value to propagate among the outgoing links as in PageRank. This means that the propagation step requires a little less work, but it does require both a very carefully chosen ξ and normalisation after each iteration [144, Paper VI].

Given an initial score $\varpi(j,0) = 1$ (100%) for pages that are assumed to be on-topic and zero otherwise, and using k as the iteration count as well as setting ξ to an appropriate value (i.e. just above the dominant eigenvalue of the adjacency matrix) we can apply the following algorithm:

$$\varpi(j,k) = \frac{1}{\xi} \sum_{(i,j)\in E} \varpi(i,k-1) + \begin{cases} \varpi(j,k-1) & \text{if } j \text{ is on-topic} \\ 0 & \text{otherwise.} \end{cases} \tag{3.5}$$

The final answer is given after normalisation of the k:th ϖ vector.

This means that the final answer depends on both the links of the web and which pages are on-topic, controlled by the choice of ξ. This corresponds to changing the value of the damping factor of Topic-sensitive PageRank, albeit using a value for the damping factor that is far away from the usual choices.

[6] The name was originally Propagation of Trust [143, Paper V].

3.3.2.1 Problems and Variants

The matrix that ProT operates on is a composition of the adjacency matrix of the original web and self-referential links for all pages that are on-topic. The problem with this matrix is that it is reducible, meaning that the matrix might have several eigenvalues of the same magnitude. This leads to *very* slow convergence when using the power method to find the dominant eigenvalue (and corresponding eigenvector) of the matrix. The convergence rate of ProT is on the same order of magnitude as for the original Topic-sensitive PageRank, as we have shown [144, Paper VI, Section 11.5.1].

Our solution to this problem is to look at one starting page at a time, then adding up all resulting vectors (called *basic vectors*) to generate a final result vector (after normalisation). This also has the advantage that larger values of ξ can be chosen, leading to even faster convergence. We call this version *Superpositioned Singleton Propagation of Topic-Relevance* (S^2ProT) [144, Paper VI, Section 11.4.4].

Another solution is the *Hybrid Superpositioned Singleton Propagation of Topic-Relevance* (HyS^2ProT) algorithm, using the same general idea as S^2ProT but diminishing each propagated value further by dividing the value with the number of outgoing links, in the same manner as in PageRank [144, Paper VI, Section 11.7.2]. The main advantage of this approach is that the matrix has a dominant eigenvalue of 1, since it uses a normalised matrix in the same way as PageRank (see Section 3.2.1 on pages 17–18). This also means that an even larger value of ξ must be chosen, since the starting values will otherwise propagate further along the links.

3.3.3 Evaluation of Empirical Result

We have used three different ways of evaluating the algorithms:

- Empirical assessment of result relevance using human graders. In Paper V [143] we took the top pages given by each algorithm and added them to a questionnaire for each chosen search term. Volunteer graders graded each page according to its perceived relevance with respect to the search term. The average of all valid answers[7] of pages belonging to the top pages of each algorithm was calculated, and compared with the results from the others.

  This assessment method was reused with minor changes in Paper VI[8]. All results indicate that our algorithms (especially S^2ProT) yield good answer sets according to the graders [144, Paper VI, Section 11.6.1.4].

- Experimental assessment of algorithm stability.
  - The stability of each algorithm when removing pages from the set of on-topic pages were tested. The results indicated that S^2ProT were more stable than Topic-sensitive PageRank, which in turn was more stable than ProT [144, Paper VI, Section 11.6.2.1].
  - The stability of each algorithm when removing links from the data-set was tested. The results show that our algorithms are very stable; the ranking order between the algorithms varies slightly depending on which measurement we use [144, Paper VI, Section 11.6.2.2].

[7] Ignoring grades of "Don't know" [143, Paper V, Section 10.2.2 on page 126].
[8] See Section 11.6.1.3 on page 168.

3.4 Summary

We have made extensive qualitative studies in various aspects of the algorithms described in this chapter, presented in Table 3.1 on the next page. Some of the results were discussed in Section 3.3.3 on the preceding page and have already been published [144, Paper VI, Sections 11.5 to 11.6], while others (specifically some of the HITS data) are based on data found in other sources [88, 103, 111].

We have graded each algorithm on a relative scale from '+' to '++++' with regards to scalability, stability, and relevance. More plus signs correspond to a higher grade. We remark that this grading reflects a qualitative assessment of the figures revealed by our tests, but that the plus signs are not directly comparable between columns. This means that one should not compare the algorithms by adding up all the plus signs directly.

For *scalability*, we have compared the cost of using larger input data-sets [144, Paper VI, Section 11.5]. It reflects both the rate of convergence and the memory requirements. The most scalable algorithms are PageRank[9] and S^2ProT, followed by HyS^2ProT, and then the others. One could argue that HITS should have a slightly higher grade because of its diminished data-set, but actual data does not agree with that; The data-set must not only be generated from the larger set but the generated set will sometimes have several eigenvalues of the same magnitude as well, which indicates that we could not give it a higher grade. Using the algorithm upgrade of Jeh and Widom [77, 78] would take Topic-sensitive PageRank up to the same level as HyS^2ProT.

Stability indicates how much the results are affected by removal of links and decreased sets of starting pages [144, Paper VI, Section 11.6.2]. We have also performed the same tests using HITS, and the results agree with the data reported by Ng et al. [111], i.e. HITS is quite unstable.

Assessment of perceived *relevance* has been one of the major parts of our work in both Paper V and Paper VI. We have chosen to group algorithms with similar results (see [143, Paper V] and [144, Paper VI, Section 11.6.1]) to the same grade, although there are minor differences within the groups. The higher the perceived relevance, the more plus signs are given.

Our conclusion is that the S^2ProT algorithm is among the best in all categories.

[9] But recall, unlike the other algorithms in the table, PageRank is not topic-sensitive.

Table 3.1: The scalability, stability and relevance of each algorithm on a scale from '+' to '++++'.

Algorithm	Scalability	Stability	Relevance
PageRank	++++	++++	+++
Personalized PageRank	++	++++	+
Topic-sensitive PageRank	++	+++	++++
HITS[a]	++	+	+++
ProT	++	++	+++
S^2ProT	++++	++++	++++
HyS^2ProT	+++	++++	++++

[a] Based partly on the results from other sources.

Figure 3.1: The problem with ambiguous search terms.
With permission from Krishna M. Sadasivam.

Chapter 4

Final Remarks

The work described in this thesis can be seen as a set of algorithms and their implementations that all operate on large quantities of discrete and textual data. The general idea is that the information is sampled, extracted, compiled and stored in a central data base that can be accessed by all tools that require the information.

Figure 4.1 on the next page illustrates our view of how such a set of tools should be interconnected. Documents to be included in the data base are processed to extract relevant information and possibly meta-data. Data propagation or implication can be performed if some documents lack sufficient data, e.g. [142, Paper IV].

Another possible source of data is a label bureau that provides clients with meta-data information about documents [14, 91, 102, 118, 138].

Multiple back-ends exist for the system as we envision it, one being a data mining system that mines for association rules. It uses CHiC [140, 141, Papers II–III] as the first step to create concept hierarchies, used in later association rule mining.

Another available back-end tool is a search engine that uses the topic-specific vectors created by our search engine algorithms [144, Paper VI] in order to facilitate searching. A prototype of a complete web-based search engine has been created and tested.

All in all, the algorithms and tools described in this thesis work together to provide answers that each of them would not be able to answer on their own. Most of the individual tools in Figure 4.1 on the following page already exist, but they have not been integrated into a framework or system.

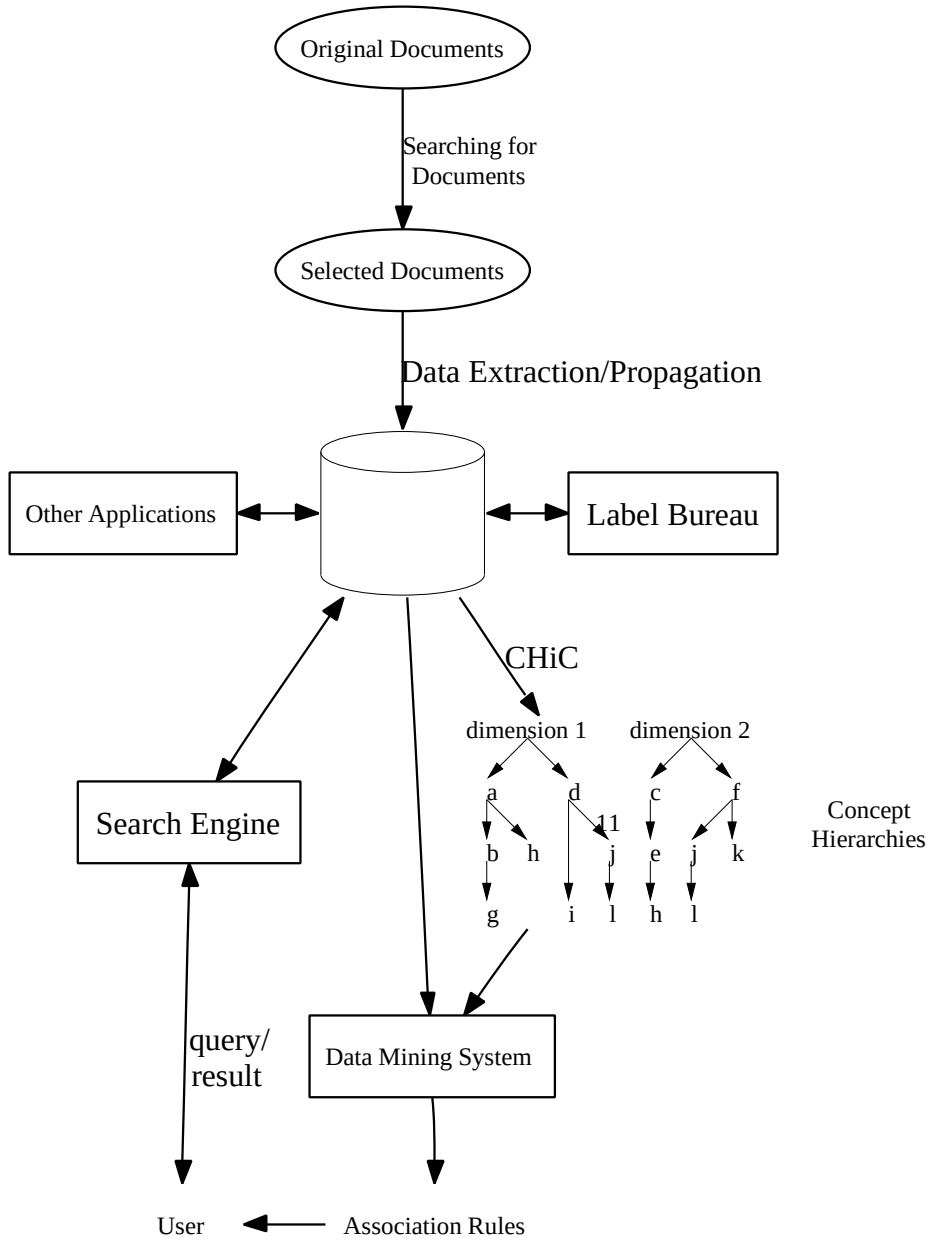


Figure 4.1: Overview of the application environment of our view of a data mining and management system for discrete and textual data.

Chapter 5

Bibliography

[1] ABDULJALEEL, N., AND QU, Y. Domain term extraction and structuring via link analysis. In Grobelnik et al. [63].

[2] ACHARYYA, S., AND GHOSH, J. A maximum entropy framework for higher order link analysis on directed graphs. In Donoho et al. [44].

[3] ADIBI, J., CHALUPSKY, H., GROBELNIK, M., MILIC-FRAYLING, N., AND MLADENIC, D., Eds. *Workshop on Link Analysis and Group Detection (LinkKDD2004)* (Seattle, WA, USA, Aug. 22, 2004). See [6, 22, 33, 37, 57, 79, 97, 106, 115, 116, 117].

[4] ADIBI, J., GROBELNIK, M., MILIC-FRAYLING, N., MLADENIC, D., AND PANTEL, P., Eds. *Workshop on Link Analysis: Dynamics and Static of Large Networks (LinkKDD2006)* (Philadelphia, PA, USA, Aug. 20, 2006). See [11, 16, 40, 70, 87, 123].

[5] ADIBI, J., GROBELNIK, M., MLADENIC, D., AND PANTEL, P., Eds. *Workshop on Link Discovery: Issues, Approaches and Applications (LinkKDD2005)* (Chicago, IL, USA, Aug. 21, 2005). See [12, 29, 50, 69, 98, 122, 127, 135].

[6] ADIBI, J., MORRISON, C. M., AND COHEN, P. R. Measuring confidence intervals in link discovery: A bootstrap approach. In Adibi et al. [3].

[7] AL HASAN, M., CHAOJI, V., SALEM, S., AND ZAKI, M. Link prediction using supervised learning. In Teredesai and Carley [130].

[8] AMERICAN LIBRARY OF CONGRESS. Electronic CIP: Cataloging in publication program. Web site, Oct. 06, 2008. Date visited given, `http://cip.loc.gov/`.

[9] AMSTERDAM INTERNET EXCHANGE. AMS-IX. Web site, Sept. 26, 2008. Date visited given, `http://www.ams-ix.net/`.

[10] ARASU, A., NOVAK, J., TOMKINS, A., AND TOMLIN, J. PageRank computation and the structure of the web: Experiments and algorithms. Tech. rep., IBM Almaden Research Center, Nov. 2001.

[11] ASUR, S., PARTHASARATHY, S., AND UCAR, D. An ensemble approach for clustering scale-free graphs. In Adibi et al. [4].

[12] BADIA, A., AND KANTARDZIC, M. Graph building as a mining activity: Finding links in the small. In Adibi et al. [5].

[13] BADIA, A., AND SKILLICORN, D., Eds. *Workshop on Link Analysis, Counterterrorism and Security (Adversarial Data Analysis)* (2008). See [17, 105].

[14] BAIRD-SMITH, A. *Jigsaw: An object oriented server*. The World Wide Web Consortium, Cambridge, Massachusetts, Feb. 1997.

[15] BATAGELJ, V., AND MRVAR, A. Density based approaches to network analysis: Analysis of reuters terror news network. In Donoho et al. [44].

[16] BECCHETTI, L., CASTILLO, C., DONATO, D., AND FAZZONE, A. Comparison of sampling techniques for web graph characterization. In Adibi et al. [4].

[17] BEER, E. A., PRIEBE, C. E., AND SCHEINERMAN, E. R. Torus graph inference for detection of localized activity. In Badia and Skillicorn [13].

[18] BEN-DOV, M., WU, W., FELDMAN, R., AND CAIRNS, P. A. Improving knowledge discovery by combining text-mining & link analysis techniques. In Cybenko and Srivastava [41].

[19] BERRY, M. W., AND BROWNE, M. Email surveillance using nonnegative matrix factorization. In Skillicorn and Carley [124], pp. 45–54.

[20] BERTINO, E., CATANIA, B., AND ZARRI, G. P. *Intelligent Database Systems*. Pearson Education, 2001.

[21] BHARAT, K., AND HENZINGER, M. R. Improved algorithms for topic distillation in a hyperlinked environment. In *SIGIR '98: Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval* (New York, NY, USA, 1998), ACM Press, pp. 104–111.

[22] BHATTACHARYA, I., AND GETOOR, L. Deduplication and group detection using links. In Adibi et al. [3].

[23] BIANCHINI, M., GORI, M., AND SCARSELLI, F. Inside PageRank. *ACM Trans. Inter. Tech. 5*, 1 (2005), 92–128.

[24] BLOEDORN, E., ROTHLEDER, N. J., DEBARR, D., AND ROSEN, L. Relational graph analysis with real-world constraints: An application in irs tax fraud detection. In Grobelnik et al. [63].

[25] BORGES, J., AND LEVENE, M. Ranking pages by topology and popularity within web sites. *World Wide Web 9*, 3 (2006), 301–316.

[26] BRIN, S., AND PAGE, L. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems 30*, 1–7 (1998), 107–117.

[27] BRODER, A. Z., LEMPEL, R., MAGHOUL, F., AND PEDERSEN, J. Efficient PageRank approximation via graph aggregation. *Information Retrieval 9*, 2 (Mar. 2006), 123–138.

[28] BUNTINE, W., LÖFSTRÖM, J., PERTTU, S., AND VALTONEN, K. Topic-specific link analysis using independent components for information retrieval. In Grobelnik et al. [63].

[29] CAI, D., SHAO, Z., HE, X., YAN, X., AND HAN, J. Mining hidden community in heterogeneous social networks. In Adibi et al. [5].

[30] CARACCIOLO, C., DE RIJKE, M., AND KIRCZ, J. Towards scientific information disclosure through concept hierarchies. In *Proceedings ELPUB 2002* (2002).

[31] CHAKRABARTI, D., ZHAN, Y., BLANDFORD, D., FALOUTSOS, C., AND BLELLOCH, G. NetMine: mining tools for large graphs. In Cybenko and Srivastava [41].

[32] CHAKRABARTI, S. Integrating the document object model with hyperlinks for enhanced topic distillation and information extraction. In *WWW '01: Proceedings of the 10th international conference on World Wide Web* (New York, NY, USA, 2001), ACM Press, pp. 211–220.

[33] CHAKRABARTI, S. Discovering links between lexical and surface features in questions and answers. In Adibi et al. [3].

[34] CHAKRABARTI, S., DOM, B. E., AND INDYK, P. Enhanced hypertext categorization using hyperlinks. In *Proceedings of SIGMOD-98, ACM International Conference on Management of Data* (Seattle, US, 1998), L. M. Haas and A. Tiwary, Eds., ACM Press, New York, US, pp. 307–318.

[35] CHAPANOND, A., KRISHNAMOORTHY, M. S., AND YENER, B. Graph theoretic and spectral analysis of enron email data. In Skillicorn and Carley [124], pp. 15–22.

[36] CHIDLOVSKII, B., RAGETLI, J., AND DE RIJKE, M. Wrapper generation via grammar induction. In *Proceedings European Conference on Machine Learning (ECML'2000)* (2000), LNCS, Springer.

[37] CHKLOVSKI, T., AND PANTEL, P. Path analysis for refining verb relations. In Adibi et al. [3].

[38] CLÉROT, F., AND NGUYEN, Q. A social network approach for the ranking of the autonomous systems of the internet. In Grobelnik et al. [63].

[39] CORSO, G. M. D., GULLI, A., AND ROMANI, F. Fast PageRank computation via a sparse linear system. In *Proceedings of Third Workshop on Algorithms and Models for the Web-Graph (WAW 2004)* (Rome, Italy, Oct. 16, 2004).

[40] CREAMER, G., AND STOLFO, S. A link mining algorithm for earnings forecast using boosting. In Adibi et al. [4].

[41] CYBENKO, G. V., AND SRIVASTAVA, J., Eds. *Workshop on Link Analysis, Counterterrorism and Security* (2004). See [18, 31, 49].

[42] DAVISON, B. D. Topical locality in the web. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval* (Athens, Greece, 2000), ACM Press, pp. 272–279.

[43] DAVISON, B. D. Unifying text and link analysis. In Grobelnik et al. [62].

[44] DONOHO, S., DYBALA, T., GROBELNIK, M., MILIC-FRAYLING, N., AND MLADENIC, D., Eds. *Proceedings of the 2003 Link Analysis for Detecting Complex Behavior (LinkKDD2003) Workshop* (Washington, DC, USA, Aug. 27, 2003). See [2, 15, 58, 85, 132].

[45] DUAN, Y., WANG, J., KAM, M., AND CANNY, J. A secure online algorithm for link analysis on weighted graph. In Skillicorn and Carley [124], pp. 71–81.

[46] DUBLIN CORE METADATA INITIATIVE. *The Dublin Core Metadata Element Set*. ISO Standard 15836 (2003) and ANSI/NISO Standard Z39.85-2007.

[47] DUNHAM, M. H. *Data Mining, Introductory and Advanced Topics*. Prentice Hall, inc., Englewood Cliffs, New Jersey, 2003.

[48] FAGNI, T., PEREGO, R., SILVESTRI, F., AND ORLANDO, S. Boosting the performance of web search engines: Caching and prefetching query results by exploiting historical usage data. *ACM Trans. Inf. Syst. 24*, 1 (2006), 51–78.

[49] FALOUTSOS, C., MCCURLEY, K. S., AND TOMKINS, A. Connection subgraphs in social networks. In Cybenko and Srivastava [41].

[50] FISSAHA ADAFRE, S., AND DE RIJKE, M. Discovering missing links in wikipedia. In Adibi et al. [5].

[51] FISSAHA ADAFRE, S., JIJKOUN, V., AND DE RIJKE, M. Link-based vs. content-based retrieval for question answering using wikipedia. In *Evaluation of Multilingual and Multi-modal Information Retrieval* (2007), pp. 537–540.

[52] FOX, J. Webless Literate Programming. *TUGboat 11*, 4 (Nov. 1990).

[53] FREITAG, D. Information extraction from HTML: Application of a general machine learning approach. In *AAAI/IAAI* (1998), pp. 517–523.

[54] FRIENDLY, L. The Design of Distributed Hyperlinked Programming Documentation. In *Proceedings of the 1995 International Workshop on Hypermedia Design* (June 1995).

[55] GANIZ, M. C., POTTENGER, W. M., AND YANG, X. Link analysis of higher-order paths in supervised learning datasets. In Teredesai and Carley [130].

[56] GETOOR, L. Link mining: a new data mining challenge. *SIGKDD Explor. Newsl. 5*, 1 (2003), 84–89.

[57] GILBERT, A. C., AND LEVCHENKO, K. Compressing network graphs. In Adibi et al. [3].

[58] GOLDENBERG, A., KUBICA, J., AND KOMAREK, P. A comparison of statistical and machine learning algorithms on the task of link completion. In Donoho et al. [44].

[59] GOLDENBERG, A., AND MOORE, A. Empirical bayes screening for link analysis. In Grobelnik et al. [62].

[60] GOLUB, G. H., AND VAN LOAN, C. F. *Matrix computations (3rd ed.)*. Johns Hopkins University Press, 1996.

[61] GREGG, D. G., AND WALCZAK, S. Adaptive web information extraction. *Commun. ACM 49*, 5 (2006), 78–84.

[62] GROBELNIK, M., MILIC-FRAYLING, N., AND MLADENIC, D., Eds. *Proceedings of the 2003 IJCAI Text-Mining & Link-Analysis Workshop* (Acapulco, Mexico, Aug. 9, 2003). See [43, 59, 93, 99].

[63] GROBELNIK, M., MILIC-FRAYLING, N., AND MLADENIC, D., Eds. *The AAAI-05 Workshop on Link Analysis (LinkAnalysis-2005)* (July 10 2005). See [1, 24, 28, 38, 107, 113, 136].

[64] HAN, J., AND KAMBER, M. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, Inc., San Francisco, California, 2001.

[65] HAVELIWALA, T., KAMVAR, S., KLEIN, D., MANNING, C., AND GOLUB, G. Computing PageRank using power extrapolation. Tech. rep., Stanford University, CA, USA, Oct. 18, 2003.

[66] HAVELIWALA, T. H. Efficient computation of PageRank. Tech. Rep. 1999-31, Stanford University Database Group, Oct. 18, 1999.

[67] HAVELIWALA, T. H. Topic-sensitive PageRank. In *Proceedings of the eleventh international conference on World Wide Web* (2002), ACM Press, pp. 517–526.

[68] HAVELIWALA, T. H., AND KAMVAR, S. D. The second eigenvalue of the google matrix. Tech. rep., Stanford University, Mar. 2003.

[69] HILL, S., AGARWAL, D., BELL, R., AND VOLINSKY, C. Tuning representations of dynamic network data. In Adibi et al. [5].

[70] HUANG, Z. Link prediction based on graph topology: The predictive value of generalized clustering coefficient. In Adibi et al. [4].

[71] HUMPHRIES, M., HAWKINS, M. W., AND DY, M. C. *Data Warehousing: Architecture and Implementation*. Prentice Hall PTC, Upper Saddle River, New Jersey, 1999.

[72] IEEE LEARNING TECHNOLOGY STANDARDS COMMITTEE. *IEEE 1484.12.1-2002 Standard for Learning Object Metadata*, 2002.

[73] INTERNET SYSTEMS CONSORTIUM, INC. ISC Domain Survey: Number of Internet Hosts. Web page, Sept. 25, 2008. Date visited given, `http://www.isc.org/index.pl?/ops/ds/host-count-history.php`.

[74] IPSEN, I. C. F., AND KIRKLAND, S. Convergence analysis of a PageRank updating algorithm by Langville and Meyer. *SIAM J. Matrix Anal. Appl. 27*, 4 (2006), 952–967.

[75] IPSEN, I. C. F., AND MEYER, C. D. Uniform stability of markov chains. *SIAM J. Matrix Anal. Appl. 15*, 4 (1994), 1061–1074.

[76] JAIN, A. K., MURTY, M. N., AND FLYNN, P. J. Data Clustering: A Review. *ACM Computing Surveys (CSUR) 31*, 3 (Sept. 1999), 264–323.

[77] JEH, G., AND WIDOM, J. Scaling personalized web search. Tech. Rep. 2002-12, Stanford University Database Group, 2002.

[78] JEH, G., AND WIDOM, J. Scaling personalized web search. In *WWW '03: Proceedings of the 12th international conference on World Wide Web* (New York, NY, USA, 2003), ACM Press, pp. 271–279.

[79] JONES, R. Semisupervised learning on small worlds. In Adibi et al. [3].

[80] KAM, J. B., AND ULLMAN, J. D. Global data flow analysis and iterative algorithms. *Journal of the ACM (JACM) 23*, 1 (1976), 158–171.

[81] KAMPS, J., MONZ, C., DE RIJKE, M., AND SIGURBJÖRNSSON, B. Approaches to robust and web retrieval. In *Proceedings TREC 2003* (2004), pp. 594–600.

[82] KAMVAR, S. D., HAVELIWALA, T. H., AND GOLUB, G. H. Adaptive methods for the computation of PageRank. Tech. rep., Stanford University, CA, USA, Apr. 2003.

[83] KAMVAR, S. D., HAVELIWALA, T. H., MANNING, C. D., AND GOLUB, G. H. Exploiting the block structure of the web for computing PageRank. Tech. rep., Stanford University, CA, USA, Mar. 4, 2003.

[84] KAMVAR, S. D., HAVELIWALA, T. H., MANNING, C. D., AND GOLUB, G. H. Extrapolation methods for accelerating PageRank computations. In *Proceedings of the Twelfth International World Wide Web Conference* (2003).

[85] KARGUPTA, H., LIU, K., DATTA, S., RYAN, J., AND SIVAKUMAR, K. Link analysis, privacy preservation, and random perturbations. In Donoho et al. [44].

[86] KEILA, P. S., AND SKILLICORN, D. B. Structure in the enron email dataset. In Skillicorn and Carley [124], pp. 55–64.

[87] KETKAR, N. S., HOLDER, L. B., AND COOK, D. J. Mining in the proximity of subgraphs. In Adibi et al. [4].

[88] KLEINBERG, J. Authoritative sources in a hyperlinked environment. In *Proc. of ACM-SIAM Symposium on Discrete Algorithms* (1998), pp. 668–677.

[89] KOLDA, T., AND BADER, B. The TOPHITS model for higher-order web link analysis. In Teredesai and Carley [130].

[90] KOSALA, AND BLOCKEEL. Web mining research: A survey. *SIGKDD: SIGKDD Explorations: Newsletter of the Special Interest Group (SIG) on Knowledge Discovery & Data Mining, ACM 2* (2000).

[91] KRAUSKOPF, T., MILLER, J., RESNICK, P., AND TREESE, W. *REC-PICS-labels-961031: PICS Label Distribution Label Syntax and Communication Protocols*. The World Wide Web Consortium, Cambridge, Massachusetts, Oct. 31, 1996.

[92] KROEKER, K. L. Finding diamonds in the rough. *Communcations of the ACM 51*, 9 (Sept. 2008), 11–13.

[93] KUBICA, J., MOORE, A., COHN, D., AND SCHNEIDER, J. cGraph: A fast graph-based method for link analysis and queries. In Grobelnik et al. [62].

[94] LANDAUER, T. K., FOLTZ, P. W., AND LAHAM, D. Introduction to latent semantic analysis. *Discourse Processes 25* (1998), 259–284.

[95] LANGVILLE, A. N., AND MEYER, C. D. Updating PageRank using the group inverse and stochastic complementation. Tech. Rep. CRSC-TR02-32, Center for Research in Scientific Computation, North Carolina State University, Raleigh, NC, USA, Nov. 2002.

[96] LEHMANN, S. Live and dead nodes. In Skillicorn and Carley [124], pp. 65–70.

[97] LESKOVEC, J., GROBELNIK, M., AND MILIC-FRAYLING, N. Learning sub-structures of document semantic graphs for document summarization. In Adibi et al. [3].

[98] LICAMELE, L., BILGIC, M., GETOOR, L., AND ROUSSOPOULOS, N. Capital and benefit in social networks. In Adibi et al. [5].

[99] LU, Q., AND GETOOR, L. Link-based text classification. In Grobelnik et al. [62].

[100] MADHAVAN, J., KO, D., KOT, Ł., GANAPATHY, V., RASMUSSEN, A., AND HALEVY, A. Google's deep web crawl. *Proc. VLDB Endow. 1*, 2 (2008), 1241–1252.

[101] MARSHALL, C. C. Making metadata: a study of metadata creation for a mixed physical-digital collection. In *DL '98: Proceedings of the third ACM conference on Digital libraries* (New York, NY, USA, 1998), ACM Press, pp. 162–171.

[102] MILLER, J., RESNICK, P., AND SINGER, D. *REC-PICS-services-961031: Rating Services and Rating Systems (and Their Machine Readable Descriptions)*. The World Wide Web Consortium, Cambridge, Massachusetts, Oct. 31, 1996.

[103] MILLER, J. C., RAE, G., SCHAEFER, F., WARD, L. A., LOFARO, T., AND FARAHAT, A. Modifications of Kleinberg's HITS algorithm using matrix exponentiation and web log records. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval* (New Orleans, Louisiana, United States, 2001), ACM Press, pp. 444–445.

[104] MOLER, C. B. Cleve's corner: The world's largest matrix computation: Google's pagerank is an eigenvector of a matrix of order 2.7 billion. Technical note, The MathWorks, Inc., 3 Apple Hill Drive, Natick, MA 01760-2098, USA, Oct. 2002.

[105] MOON, I.-C., CARLEY, K. M., AND LEVIS, A. H. Vulnerability assessment on adversarial organization: Unifying command and control structure analysis and social network analysis. In Badia and Skillicorn [13].

[106] MUKHERJEE, M., AND HOLDER, L. B. Graph-based data mining on social networks. In Adibi et al. [3].

[107] MURRAY, K., HARRISON, I., LOWRANCE, J., RODRIGUEZ, A., THOMERE, J., AND WOLVERTON, M. Pherl: An emerging representation language for patterns and hypotheses and evidence. In Grobelnik et al. [63].

[108] MUSLEA, I., MINTON, S., AND KNOBLOCK, C. A. Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems 4*, 1/2 (2001), 93–114.

[109] NESTOROV, S., ABITEBOUL, S., AND MOTWANI, R. Extracting schema from semistructured data. In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data* (New York, NY, USA, 1998), ACM Press, pp. 295–306.

[110] NG, A. Y., ZHENG, A. X., AND JORDAN, M. Link analysis, eigenvectors, and stability. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)* (2001).

[111] NG, A. Y., ZHENG, A. X., AND JORDAN, M. Stable algorithms for link analysis. In *Proceedings of the Twenty-fourth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Sept. 2001).

[112] PAGE, L., BRIN, S., MOTWANI, R., AND WINOGRAD, T. The PageRank citation ranking: Bringing order to the web. Tech. rep., Stanford Digital Library Technologies Project, 1998.

[113] PAPERNICK, N., AND HAUPTMANN, A. G. Summarization of broadcast news video through link analysis of named entities. In Grobelnik et al. [63].

[114] PARREIRA, J. X., CASTILLO, C., DONATO, D., MICHEL, S., AND WEIKUM, G. The Juxtaposed approximate PageRank method for robust PageRank approximation in a peer-to-peer web search network. *The International Journal on Very Large Data Bases 17*, 2 (Mar. 2008), 291–313.

[115] PIOCH, N. J., HUNTER, D., WHITE, J. V., KAO, A., BOSTWICK, D., AND JONES, E. K. Multi-hypothesis abductive reasoning for link discovery. In Adibi et al. [3].

[116] RAGHAVAN, H., ALLAN, J., AND MCCALLUM, A. An exploration of entity models, collective classification and relation description. In Adibi et al. [3].

[117] RESIG, J., DAWARA, S., HOMAN, C. M., AND TEREDESAI, A. Extracting social networks from instant messaging populations. In Adibi et al. [3].

[118] RESNICK, P., AND MILLER, J. PICS: Internet Access Controls Without Censorship. *Communications of the ACM 39*, 10 (1996), 87–93.

[119] SALTON, G., WONG, A., AND YANG, C. S. A vector space model for automatic indexing. *Communications of the ACM 18*, 11 (Nov. 1975), 613–620.

[120] SANDERSON, M., AND CROFT, B. Deriving concept hierarchies from text. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval* (Berkeley, California, United States, 1999), ACM Press, pp. 206–213.

[121] SCOTT, J. A. An Arnoldi code for computing selected eigenvalues of sparse, real, unsymmetric matrices. *ACM Trans. Math. Softw. 21*, 4 (1995), 432–475.

[122] SHETTY, J., AND ADIBI, J. Discovering important nodes through graph entropy - the case of enron email database. In Adibi et al. [5].

[123] SIDIROPOULOS, A., KATSAROS, D., AND MANOLOPOULOS, Y. Generalized h-index for revealing latent facts in social networks of citations. In Adibi et al. [4].

[124] SKILLICORN, D., AND CARLEY, K., Eds. *Workshop on Link Analysis, Counterterrorism and Security* (2005). See [19, 35, 45, 86, 96].

[125] SODERLAND, S. Learning information extraction rules for semi-structured and free text. *Machine Learning 34*, 1-3 (1999), 233–272.

[126] SODERLAND, S., FISHER, D., ASELTINE, J., AND LEHNERT, W. CRYSTAL: Inducing a conceptual dictionary. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (San Francisco, 1995), C. Mellish, Ed., Morgan Kaufmann, pp. 1314–1319.

[127] STOILOVA, L., HOLLOWAY, T., MARKINES, B., MAGUITMAN, A., AND MENCZER, F. Givealink: Mining a semantic network of bookmarks for web search and recommendation. In Adibi et al. [5].

[128] TAN, P.-N., AND KUMAR, V. Mining indirect associations in web data. In *Proceedings of the 2002 Mining Log Data Across All Customer TouchPoints (WebKDD2001) Workshop* (Aug. 2001).

[129] TAN, P.-N., STEINBACH, M., AND KUMAR, V. *Introduction to Data Mining*. Addison-Wesley, Reading, Massachusetts, 2006.

[130] TEREDESAI, A., AND CARLEY, K., Eds. *Workshop on Link Analysis, Counterterrorism and Security* (2006). See [7, 55, 89].

[131] TIAN, Y., HUANG, T., AND GAO, W. A web site mining algorithm using the multiscale tree representation model. In *Proceedings of the 2003 Webmining as a Premise to Effective and Intelligent Web Applications (WebKDD'2003) Workshop* (Aug. 2003), pp. 83–92.

[132] TIAN, Y., MEI, Z., HUANG, T., AND GAO, W. Incremental learning for interaction dynamics with the influence model. In Donoho et al. [44].

[133] TWO CROWS CORPORATION. Introduction to Data Mining and Knowledge Discovery, Third Edition. Potomac, MD, USA, 2005.

[134] VAN HEESCH, D. *doxygen: Manual for version 1.5.7.1*, 2008. Available for download from `ftp://ftp.stack.nl/pub/users/dimitri/doxygen_manual-1.5.7.1.pdf.zip`.

[135] WANG, X., MOHANTY, N., AND MCCALLUM, A. Group and topic discovery from relations and text. In Adibi et al. [5].

[136] WOLVERTON, M., AND THOMERE, J. The role of higher-order constructs in the inexact matching of semantic graphs. In Grobelnik et al. [63].

[137] XIAO, Y., AND DUNHAM, M. H. Efficient mining of traversal pattern. *Data and Knowledge Engineering 39*, 2 (Nov. 2001), 191–214.

[138] ÅGREN, O. Reuse via the World Wide Web: How to Find the Software Required for Reuse. Master's thesis, Umeå University, Umeå, Sweden, Dec. 1998. UMNAD 242.98.

[139] ÅGREN, O. ALGEXT - an ALGorithm EXTractor for C Programs. Tech. rep., Umeå University, Umeå, Sweden, May 2001. UMINF 01.11, ISSN 0348-0542, Paper I on page 41.

[140] ÅGREN, O. Automatic Generation of Concept Hierarchies for a Discrete Data Mining System. In *Proceedings of the International Conference on Information and Knowledge Engineering (IKE '02)* (Las Vegas, Nevada, USA, June 24-27, 2002), pp. 287–293. Paper II on page 59.

[141] ÅGREN, O. CHIC: A Fast Concept HIerarchy Constructor for Discrete or Mixed Mode Databases. In *Proceedings of the Fifteenth International Conference on Software Engineering and Knowledge Engineering (SEKE'03)* (San Francisco, California, USA, July 1-3, 2003), pp. 250–258. Paper III on page 77.

[142] ÅGREN, O. Propagation of Meta Data over the World Wide Web. In *Proceedings of the International Conference on Internet Computing (IC '03)* (Las Vegas, Nevada, USA, June 23-26, 2003), vol. 2, pp. 670–676. Paper IV on page 103.

[143] ÅGREN, O. Assessment of WWW-Based Ranking Systems for Smaller Web Sites. *INFOCOMP Journal of Computer Science 5*, 2 (June 2006), 45–55. Paper V on page 117.

[144] ÅGREN, O. S^2ProT: Rank Allocation by Superpositioned Propagation of Topic-Relevance. *International Journal of Web Information Systems 4*, 4 (2008), 416–440. Paper VI on page 141.

I

Chapter 6

ALGEXT — an ALGorithm EXTractor for C Programs

Ola Ågren

May 30, 2001

UMINF 01.11
ISSN-0348-05422

Abstract

ALGEXT *is a program that extracts strategic/block comments from C source files to improve maintainability and to keep documentation consistent with source code. This is done by writing the comments in the source code in what we call* extractable algorithms*, describing the algorithm used in the functions.*

ALGEXT *recognizes different kinds of comments:*

- *Strategic comments are comments that proceed a block of code, with only whitespace preceding it on the line,*
- *Tactical comments are comments that describes the code that precedes it on the same line,*
- *Function comments are comments immediately preceding a function definition, describing the function,*
- *File comments are comments at the head of the file, before any declarations of functions and variables, and finally*
- *Global comments are comments within the global scope, but not associated with a function.*

Only strategic comment are used as basis for algorithm extraction in ALGEXT.

The paper discusses the rationale for ALGEXT *and describes its implementation and usage. Examples are presented for clarification of what can be done with* ALGEXT.

Our experience shows that students who use ALGEXT *for preparing their assignments tend to write about 66% more comments than non-*ALGEXT *users.*

6.1 Introduction

One common problem in the software industry is to keep the documentation up-to-date with the source code, especially when using evolutionary prototyping [6, 7, 8] or extreme programming [1]. A number of approaches for handling the problem have been proposed, including literate programming [5] and processes that require the documentation to be done before coding. In this report we propose another approach, that of extractable algorithms contained within a program.

An *extractable algorithm* is a description of the algorithm in one form or another, contained within the source code. It must not interfere in any way with the compiler/interpreter when compiling/running the program. It must also be easily extractable from within the program, in order to update the documentation after each update of the source code.

The idea to keep both the source and documentation in the same place is not new. A number of approaches to keep source and documentation conceptionally as close as possible have been devised.

Literate programming [4] requires that both source code and documentation are contained in a special type of file called a Web[1] source document. A set of programs called `tangle` and `weave` are then used to extract source code for the compiler and the typesetting environment, respectively.

A simplified version of literate programming is c~~web~~ [2]. It requires no external program, since the source code is the same for both the C compiler and the TEX/LATEX typesetting environment.

Another approach that has been used (in e.g., the `EXCO word` text editor) is the hierarchical approach. The source code is added as the bottom-most level of the documentation, and a special program extracts that level before compiling the program.

Other related works are those that extract information from the source code directly, e.g.

- `cextract`[2] by Adam Bryant extracts function comments, function signatures and optionally file comments (see Section 6.2) from a set of C files. Furthermore, `cextract` can transform this into C header files, pure text (e.g. Figure 6.8 on page 54) or `nroff`/`troff`/`groff` input.
- Javadoc [3] is a tool that parses the declarations and documentation comments in a set of Java source files. Its output is a set of HTML pages describing the classes, inner classes, interfaces, constructors, methods, and fields.

[1] Not to be mistaken for the WWW.

[2] Source code available from `http://dev.w3.org/cvsweb/Amaya/tools/cextract-1.7/`.

Javadoc requires a special set of tags within the comments, otherwise it extracts very sparse documentation.

Our approach to extractable algorithms can be used with any type of mark-up language (see examples in Section 6.5), as long as it can be embedded in the comments of the programming language in use. The current implementation handles ANSI C with some minor restrictions (see Section 6.3) with respect to source code formatting.

6.2 Contents of a C File

A normal C file is just an ordered set of comments, declarations and function definitions. For a C compiler, a comment is something that can be removed from the code since it doesn't contain syntactic or semantic elements to be used when translating the C source code into object code. This means that a user can add any type of textual information in the comments, so that the any user of the source code[3] can more easily grasp what the code does.

All of the comments within a source file can be classified into five different types, mainly depending on their placement in the source file.

- *File comments* are comments at the head of the file, before any declarations of functions and variables. File comments usually contain information that is true for the entire file, e.g., name, description of content, author name and change history.
- *Function comments* are comments immediately preceding a function definition, describing the function. Function comments usually contain information about the immediately following function, e.g. name, description of functionality, parameter descriptions, return values, etc.
- *Global comments* are comments within the global scope, but not associated with a function.
- *Strategic comments* are comments within a function just before a block of code, with only whitespace preceding it on the line. Strategic comments describes the block of comment that follows and are therefore also called *block comments*. This is the only type of comment extracted by ALGEXT.
- *Tactical comments* are comments within a function that describe the code that precedes it on the same line.

The different types of comments can be seen in Figure 6.1 on the facing page.

[3] Other users or the original author at a later time.

Figure 6.1: Sample code showing the different comment types available.

```
/* This is a file comment. */

/* This is a global comment. */
typedef int rettype;

/* This is a function comment. */
rettype main()
{
    /* This is a strategic comment */
    return 0;   /* This is a tactical comment */
}
```

6.3 Source Code Requirements

The current implementation of ALGEXT does not contain a full parser for the C language (in much the same way as `cextract` and `cflow`[4]), which puts some restrictions on the code that is to be handled:

- There must be no whitespace between the function name and the parenthesis that surrounds the formal arguments. As an example,

  ```
  int main(int argc, char **argv)
  ```

  will be parsed correctly, while

  ```
  int main (int argc, char **argv)
  ```

  will not be recognized as a function header. We do not see this as a major problem, since the use of whitespace between the function name and the opening parenthesis is not that common.

- C++ style comments ('//...') are not handled, only the old style of C comments ('/* ... */') are accepted. A future version of ALGEXT will handle these as well.

- Lines cannot be more than 1023 characters long. This is controlled by a constant in the source code and thus easily changed.

6.4 Implementation

The functionality of ALGEXT can be described in one sentence:

> If a comment starts in a function with nothing but whitespace preceding it on the line, write the entire comment (with preceding whitespace, but without the comment tokens) to standard output together with all function names.

ALGEXT is written in ANSI C, using only POSIX-compliant input/output functions to be portable to any platform. It is written with the intent of being usable as a filter in a UNIX command pipeline. It read its input from standard input and writes on standard output. This makes it easy to use from within shell scripts, and we normally call it using the Bourne shell script in Figure 6.6 on page 53.

[4] `cflow` generates a C flow graph. It analyses a collection of C, YACC, LEX, assembler, and object files, and attempts to build a graph charting the external references. `cflow` is available on most platforms.

6.5 Examples

ALGEXT can be used to extract any type of textual information. The most basic type of information that can be extracted is normal ASCII text, as in Section 6.5.1. A somewhat more elaborate algorithm description can be found in Section 6.5.2, using LaTeX comments directly in the text.

6.5.1 Basic comments

Given the program in Figure 6.2, ALGEXT will produce the output in Figure 6.3.

Figure 6.2: Sample C code with basic comments.

```
int fac(int n)
{
    /* fac(n) = */
    if (n > 0)
        /* n * fac(n - 1),      if n > 0 */
        return n * fac(n - 1);
    else
        /* 1,                   otherwise */
        return 1;
}
```

Figure 6.3: Sample algorithm extracted using ALGEXT.

```
fac:
     fac(n) =
         n * fac(n - 1),        if n > 0
         1,                     otherwise
```

6.5.2 LaTeX comments

The same file could just as easily be commented using a slightly more elaborate scheme, e.g. LaTeX code. Given the program in Figure 6.4, ALGEXT will produce the output in Figure 6.5.

Figure 6.4: Sample C code with LaTeX-style comments.

```
int fac(int n)
{
    /* \begin{equation} \text{fac}(n) = \begin{cases} */
    if (n > 0)
        /* n * \text{fac}(n - 1) & \text{if $n > 0$}, \\ */
        return n * fac(n - 1);
    else
        /* 1                     & \text{otherwise} */
        return 1;
    /* \end{cases} \label{fac} \end{equation} */
}
```

Figure 6.5: LaTeX style algorithm extracted using ALGEXT.

```
fac:
     \begin{equation} \text{fac}(n) = \begin{cases}
         n * \text{fac}(n - 1) & \text{if $n > 0$}, \\
         1                     & \text{otherwise}
     \end{cases} \label{fac} \end{equation}
```

This will (typeset by LaTeX) yield the algorithm description for fac as seen in equation 6.1.

fac:

$$\text{fac}(n) = \begin{cases} n * \text{fac}(n-1) & \text{if } n > 0, \\ 1 & \text{otherwise} \end{cases} \tag{6.1}$$

6.6 Discussion

ALGEXT has been used extensively since 1996 primarily by students at the Department of Computing Science, Umeå University. It has been used to document algorithms for both course assignments (see Appendix 6.C) and bigger projects (most notably the `sawit` web server [9]). Almost all comments have been positive. Most requests for updates have been in the form of support for additional languages.

We have found that users of ALGEXT tend to write more lines with comments in their functions than non-users (approximately 66% more, see Appendix 6.C). We see this as an indication that the ALGEXT users tend (on average) to put more thought into commenting their code.

When comparing ALGEXT with other programs, we see that:

- `cextract` and Javadoc [3] work on another abstraction level, that of file/class overview rather than function view. Algorithms can be added in function comments, but it increases the distance between algorithm and source code.
- Literate programming [5, 2] forces the user into a certain programming model — one file equals one document equals one source file. Moreover, documentation generated by literate programs (rather than c~~web~~) tend to be broken up into small fragments of code with links between them that are not that easy to follow.
- The hierarchical approach forces the user to use a special set of tools in order to edit their code, instead of their favorite editor.

We have moreover found that the *conceptual distance*[5] between code and algorithm has a great impact on documentation. The larger the conceptual distance between source code and documentation, the higher the possibility of discrepancy between them. This means that if the documentation resides on another system it takes an effort to change to that system in order to update the documentation, while documentation that is within the same logical scope as the source code do not require as much effort to keep up to date. Using ALGEXT will shorten the conceptual distance between source code and documentation, since both resides in the same logical space as seen by the user.

[5] Conceptual distance is how far the two logical scopes involved are from each other. The more that have to be changed (language, viewpoint, formality, system, etc.) the larger the conceptual distance between the two.

6.7 References

[1] BECK, K. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, Reading, Massachusetts, Oct. 15, 1999.

[2] FOX, J. Webless Literate Programming. *TUGboat 11*, 4 (Nov. 1990).

[3] FRIENDLY, L. The Design of Distributed Hyperlinked Programming Documentation. In *Proceedings of the 1995 International Workshop on Hypermedia Design* (June 1995).

[4] KNUTH, D. E. Literate Programming. *The Computer Journal*, 27 (1984).

[5] KNUTH, D. E. *Literate Programming*. No. 27 in CSLI Lecture Notes. Center for the Study of Language and Information, Stanford, CA, 1992.

[6] PREECE, J., ET AL. *Human-computer interaction*. Addison-Wesley, Reading, Massachusetts, 1994.

[7] PRESSMAN, R. S. *Software Engineering, A Practitioner's Approach*, european adaption, third ed. McGraw-Hill, London, England, 1994.

[8] SOMMERVILLE, I. *Software Engineering*, fourth ed. Addison-Wesley, Reading, Massachusetts, 1992.

[9] ÅGREN, O. Reuse via the World Wide Web: How to Find the Software Required for Reuse. Master's thesis, Umeå University, Umeå, Sweden, Dec. 1998. UMNAD 242.98.

6.A Users' Guide

Since ALGEXT uses only POSIX-compliant input/output and is written entirely in ANSI C it should be portable to any operating system that contains an ANSI C compiler or the GNU C compiler. The source code to parse is read from standard input and the extracted comments are presented on standard output, which makes ALGEXT suitable as a filter in a UNIX command pipeline.

The extracted comments are given in the form below.

Function name:
extracted comments, indented as in source code

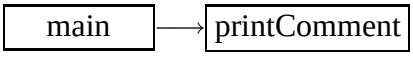
Most users of ALGEXT call it indirectly using the Bourne shell script given in Figure 6.6.

Figure 6.6: Sample Borne-shell wrapper for ALGEXT.

```
#!/bin/sh
if [ ! -x /Home/staff/ola/bin/‘arch‘/algext ]
then
        echo "Failed to find algorithm extractor for " ‘arch‘
        exit 1;
fi

for i in $*
do
        name=‘echo $i | sed ’s/\.c/.alg/’‘
        echo "Extracting $i to $name"
        /Home/staff/ola/bin/‘arch‘/algext < $i > $name
done
```

Figure 6.7: The call graph of ALGEXT.



6.B System Documentation

6.B.1 System Description

The call graph of ALGEXT is seen in Figure 6.7 on the preceding page and the function description is given in Figure 6.8. The functional description is extracted using the `cextract` program.

Figure 6.8: The function description of ALGEXT.

```
Function: main
File:     algext.c

/*
 * main:
 * This program extracts the names of routines and
 * all block comments inside the routines
 * arguments:   argc is the number of arguments
 *              argv is a list of the arguments
 * returns:     0 if all went OK, 1 otherwise
 * calls:       printComment, ERR
 */
int main ( int argc, char *argv[] );


Function: printComment
File:     algext.c

/*
 * printComment:
 * This routine handles one comment
 * argument:    i is the position on the line
 *              print is if we are to print the comment
 * returns:     the next position on the line
 * calls:       ERR
 */
int printComment ( int i, int print );
```

6.B.2 Algorithms

The algorithms of ALGEXT is given in algorithms 6.1 and 6.2 on the next page. These algorithms are extracted from the source code of ALGEXT, by ALGEXT.

ALGORITHM 6.1 (MAIN)
Input — a C source file
Output — the extracted comments
proc *main()* ≡

```
⌜  foreach line ∈ input do
       foreach character ∈ line do
           if character is ...
               "\" ∨ "'" then while ¬ end of character/string constant do
                                   while character ∈ constant do
                                       if character = "\" then Skip
                                       Skip
                                   done
                                   if EndOfLine without EndOfConstant then
                                       if character = "\"
                                           then Continue on next line
                                           else Give error message and quit
                               done
               "{" then
                         Go to higher level of blocks
                         if at start of a function
                             then Present function name
               "}" then if inside a block
                             then Go to lower level of blocks
                             else Give error message and quit
               "/" then if start of comment
                             then Handle this comment
                             else Ignore
               "(" then Stop adding characters to function name
               otherwise if ¬ whitespace
                             then if adding characters to a name of a function
                                      then Add character
                             else Empty name of function
       done
   done
⌞  Tell invocator that we are finished
```

ALGORITHM 6.2 (PRINTCOMMENT)
Input — a C source comment(/line)
Output — the extracted comments
proc *printComment()* ≡
⌜ **if** *this comment is to be printed*
then *Present all characters up to the start of the comment*
while ¬ *EndOfComment* **do**
while ¬ *(EndOfLine* ∨ *EndOfComment)* **do**
if *this comment is to be printed*
then *Print this character*
done
if *the comment terminates before EndOfLine*
then if *this comment is to be printed*
then *Print an EndOfLine*
Continue with next character
The comment was not ended on this line, continue with next
⌟ **done**

Table 6.1: Strategic comments of student source code.

Number of strategic comment lines per function	Value
Minimum (x_{min})	1
Median (x_{median})	2
Mean ($\overline{x}$)	3.8362
Maximum (x_{max})	26
Standard deviation (σ)	4.2078
Variance (σ^2)	17.7059

6.C Comment Comparison Between Users and Non-Users of ALGEXT

Users of ALGEXT tend to write approximately 66% more comment lines per function than the other students (see tables 6.1 on the facing page and 6.2). These numbers are generated from the source code written by students of the course in Operating Systems at the Department of Computing Science, University of Umeå anno 1998. We believe that the increase is mainly due to the extra planning done when writing the comments.

In fact, some students have reported that they write the entire algorithm as comments within the function before adding any code.

We must also report that the highest number of comment lines per function by the non-ALGEXT users is a function that has only 30 lines of statements, so these comments are more tactical than strategic in their nature. The statistics would show even more discrepancy if this function is removed.

Table 6.2: Strategic comments of source code by students using extractable algorithms.

Number of strategic comment lines per function	Value
Minimum (x_{min})	2
Median (x_{median})	5.5
Mean ($\bar{x}$)	6.3571
Maximum (x_{max})	16
Standard deviation (σ)	3.5433
Variance (σ^2)	12.5549

Figure 6.9 on the following page contains examples of extracted algorithms. They are extracted from the source code of a process tracer written by a student. The program shows the same information about a traced process as does `ps` and (`s`)`trace`.

Figure 6.9: The algorithm description for a process tracer, extracted from source code made by a student.

```
openFile:
   Generate correct name of file to open and return it
     Failed to open with full authority, try minimal version

traceProc:
   Forever do the following
     Check if we are to change process to trace
       We are, do we have a file to read from?
         If yes, read file and change process
         If not, continue tracing the old process
     Read process status
     If status has changed since last loop ...
       Send to other process

printTrace:
   Print header line
   Forever do the following ...
     Get one information record from tracing process
     If data has changes since last time we checked ...
       Print information including possible system call

sigHandler:
   If signal is ...
     SIGHUP:
       Make sure that the meta proc file is read
     SIGPIPE:
       We'd better die, since we have lost connection to the other process

main:
   Parse argument line
   Get pid for process to work with
   Open proc file
   Prepare for IPC
   Set up signal handler
   Create another process
     This is parent, remove READ end in pipe
       Redirect all output to child and follow process
     This is child, remove WRITE end in pipe
       Read from parent and present in standard format
   End execution here and now
```

II

Chapter 7

Automatic Generation of Concept Hierarchies for a Discrete Data Mining System

Paper appears with kind permission from CSREA Press.

Abstract

In this paper we propose an algorithm for automatic creation of concept hierarchies from discrete databases and datasets. The reason for doing this is to accommodate later data mining operations on the same set of data without having an expert create these hierachies by hand.

We will go through the algorithm thoroughly and show the results from each step of the algorithm using a (small) example. We will also give actual execution times for our prototype for non-trivial example data sets and estimates of the complexity of the algorithm in terms of the number of records and the number of distinct data values in the data set.

Keywords: Data Mining, Data Preprocessing, Hierarchy Generation

7.1 Introduction

Concept hierarchies are descriptors over data sets, in such a way that all records in the corresponding data sets can be described by the hierarchies. Concept hierarchies are typically used in retrieval systems [14, 13, 3] and for data mining [6]. Each facet or aspect of the records has a corresponding hierarchy, often in the form of a tree. An example of such a hierarchy that describes a C source code could be $Imperative \rightarrow C \rightarrow ANSI$ for the language facet and $Ordered \rightarrow Tree \rightarrow Binary$ for the data type facet. Creating a concept hierarchy manually is very time consuming and relies on the understanding of the actual data at hand. Creation of concept hierarchies can be automated (as in this work). There are unfortunately still some doubts about the soundness of the generated facets/dimensions, especially since no exact rules for what is a good concept hierarchy exists today.

There exists a large number of techniques for gathering information from large sets of data. These techniques include such diverse methods as machine learning, decision trees [16], fuzzy sets/neural networks [5], and data mining [6]. All but the last one have been used successfully to propose class membership of each record if given new data. Most of them have found their own niche even though the techniques are meant to be of general use, e.g. fuzzy sets/neural networks are often used in medical diagnostics systems, for example to indicate existence/non-existence of a medical syndrome [8].

Data mining has some special properties. The main goal of a data mining system is to find and extract correlations between different properties or aspects of records in given data. Such systems have been especially good at finding data that are closely related in a large set of data, so called clustering [7]. A data mining system will report and/or visualize all relationships found so that the user can understand and make use of the extracted information. Typical uses of data mining includes finding patterns in sales data in the form of *buyers of product A tend to buy product B as well* or *buyers under X years of age with an income of at least Y tend to buy C*, so that it is possible to pinpoint whom to direct an advertisement to [6].

A crusial part of a data mining system is the concept hierarchy that describes all aspects of the data in a number of layers, where lower levels correspond to a low data abstraction and higher levels correspond to a higher abstraction level, i.e. a more generalized and summarized data. Generating a concept hierarchy for a dataset is a time consuming task, especially if the nature of the dataset is not known beforehand. In this paper we describe an algorithm for automatic creation of concept hierarchies for discrete datasets (e.g. each data record contains one or more of a certain number of keywords/tokens). An example of such a discrete dataset is given in Figure 7.1 on the next page.

```
01 Makefile:   make commands text
02 clean:      Bourne shell script text
03 combinator: ELF 32-bit MSB executable, SPARC, version 1, dynamically linked, not stripped
04 db:         directory
05 fest.txt:   International language text
06 fil.aux:    LaTeX auxiliary file
07 fil.dot:    ASCII text
08 fil.eps:    PostScript document text conforming at level 2.0
09 fil.log:    TeX transcript text
10 fil.tex:    LaTeX 2e document text
11 input:      English text
12 lex.yy.c:   C program text
13 main.c:     C program text
14 main.o:     ELF 32-bit MSB relocatable, SPARC, version 1, not stripped
15 words.l:    lex description text
16 words.o:    ELF 32-bit MSB relocatable, SPARC, version 1, not stripped
```

Figure 7.1: A short example of discrete meta-data, in this case output of the "file" command.

7.2 Definitions

Given $k_i \in \{keywords\}$ (the keywords in the system) and $\sigma[k_i] \in \mathbb{P}\{records\}$ (the set of records in the database that contain keyword k_i), we can define the concepts used in this work.

Two or more keywords that always appear together are **keyword equivalent**, i.e.

$$k_i =_k k_j \iff \sigma[k_i] = \sigma[k_j], \quad \text{where } k_i \neq k_j. \tag{7.1}$$

This implies that they may be **folded** into one keyword, i.e. the name of the latter is added as a synonym of the first and all references to the latter are removed or ignored. Such removed keywords are said to belong to k_is **family**, i.e.

$$k_i =_k k_j \Longrightarrow k_i, k_j \in fam_i, fam_i \in \mathbb{P}\{keywords\}. \tag{7.2}$$

Formally, **subsumption** is defined as an implicit subset/superset relationship between the interpretations of the two concepts [2]. This means that if a keyword k_i never appears in a record without k_j appearing, but not vice versa, then k_i is **subsumed** by k_j.

$$k_i <_k k_j \iff \sigma[k_i] \subset \sigma[k_j] \tag{7.3}$$

A keyword k_i that is not subsumed by another keyword is said to be a **vector**, and is used as the bases in the concept hierarchies.

7.3 The Algorithm

The algorithm given here is straight forward, although some of the implementation details can be anything but trivial to implement. The only requirement is that the keywords must have a consistent enumeration order, but the different parts of the algorithm are independent of different implementations of sets, etc.

Further information about each part of the algorithm, including cost estimations based on code complexity, are given in Section 7.5.

0. Create database and index;

   This step has to be done, but is beyond the scope of this article.

1. Read index data;

   proc *ReadInvertedFiles*() $\equiv$
   ⌜ **foreach** $k \in \{keywords\}$ **do**
   $\sigma[k] \leftarrow$ *read inverted DB File*$_k$;
   ⌞ **od.**

   ReadInvertedFiles reads the index/inverted data files [10] from the disc, or must generate them (at some run time cost) if they do not exist. In this algorithm description (and in the cost estimation in Section 7.5.1) we expect them to be on disc before execution.

2. Find and fold keyword families;

   proc *FindAndFoldFamilies*() $\equiv$
   ⌜ **while** $\exists k_i, k_j \in \{keywords\} \bullet i < j \wedge \sigma[k_i] = \sigma[k_j]$ **do**
   $fam_i \leftarrow fam_i \cup fam_j$; ($k_j$ is always a member of fam_j)
   remove k_j *from computations*;
   ⌞ **od.**

   FindAndFoldFamilies finds keywords that are keyword equivalent, adds the latter to the formers family and removes the latter (according to the keyword enumeration order) from further computations.

3. Find all subsets and subsumptions;

   proc *FindSubsets*() $\equiv$
   ⌜ **while** $\exists k_i, k_j \in \{keywords\} \bullet \sigma[k_i] \subset \sigma[k_j]$ **do**
   $k_i <_k k_j$;
   ⌞ **od.**

   FindSubsets finds all cases where one keyword is subsumed by another, while ignoring folded keywords altogether.

   The reason for performing family folding before finding subsets is to prune the solution space as quickly as possible, since all keywords in a family will subsume/be subsumed by exactly the same keywords.

4. Find all vector nodes;

   proc *FindVectorNodes*() $\equiv$
   ⌜ *vectors* $\leftarrow$ empty array; (Sorted in decreasing $\mid k \mid$ order)
   foreach $k \in \{keywords\}$ **do**
   if $\nexists k_j \bullet k <_k k_j$
   then Insert k in *vectors*;
   fi
   ⌞ **od.**

   FindVectorNodes finds all vector keywords in the set of keywords. The vector keywords are those that are not subsumed by any other keyword.

   All found vector keywords will be added to an array in a decreasing order according to the cardinality of the keywords. The order is important for the heuristics of step 5 on the facing page.

5. Group vectors into different dimensions/facets;

   proc *ColourVectorNodes*() $\equiv$
   ⌜ $dim \leftarrow 0$;
   while *vectors* $\neq$ empty array **do**
   $dim \leftarrow dim + 1$;
   while $\exists k \in vectors \bullet \sigma[k] \cap \bigcup_{i \in vectors_{dim}} \sigma[i] = \varnothing$ **do**
   Remove k from *vectors*;
   Add k to $vectors_{dim}$;
   od
   od
   ⌞ $maxDimension \leftarrow dim$.

   ColourVectorNodes is a heuristics based greedy algorithm. We have found that it will yield concept hierarchies that are quite good, even though it sometimes takes a keyword that would logically fit better in a later dimension.

6. Fill out the concept hierarchies according to the subsumptions found in step 3 on the preceding page;

   proc *GenerateDimensions*() $\equiv$
   ⌜ **for** $i \leftarrow 1$ **to** *maxDimension* **do**
   $N_i \leftarrow vectors_i$;
   foreach $k_i \in vectors_i$ **do**
   $N_i \leftarrow N_i \cup \{\forall k_j \in \{keywords\} \bullet k_j <_k k_i\}$;
   od
   $E_i \leftarrow \{\forall k_i, k_j \in \{N_i, N_i\} \bullet k_i <_k k_j \mid (k_j, k_i)\}$;
   Perform topological sort on E_i;
   ⌞ **od.**

   GenerateDimensions generates the concept hierarchies for each dimension (see Figure 7.3 on page 70 for an example), i.e. the desired output for later data mining on the data.

7.4 Example of Execution

If we take the dataset in Figure 7.1 as input to our algorithm we will get the following results:

1. Reading inverted files yields (this is subset of the results, full results are given in Table 7.2 on page 76 in Appendix 7.A):

$$\sigma[make] = \{01\}$$
$$\sigma[text] = \{01, 02, 05, 07-13, 15\}$$
$$\sigma[ELF] = \{03, 14, 16\}$$
$$\sigma[MSB] = \{03, 14, 16\}$$
$$\sigma[executable] = \{03\}$$

2. The following non-trivial keyword families are found in the data:

family	**folded keywords**
make	commands
Bourne	shell, script
ELF	32-bit, MSB, SPARC, version, 1, not, stripped
executable	dynamically, linked
International	language
auxiliary	file
PostScript	conforming, at, level, 2.0
TEX	transcript
C	program
lex	description

3. Subsumptions found are:

$2e <_k document$	$English <_k text$	$document <_k text$
$2e <_k text$	$International <_k text$	$executable <_k ELF$
$2e <_k LaTeX$	$PostScript <_k document$	$lex <_k text$
$ASCII <_k text$	$PostScript <_k text$	$make <_k text$
$Bourne <_k text$	$TeX <_k text$	$relocatable <_k ELF$
$C <_k text$	$auxiliary <_k LaTeX$	

The results of steps 2 and 3 can be seen in Figure 7.2 on the facing page.

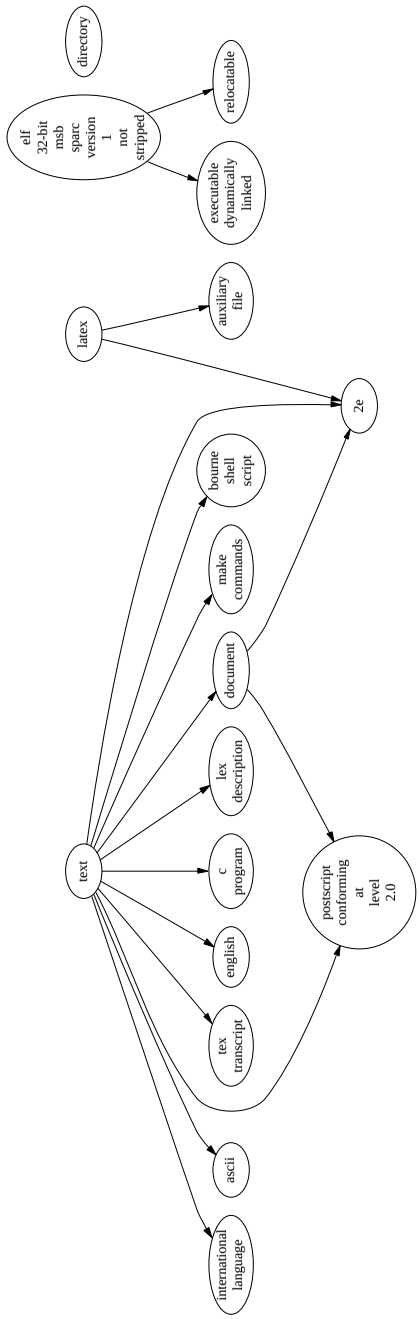


Figure 7.2: Subsumption and family graph of the example data in Figure 7.1 on page 63.

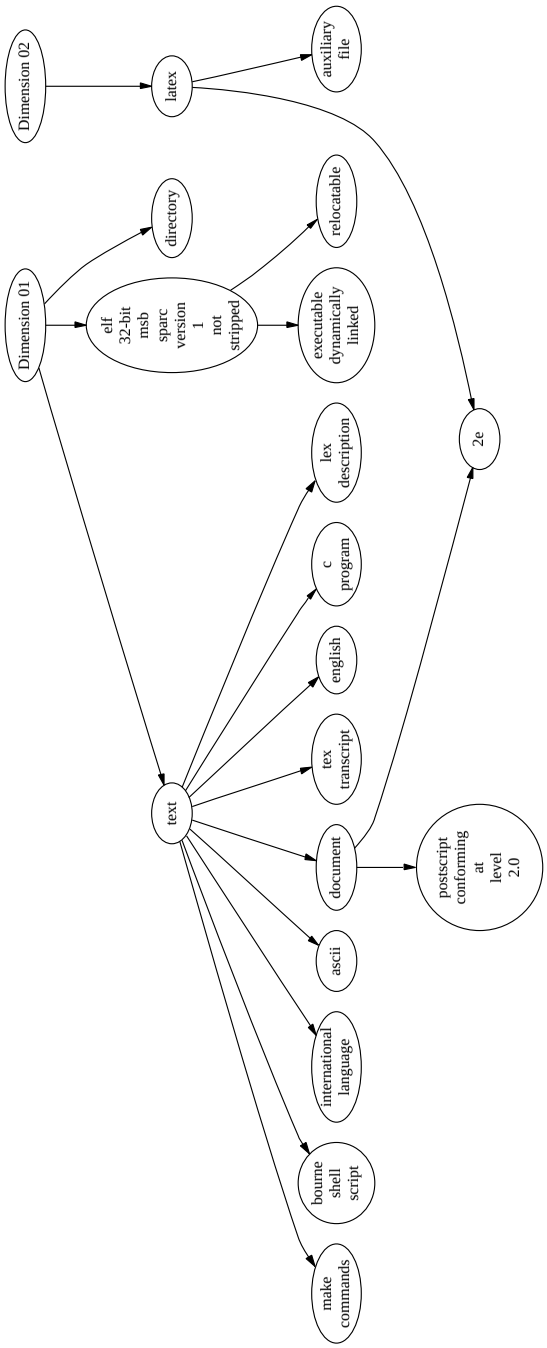


Figure 7.3: The concept hierarchy generated from the data in Figure 7.1 on page 63.

4. The vector families in the sample are *text*, *ELF*, *LaTeX* and *directory* with 11, 3, 2 and 1 instances respectively.

5. The first dimension consists of the vectors *text*, *ELF* and *directory*, while the second dimension has only one vector, *LaTeX*.

6. Only 2 edges are removed in the topological sorting of this small dataset, *(text,PostScript)* and *(text,2e)*. The end result of running the algorithm is seen in Figure 7.3 on the preceding page.

7.5 Algorithm Analysis

The algorithm given in Section 7.3 is a step by step description of what should be done in order to get a consistent concept hierarchy for later data mining. The steps are to be performed one after another and some of the steps can easily be broken down in parts to be executed in parallel (e.g. all dimensions can be handled simultaneously in Step 6 on page 67).

Two things have to be considered though, and that is the maximal cost of running the algorithm and the actual running times on real data. The complexity of the algorithm is calculated in Section 7.5.1 and the running time of the prototype is given in Section 7.5.2.

7.5.1 Cost Estimation

Given m (number of records) and n (number of keywords), we can give the following upper bounds of the algorithmic complexities:

proc *ReadInvertedFiles*() $\equiv O(mn)$.
proc *FindAndFoldFamilies*() $\equiv O(mn^2)$.
proc *FindSubsets*() $\equiv O(mn^2)$.
proc *FindVectorNodes*() $\equiv O(mn^2)$.
proc *ColourVectorNodes*() $\equiv O(mn^2)$.
proc *GenerateDimensions*() $\equiv O(mn+n^2)$.
total $\equiv O(mn^2)$.

This includes the cost of set operations that we estimate to be linear to the number of possible elements in a set, e.g. $O(m)$ or $O(n)$.

7.5.2 Actual Execution Times

The execution times of the prototype as given in Table 7.1 are on a single processor of a Sun Ultra Enterprise 450 server with 4 UltraSPARC-II processors, 4 MB level 2 cache and 4 GB of memory.

Table 7.1: Execution times for some sample datasets.

Dataset	# records	# keywords	User time (s)
1	12492	634	5.93
2	12492	780	10.36
3	256388	7182	16513.28

7.6 Related Work

Semantic and knowledge indexing has been a widely spread research area that includes such diverse topics as finding keywords for hierarchical summarization [11], knowledge acquisition tools [4] and automatic indexing of system commands based on their manual pages on a UNIX system [17].

Conceptual clustering systems [18] are also quite closely related to this work, but from the viewpoint of the data mining system rather than at the preprocessing stage.

Automatic generation of concept hierarchies [15] is related but works on a probabalistic, rather than a discrete, view of the data. It will therefore throw away some of the relevant subsumptions in the data and that is not an option for a data mining system.

The concept hierarchies generated by our algorithm have strong resemblances to the feature-oriented classification trees used by Salton [14], Prieto-Díaz [13] and Börstler [3]. We believe that this is not a coincident, and that such structures evolve naturally when working with closely related pieces of data like keywords or software assets.

The OPTICS system [1] finds critical points in a large database, points that can be used as basis for later clustering. These points should be viewed as “hot-spots” where interesting information resides (see 2*e* in Figure 7.3 on page 70 for such a “hot-spot” found by our algorithm).

7.7 Discussion

There exists one method of speeding up the *average* case while keeping the worst case the same. Sorting the keywords in decreasing cardinality order in Step 1 on page 65 will speed the algorithm up for the average case, since keyword equality implies the same cardinality and subset a lower cardinality. This implies that the number of keywords to be checked in Steps 2 on page 65 & 3 on page 66 can be decreased significantly (with factors of approximately $1/\log(n)$ and at least $1/2$, respectively).

The grouping of vectors into dimensions is currently sub-optimal, since it uses a heuristic algorithm rather than a best fit or even backtrack based graph colouring algorithm [12, 9]. We first believed such to be too expensive in computing time, but since all of our test datasets have had a relatively small number of vectors (144 keywords in the case of our largest dataset with 256388 records) it is feasible.

Some of the non-vector keywords are subsumed by more than one vector, thereby generating a lattice structure with rather complex properties. This is both a weakness and a strength; A weakness since it will generate the same set of vertices more than once and a strength since such keywords are excellent candidates for finding correlations and interesting data points in later Data Mining operations. It is trivial to exclude keywords after they have been used for the first time in a dimension if that would fit a certain problem.

Another somewhat harder problem with our implementation is that it will yield a conceptual hierarchy for each dimension even though the data would actually indicate that a lattice would be expected or more fitting (see Figure 7.4 on the following page for examples of both). The reason for this is the topological sorting done in the last step of GenerateDimensions (Step 6 on page 67). Removing the correct vertices from the set to generate a lattice is not that hard; Subsumption is a transitive function, e.g. if keyword *a* subsumes both *b* and *c* while *b* subsumes *c* then the vertex (a,c) can safely be removed from the resulting set. This technique would be very useful (especially if sorting in decreasing cardinality order as in the first paragraph of the discussion has already been done) even though the cost would rise from $O(mn+n^2)$ to $O(mn^2)$ for Step 6 on page 67.

The time required for generating the concept hierarchy for our biggest dataset seems rather high (just over 4.5 hours of computation, see Section 7.5.2), but since this is done once for each dataset before doing data mining we believe it to be satisfactory anyway. One way of speeding this up is to create an extra dataset that contains no duplicate records and use this dataset when creating the concept hierarchy. The time for concept hierarchy generation on our large dataset (with 38187 unique records) falls to around 40 minutes execution time. Data mining are often done on static databases, e.g. data warehouses, so the cost of concept

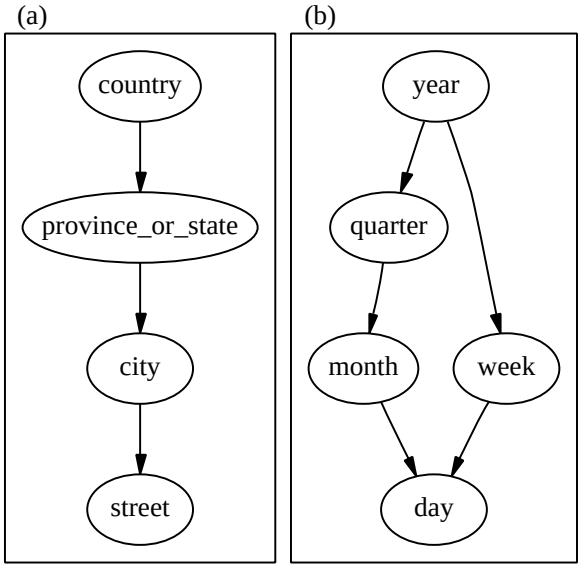


Figure 7.4: Hierarchical (a) and lattice (b) structures of attributes.

hierarchy generation should generally be amortized over the number of times that the resulting data is later used. Our approach should probably not be used in a constantly evolving database with major upgrades going on simultaneously.

Concluding Remarks and Future Work

All work so far on the prototype have been fruitful in the form of data relations both directly from concept hierarchy generation and also from later data mining. The concept hierarchies generated have generally been of good quality (with some minor glitches) and we hope to see future use of our algorithm in the works of others.

One thing that has to be further studied is the impact of the spanning in each dimension, e.g. it might not always be true that it is better to have one completely and one 50% spanned dimension rather than two dimensions with 75% spanning.

We are currently considering an upgraded version of the prototype that includes another algorithm for dimension generation and modifications in order to find lattice structures in the generated dimensions.

7.8 References

[1] ANKERST, M., BREUNIG, M. M., KRIEGEL, H.-P., AND SANDER, J. OPTICS: ordering points to identify the clustering structure. *ACM SIGMOD Record 28*, 2 (June 1999), 49–60.

[2] BORGIDA, A. Description Logics in Data Management. *IEEE Trans. Knowledge and Data Engineering 7*, 5 (Oct. 1995), 671–682.

[3] BÖRSTLER, J. FOCS: A Classification System for Software Reuse. In *Proceedings of the 11th Pacific Northwest Software Quality Conference* (PNSQC, Beaverton, OR, June 22-24, 1993), pp. 201–211.

[4] FUJIHARA, H., SIMMONS, D. B., ELLIS, N. C., AND SHANNON, R. E. Knowledge Conceptualization Tool. *IEEE Trans. Knowledge and Data Engineering 9*, 2 (Mar.–Apr. 1997), 209–220.

[5] FULLÉR, R. Neural Fuzzy Systems. Tech. Rep. A:443, Institute for Advanced Management Systems Research, Åbo, Finland, 1995.

[6] HAN, J., AND KAMBER, M. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, Inc., San Francisco, California, 2001.

[7] JAIN, A. K., MURTY, M. N., AND FLYNN, P. J. Data Clustering: A Review. *ACM Computing Surveys (CSUR) 31*, 3 (Sept. 1999), 264–323.

[8] KALLIN, L. *Applied Neural Logic*. Licentiate thesis, Umeå University, Umeå, Sweden, 1998.

[9] KNUTH, D. E. Estimating the efficiency of backtrack programs. *Mathematics of Computation 29* (1975), 121–136.

[10] KNUTH, D. E. *The Art of Computer Programming: Vol 3, Sorting and Searching*, second ed. Addison-Wesley, Reading, Massachusetts, 1998.

[11] LAWRIE, D., CROFT, W. B., AND ROSENBERG, A. Finding topic words for hierarchical summarization. In *Proceedings of the 24th ACM/SIGIR International Conference on Research and Development in Information Retrieval* (Sept. 9-12, 2001), pp. 349–357.

[12] MANBER, U. *Introduction to algorithms*. Addison-Wesley, Reading, Massachusetts, 1989.

[13] PRIETO-DÍAZ, R. Implementing Faceted Classification for Software Reuse. *Communications of the ACM 34*, 5 (May 1991), 88–97.

[14] SALTON, G. Manipulation of trees in information retrieval. *Communications of the ACM 5*, 2 (Feb. 1962), 103–114.

[15] SANDERSON, M., AND CROFT, B. Deriving concept hierarchies from text. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval* (Berkeley, California, United States, 1999), ACM Press, pp. 206–213.

[16] WITTEN, I. H., AND FRANK, E. *Data Mining*. Morgan Kaufmann Publishers, Inc., San Francisco, California, 2000.

[17] YE, H., AND LO, B. W. N. Towards a self-structuring software library. *IEE Proc. Soft. 148*, 2 (Apr. 2001), 45–55.

[18] YOO, J. P., PETTEY, C. C., AND YOO, S. A hybrid conceptual clustering system. In *Proceedings of the 1996 ACM 24th annual conference on Computer science* (Philadelphia, Pennsylvania, United States, 1996), ACM Press, pp. 105–114.

7.A All Results from Step One in Section 7.4

Table 7.2 contains all results given by step one of the algorithm when running it on the sample data in Figure 7.1 on page 63. The record numbers in the table are those used in Figure 7.1 on page 63.

Table 7.2: All keywords found and the corresponding record numbers.

k_i	$\sigma[k_i]$	k_i	$\sigma[k_i]$
make	01	LaTeX	06, 10
commands	01	auxiliary	06
text	01, 02, 05, 07 – 13, 15	file	06
Bourne	02	ASCII	07
shell	02	PostScript	08
script	02	document	08, 10
ELF	03, 14, 16	conforming	08
32-bit	03, 14, 16	at	08
MSB	03, 14, 16	level	08
executable	03	2.0	08
SPARC	03, 14, 16	TeX	09
version	03, 14, 16	transcript	09
1	03, 14, 16	2e	10
dynamically	03	English	11
linked	03	C	12, 13
not	03, 14, 16	program	12, 13
stripped	03, 14, 16	relocatable	14, 16
directory	04	lex	15
International	05	description	15
language	05		

III

Chapter 8

CHIC: A Fast Concept HIerarchy Constructor for Discrete or Mixed Mode Databases

Paper appears with kind permission from the Knowledge Systems Institute.

Abstract

In this paper we propose an algorithm that automatically creates concept hierarchies or lattices for discrete databases and datasets. The reason for doing this is to accommodate later data mining operations on the same sets of data without having an expert create these hierarchies by hand.

Each step of the algorithm will be examined; We will show inputs and output for each step using a small example. The theoretical upper bound of the complexity for each part of the algorithm will be presented, as well as real time measurements for a number of databases. We will finally present a time model of the algorithm in terms of a number of attributes of the databases.

Keywords: Data Mining, Data Preprocessing, Hierarchy Generation, Lattice Generation

8.1 Introduction

Data mining in large sets of data is not a trivial occupation. Choosing one type of algorithm over another for a certain problem can mean the difference between getting good (for some definition of good) or no, or even inconclusive, results. Almost all of the methods used today in data mining require either structured data (so that clustering can easily be performed) or a concept hierarchy that envisions the inner structure of the data [6, 4, 16, 5].

Concept hierarchies are descriptors over data sets, in such a way that all records in the corresponding data sets can be described by the hierarchies. Concept hierarchies are typically used in retrieval systems [12, 11, 2] and for data mining [5]. Each facet or aspect of the records has a corresponding hierarchy, often in the form of a tree. An example of such a hierarchy that describes a C source code could be $Imperative \rightarrow C \rightarrow ANSI$ for the language facet and $Ordered \rightarrow Tree \rightarrow Binary$ for the data type facet. Constructing a concept hierarchy manually is very time consuming and relies on a thorough understanding of the actual data at hand. Construction of concept hierarchies can be automated (as in this work). However, there are still some doubts about the soundness of automatically generated facets/dimensions, especially since no exact rules for what is a "good" concept hierarchy exist today.

The remainder of the paper is organized as follows; The second section is the background to the work. The third section gives the definitions necessary for understanding the later parts of this paper. The fourth section is the algorithm, while the fifth contains an example of running the algorithm. The sixth section contains an analysis of the algorithm and the seventh shows relationships with previous work. The last two sections contains the discussion and the experiences that we have had with CHIC, respectively.

8.2 Background

A few years ago we were faced with a large data set consisting of two parts; One relational and one consisting of a set of keywords. Feeding only the relational part of the data set into data mining systems revealed no new information about it. We realized that the only way of getting more information from the data set was to use the given set of keywords as means of additional clustering. We were unable to find any already published algorithm for finding structures in discrete data, so we had to create one ourselves. The result of our work was a concept hierarchy constructor, called CHiC.

CHiC will automatically find concept hierarchies (or lattices, as the case might be) in sets of discrete data. Our definition of discrete data is either keyword based or otherwise enumerated data, e.g. the sample data in Figure 8.1 on the following page. Our algorithm knows nothing about differences between numbers, whether integer or floating point, so it cannot be used to cluster numeric or continuous data.

CHiC has been used by major divisions of at least two multinational companies to generate concept hierarchies from their mixed mode databases.

```
01 Makefile:   make commands text
02 clean:      Bourne shell script text
03 combinator: ELF 32-bit MSB executable, SPARC, version 1, dynamically linked, not stripped
04 db:         directory
05 fest.txt:   International language text
06 fil.aux:    LaTeX auxiliary file
07 fil.dot:    ASCII text
08 fil.eps:    PostScript document text conforming at level 2.0
09 fil.log:    TeX transcript text
10 fil.tex:    LaTeX 2e document text
11 input:      English text
12 lex.yy.c:   C program text
13 main.c:     C program text
14 main.o:     ELF 32-bit MSB relocatable, SPARC, version 1, not stripped
15 words.l:    lex description text
16 words.o:    ELF 32-bit MSB relocatable, SPARC, version 1, not stripped
```

Figure 8.1: A short example of discrete meta-data, in this case output of the “file” command.

8.3 Definitions

Given $k_i \in$ *keywords* (the keywords in the system) and $\sigma_{k_i} \in \mathbb{P}$*records* (the set of records in the database that contain keyword k_i), we can define the concepts used in this work.

A **bucket** is a placeholder for a set of keywords.

Two or more keywords that always appear together are **keyword equivalent**, i.e.

$$k_i =_k k_j \iff \sigma_{k_i} = \sigma_{k_j}, \quad \text{where } k_i \neq k_j. \tag{8.1}$$

This implies that they may be **folded** into one keyword, i.e. the name of the latter is added as a synonym of the first and all references to the latter are removed or ignored. Such removed keywords are said to belong to k_is **family**, i.e.

$$k_i =_k k_j \Longrightarrow k_i, k_j \in fam_i, fam_i \in \mathbb{P}keywords. \tag{8.2}$$

Formally, **subsumption** is defined as an implicit subset/superset relationship between the interpretations of the two concepts [1]. This means that if a keyword k_i never appears in a record without k_j appearing, but not vice versa, then k_i is **subsumed** by k_j.

$$k_i <_k k_j \iff \sigma_{k_i} \subset \sigma_{k_j} \tag{8.3}$$

A keyword k_i that is not subsumed by another keyword is said to be a **vector**. Vectors are used as the bases of the concept hierarchies.

8.4 The Algorithm

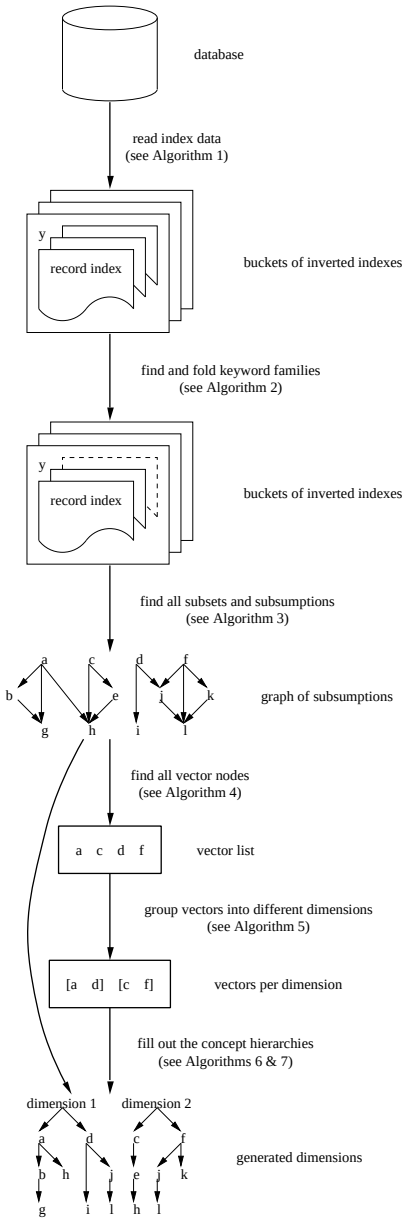


Figure 8.2: Each step of the algorithm and all intermediate results.

The algorithm (as outlined in Figure 8.2 on the preceding page) given here is straight forward, although some of the implementation details can be anything but trivial to implement. The only requirements are that the keywords must have a consistent enumeration order (as given in Equation 8.4) and that the database has already been created. The different parts of the algorithm are otherwise independent of different implementations of sets, etc.

$$\forall x, y \in \mathit{keywords} \bullet x \neq y \longrightarrow \mathit{ord}_x \neq \mathit{ord}_y \tag{8.4}$$

Further information about each part of the algorithm are given in Section 8.4.7 (correctness of the algorithms) and Section 8.6 (cost estimations based on code complexity).

8.4.1 Read Index Data

All keywords should be given an enumeration order and the inverted file (see [8]) corresponding to each keyword is read from disk. The following must hold after the step in the algorithm has been executed:

$$\forall x \in \mathit{keywords} \bullet \sigma_x = \text{inverted DB file}_x \tag{8.5}$$

$$\forall x \in \mathit{keywords} \bullet |\sigma_x| = y \longrightarrow x \in \mathit{bucket}_y \tag{8.6}$$

ALGORITHM 8.1 (READINVERTEDFILES) *This step of the algorithm reads the index/inverted data files from the disc, or must generate them (at some run time cost) if they do not exist. In this algorithm description (and in the cost estimation in Section 8.6.1) we expect them to be on disc before execution.*

proc *ReadInvertedFiles()* $\equiv$
⌜ Empty all buckets*;*
foreach $x \in \mathit{keywords}$ **do**
$\sigma_x \leftarrow \mathit{inverted\ DB\ File}_x$*;*
$y \leftarrow |\sigma_x|$*;*
$\mathit{bucket}_y \leftarrow \mathit{bucket}_y \cup \{x\}$*;*
⌟ **done**

8.4.2 Find and Fold Keyword Families

The second step of the algorithm finds keywords that are keyword equivalent, adds the latter to the formers family and removes the latter (according to the keyword enumeration order) from further computations.

The following must always hold true for the algorithm used:

$$\forall x,y \in keywords \bullet ord_x < ord_y \wedge \sigma_x = \sigma_y \longrightarrow x =_k y \tag{8.7}$$

where y should be excluded from further computations.

ALGORITHM 8.2 (FINDANDFOLDFAMILIES) *This step of the algorithm compares each keyword in a bucket with all other keywords with a later enumeration order in the same bucket, removing the latter if the two are keyword equivalent.*

proc *FindAndFoldFamilies()* $\equiv$
foreach *bucket* $\in \{buckets\}$ **do**
  while $\exists x,y \in bucket \bullet ord_x < ord_y \wedge \sigma_x = \sigma_y$ **do**
    $fam_x \leftarrow fam_x \cup fam_y$;
    $bucket \leftarrow bucket \setminus fam_y$;
    $keywords \leftarrow keywords \setminus fam_y$;
  done
done

8.4.3 Find All Subsets and Subsumptions

This step of the algorithm finds where one keyword is subsumed by another, while ignoring folded keywords altogether. The effect of this step can be summarized as:

$$\forall x,y \in \mathit{keywords} \bullet x \neq y \wedge \sigma_x \subset \sigma_y \longrightarrow x <_k y \tag{8.8}$$

ALGORITHM 8.3 (FINDSUBSETS) *This step of the algorithm will find all subsumptions in the database.* CHIC *works in decreasing bucket size order to further optimize the algorithm.*

proc *FindSubsets()* $\equiv$
foreach $x \in \mathit{keywords}$ **do**
  foreach $y \in \mathit{keywords} \bullet |\sigma_y| < |\sigma_x|\}$ **do**
    if $\sigma_x \subset \sigma_y$ **then** $x <_k y$;
  done
done

The reason for performing family folding before finding subsets is to prune the solution space as quickly as possible, since all keywords in a family will subsume/be subsumed by exactly the same keywords.

8.4.4 Find All Vector Nodes

The goal of the next step is to find all vectors (keywords that are not subsumed by any other keyword) in the given dataset, i.e.:

$$\mathit{vectors} = \{x \in \mathit{keywords} \mid \nexists y \in \mathit{keywords} \bullet x <_k y\} \tag{8.9}$$

ALGORITHM 8.4 (FINDVECTORNODES) *This step of the algorithm will add all found vectors in an array, sorted in decreasing cardinality order. The order is important for the heuristics of the next step.*

proc *FindVectorNodes()* $\equiv$
vectors $\leftarrow$ *empty array;*
foreach $x \in \mathit{keywords}$ **do**
  if $\nexists y \bullet y \in \mathit{keywords} \wedge x <_k y$
    then *insert k in vectors;*
done

8.4.5 Group Vectors into Different Dimensions/Facets

Partition the set of vectors into as few partitions ($vectors_i$) as possible, while maintaining:

$$\forall x,y \in vectors_i \bullet x \neq y \wedge \sigma_x \cap \sigma_y = \varnothing \tag{8.10}$$

ALGORITHM 8.5 (COLOURVECTORNODES) *The algorithm is a greedy implementation based on heuristics that maintains Equation 8.10.* CHIC *contains two different versions of keyword selection; The first version chooses the keyword with the highest cardinality that fits and the second selects the keyword that has the highest number of conflicts with other keywords first. The latter version is much more costly in terms of CPU time as can be seen in Section 8.6.1.*

proc *ColourVectorNodes()* $\equiv$
```
    dim ← 0;
    while vectors ≠ empty array do
        dim ← dim + 1;
        while ∃k ∈ vectors • σ_k ∩ ⋃_{y∈vectors_dim} σ_y = ∅ do
            Remove k from vectors;
            Add k to vectors_dim;
        done
    done
    maxDimension ← dim;
```

There exists yet another option to CHIC that changes the behavior of this step slightly. It is possible to ask the program to fill the dimensions as much as possible by reusing vectors from one dimension in later dimensions if there is no conflict with vectors already added in that dimension. The easiest way to do this is to mark vectors with no conflict with any remaining vector in *vectors* to be used in current dimension and all that follows it. It is also very easy to go back and check already used vectors for conflicts when setting up latter dimensions.

The reason for using a heuristics based approach rather than trying all possible combinations is the extreme processing cost that it would incur. Finding the optimal colouring for even a small data base of, e.g., 36 vectors and 6 dimensions would yield 6^{35} combinations[1]. Given a computer system that would test 50,000 combinations per second that would take in the order of $3.44 * 10^{22}$ seconds, or roughly 10^{15} years. Our system yields an answer in a fraction of a second for the same data base, but we can not conclusively say that it is an optimal solution (neither in the number of dimensions, nor in the spanning of each dimension).

[1] One vector locked in the first dimension and with many permutations of combinations occurring more than once but in different dimensions.

8.4.6 Fill Out the Concept Hierarchies/Lattices According to the Subsumptions Found in Step 8.4.3 on page 87

This step generates a number of graphs $G_i = (N_i, E_i)$ where the nodes and the edges are given by equations 8.11 and 8.12.

$$N_i = \{x \in \mathit{keywords} \mid x \in \mathit{vectors}_i \vee \exists y \in \mathit{vectors}_i \bullet x <_k y\} \tag{8.11}$$

$$E_i = \{(y,x) \mid x,y \in \mathit{keywords} \wedge x <_k y \wedge \nexists z \bullet (x <_k z \wedge z <_k y)\} \tag{8.12}$$

ALGORITHM 8.6 (GENERATEDIMENSIONS, HIERARCHY VERSION) *This algorithm generates concept hierarchies for each dimension (see Figure 8.4 on page 95 for an example), i.e. the desired output for later data mining on the data.*

proc *GenerateDimensions()* $\equiv$
for $i \leftarrow 1$ **to** *maxDimension* **do**
  $N_i \leftarrow \mathit{vectors}_i$;
  foreach $x \in \mathit{vectors}_i$ **do**
    $N_i \leftarrow N_i \cup \{y \in \mathit{keywords} \mid y <_k x\}$;
  done
  $E_i \leftarrow \{(y,x) \mid \forall x,y \in N_i \bullet x <_k y\}$;
  Perform topological sort on E_i;
done

ALGORITHM 8.7 (GENERATEDIMENSIONS, LATTICE VERSION) *This step of the algorithm generates concept lattices for each dimension. This means that there might be more than one path between a vector and a given node further down in the graph.*

proc *GenerateDimensions()* $\equiv$
for $i \leftarrow 1$ **to** *maxDimension* **do**
$N_i \leftarrow vectors_i$;
foreach $x \in vectors_i$ **do**
$N_i \leftarrow N_i \cup \{y \in keywords \mid y <_k x\}$;
done
done
foreach $x \in keywords$ **do**
foreach $y \in keywords \bullet y <_k x$ **do**
foreach $z \in keywords \bullet z <_k y$ **do**
Remove $z <_k x$
done
done
done
for $i \leftarrow 1$ **to** *maxDimension* **do**
$E_i \leftarrow \{(y,x) \mid \forall x,y \in N_i \bullet x <_k y\}$;
done

8.4.7 Correctness of the Algorithm

Equation 8.4 on page 85 implies that each keyword should have an unique order number. This number is used when selecting which keyword to fold and which to keep in Algorithm 8.2 on page 86, and it is also used by CHIC to prune the solution space in Algorithm 8.3 on page 87. Our solution to the ordering number is to give each keyword a number from 1024 and up in the order that they were seen when generating the database, which implies that a keyword cannot be subsumed by a keyword with a higher number unless they appeared for the first time in the same record.

Equation 8.5 on page 85 is trivial. The implication of this equation is that the values have to be available at later stages, but how these are made available is not important for the algorithm.

Equation 8.6 on page 85 shows the bucket sorting stage of the algorithm. The algorithm will work even if the sorting stage is removed, but at a much higher computational cost (see [17]).

The slightly rewritten form of Algorithm 8.1 on page 85 seen below shows that it does indeed fulfill Equations 8.5 and 8.6 on page 85.

proc ReadInvertedFiles() $\equiv$
$\ulcorner$ $\forall x \in keywords \bullet \sigma_x \leftarrow$ inverted DB File$_x$;
$\llcorner$ $\forall x \in keywords \bullet bucket_{|\sigma_x|} \leftarrow bucket_{|\sigma_x|} \cup \{x\}$;

Equation 8.7 on page 86 is a combination of Definitions 8.2 and 8.3 on page 83. The implication of the equation is that only one of each pair of keyword equivalent keywords should be used in later computation; The latter keyword is used as yet another name on the syntactic rather than semantic level of the keyword still remaining. Proving that Algorithm 8.2 on page 86 fulfills Equation 8.7 on page 86 is trivial, since it follows directly from Definitions 8.2 and 8.3 on page 83 together with Equation 8.4 on page 85.

Equation 8.8 on page 87 follows directly from Definition 8.3 on page 83. Algorithm 8.3 on page 87 is a rewriting of the equation to ignore some of the impossible solutions (i.e., checking if the keyword x subsumes the keyword y **iff** $|\sigma_x| > |\sigma_y|$), thereby making it more efficient.

Equation 8.9 on page 87 is the definition of **vector** and Algorithm 8.4 on page 87 is just an extension of that equation.

Equation 8.10 on page 88 is the minimal requirements for step 8.5 on page 88, but it does not specify how it it achieved. Algorithm 8.5 on page 88 is written is such a way that it complies with Equation 8.10 on page 88, but the critical part in this step is the selection algorithm in the inner while loop. CHiC contains two different selection algorithms, as mentioned in Section 8.4.5.

Equation 8.11 on page 89 makes sure that only those keywords that can be reached from the vectors in a given dimension is added to that dimension.

Equation 8.12 on page 89 yields the edges (subsumptions) that are between keywords belonging to that dimension, as per Equation 8.11 on page 89.

Algorithm 8.7 on the facing page follows Equations 8.11 and 8.12 on page 89, while Algorithm 8.6 on page 89 has the added rule that there must exist at most one path between a vector and any given keyword of that dimension. This is the difference between generating a hierarchy and a lattice.

8.5 Example of Execution

If we take the dataset in Figure 8.1 on page 82 as input to our algorithm we will get the following results:

1. Reading inverted files yields:

k_i	$\sigma[k_i]$
make	01
commands	01
text	01, 02, 05, 07 – 13, 15
Bourne	02
shell	02
script	02
ELF	03, 14, 16
32-bit	03, 14, 16
MSB	03, 14, 16
executable	03
SPARC	03, 14, 16
version	03, 14, 16
1	03, 14, 16
dynamically	03
linked	03
not	03, 14, 16
stripped	03, 14, 16
directory	04
International	05
language	05

k_i	$\sigma[k_i]$
LaTeX	06, 10
auxiliary	06
file	06
ASCII	07
PostScript	08
document	08, 10
conforming	08
at	08
level	08
2.0	08
TeX	09
transcript	09
2e	10
English	11
C	12, 13
program	12, 13
relocatable	14, 16
lex	15
description	15

The buckets are also filled in with the following contents:

bucket #	**Content**
1	make, commands, Bourne, shell, script, executable, dynamically, linked, directory, International, language, auxiliary, file, ASCII, PostScript, conforming, at, level, 2.0, TeX, transcript, 2e, English, lex, description
2	LaTeX, document, C, program, relocatable
3	ELF, 32-bit, MSB, SPARC, version, 1, not, stripped
11	text

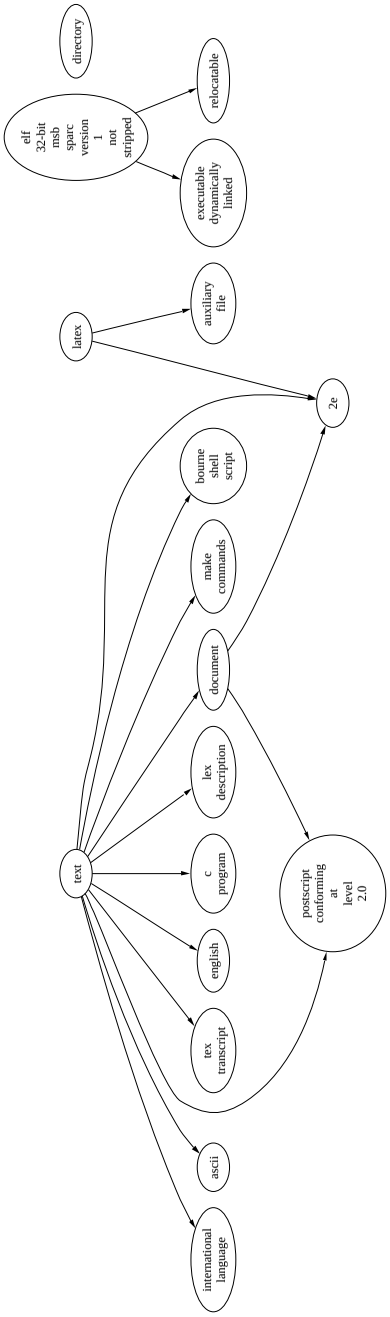


Figure 8.3: Subsumption and family graph of the example data in Figure 8.1.

2. The following non-trivial keyword families are found in the data:

family	**folded keywords**
make	commands
Bourne	shell, script
ELF	32-bit, MSB, SPARC, version, 1, not, stripped
executable	dynamically, linked
International	language
auxiliary	file
PostScript	conforming, at, level, 2.0
TEX	transcript
C	program
lex	description

3. Subsumptions found are:

$2e <_k document$	$English <_k text$	$document <_k text$
$2e <_k text$	$International <_k text$	$executable <_k ELF$
$2e <_k LaTeX$	$PostScript <_k document$	$lex <_k text$
$ASCII <_k text$	$PostScript <_k text$	$make <_k text$
$Bourne <_k text$	$TeX <_k text$	$relocatable <_k ELF$
$C <_k text$	$auxiliary <_k LaTeX$	

   The results of steps 2 and 3 can be seen in Figure 8.3 on the preceding page.

4. The vector families in the sample are *text*, *ELF*, *LaTeX* and *directory* with 11, 3, 2 and 1 instances respectively.

5. The first dimension consists of the vectors *text*, *ELF* and *directory*, while the second dimension has only one vector, *LaTeX*.

6. Only 2 edges are removed in the topological sorting of this small dataset, *(text,PostScript)* and *(text,2e)*. The end result of running the algorithm is seen in Figure 8.4 on the next page.

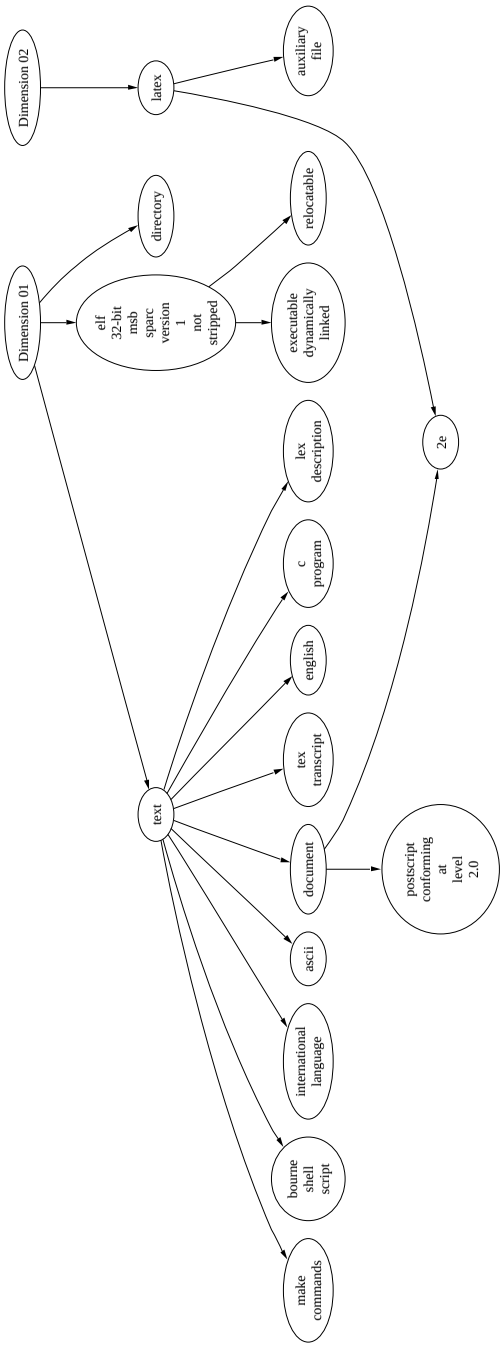


Figure 8.4: The concept hierarchy generated from the data in Figure 8.1.

8.6 Algorithm Analysis

The algorithm given in Section 8.4 is a step by step description of what should be done in order to get a consistent concept hierarchy for later data mining. The steps are to be performed one after another and some of the steps can easily be broken down in parts to be executed in parallel (e.g. all dimensions can be handled simultaneously in steps 8.6 on page 89 and 8.7 on page 90).

Two things have to be considered though, and that is the maximal cost of running the algorithm and the actual running times on real data. The complexity of the algorithm is calculated in Section 8.6.1 and the running time of the prototype is given in Section 8.6.2.

8.6.1 Cost Estimation

Given m (number of records) and n (number of keywords), we can give the following upper bounds of the algorithmic complexities:

$$\textbf{proc}\ \text{ReadInvertedFiles}() \equiv O(mn)$$
$$\textbf{proc}\ \text{FindAndFoldFamilies}() \equiv O(mn^2)$$
$$\textbf{proc}\ \text{FindSubsets}() \equiv O(mn^2)$$
$$\textbf{proc}\ \text{FindVectorNodes}() \equiv O(mn^2)$$
$$\textbf{proc}\ \text{ColourVectorNodes}() \equiv O(mn^2) \text{ or } O(mn^{2.5})$$
$$\textbf{proc}\ \text{GenerateDimensions}() \equiv O(mn+n^2) \text{ or } O(mn^2)$$
$$\text{total} \equiv O(mn^2) \text{ or } O(mn^{2.5}).$$

This includes the cost of set operations that we estimate to be linear to the number of possible elements in a set, e.g. $O(m)$ or $O(n)$.

Table 8.1: Descriptions of the test data bases.

Set	**records** (m)	**keywords** (n)	**vectors** (n')	**subsets** (m')
1	12492	634	36	2409
2	12492	780	63	3850
3	256388	7182	114	49631
4	9480	5042	2458	6781
5	7496	3204	1400	4357

8.6.2 Actual Execution Times

Some of the data bases used for testing the algorithm are summarized in Table 8.1. The first three are prototypical for the discrete databases that we have encountered elsewhere. The last two are more academic in their nature, since they have unusually high number of keywords and vectors for their size.

The execution times of the prototype as given in Table 8.2 on the next page are from a Sun Blade-1000 workstation with a 750 MHz UltraSPARC-III processor, 8 MB level 2 cache and 512 MB of memory. CHIC was compiled using gcc version 2.95.2.

Modeling the execution time in terms of the values given in Table 8.1 and the time values in Table 8.2 on the next page (together with multiple other data sets) yielded a simple model for each step. We have not modeled the extra time added in step 5a/5b if reusing of vectors in later dimensions is used (as per Section 8.4.5), but since it is in the order of 0.2% of the execution time (for all but the smallest of our data bases, where it increases the time 0.7%) it is almost negligible.

Table 8.2: Average execution times per step in seconds (rounded to three decimals).

Set	**1**	**2**	**3 and 4**	**5a**	**5b**	**6a**	**6b**	**fill+6a**
1	0.287	0.363	0.869	0.012	0.081	0.329	0.116	0.437
2	0.442	0.382	1.574	0.041	0.397	0.516	0.230	0.654
3	34.931	1732.080	3047.250	5.060	6.678	64.088	23.861	67.413
4	2.274	8.436	42.454	36.407	1012.340	5.241	5.952	48.582
5	1.206	2.410	12.409	9.662	266.126	2.002	2.375	14.383

$$t_1 \approx 1.77 * 10^{-8} mn + 3.19 * 10^{-4} n \tag{8.13}$$

$$t_2 \approx 1.31 * 10^{-10} mn^2 \tag{8.14}$$

$$t_{3\&4} \approx 2.30 * 10^{-10} mn^2 \tag{8.15}$$

$$t_{5_a} \approx 6.40 * 10^{-10} mn'^2 \tag{8.16}$$

$$t_{5_b} \approx 1.36 * 10^{-6} \ln(m) n'^{2.333} \tag{8.17}$$

$$t_6 \approx 1.77 * 10^{-7} m'n \tag{8.18}$$

$$t_{fill+6} \approx 2.44 * 10^{-11} m'n^2 + 3.51 * 10^{-6} nn' \tag{8.19}$$

$$t_7 \approx 4.04 * 10^{-8} m'n + 1.79 * 10^{-7} n^2 \tag{8.20}$$

The number of vectors and subsets can often be hard to guess before calculations have been done, but we have found that $n' \approx n^2/m$ and $m' \approx \sqrt{mn}$ are useful approximations. They will usually yield results within 25% from the correct value for almost all data bases[2].

[2] Set three in Table 8.2 is one such exception, since it overestimates n' by 76%.

8.7 Related Work

The work reported in this article is closely related to semantic and knowledge indexing. It is a widely spread research area that includes such diverse topics as finding keywords for hierarchical summarization [9], knowledge acquisition tools [3] and automatic indexing of system commands based on their manual pages on a UNIX system [14].

Conceptual clustering systems [15] are also quite closely related to this work, but from the viewpoint of the data mining system rather than at the preprocessing stage.

Automatic generation of concept hierarchies as defined in [13] is related but works on a probabilistic, rather than a discrete, view of the data. The rule used to find subsumptions in that work is that x would subsume y if $P(x|y) \geq 0.8, P(y|x) < 1$. This means that transitivity will not work for subsumptions; Given three keywords (x, y and z) such that $P(x|y) \geq 0.8, P(y|x) < 1$, $P(y|z) \geq 0.8, P(z|y) < 1$ and $P(x|z) \ngeq 0.8, P(z|x) < 1$ would yield a very problematic state since z is subsumed by y which in turn is subsumed by x but z is not subsumed by x. Whether this is acceptable or not is up to the user of the system.

The concept hierarchies generated by our algorithm have strong resemblances to the feature-oriented classification trees used by Salton [12], Prieto-Díaz [11] and Börstler [2]. We believe that this is not a coincident, and that such structures evolve naturally when working with closely related pieces of data like software assets.

8.8 Discussion

One method of speeding up the *average* case while keeping the worst case the same has been used in a newer version of CHIC. The keywords are sorted (using bucket sort) in decreasing cardinality order in step 8.1. This speeds up the algorithm considerably for the average case, since keyword equality implies the same cardinality and subset a lower cardinality. This implies that the number of keywords to be checked in steps 8.2 & 8.3 on page 87 was decreased significantly (with factors of approximately $1/\log(n)$ and at least $1/2$, respectively).

The grouping of vectors into dimensions uses a heuristic algorithm rather than a best fit or even backtrack based graph colouring algorithm [10, 7]. We have experimented with multiple versions of backtracking algorithms, but were not able to find any that were both fast enough and gave sufficiently better results than our heuristic algorithm.

Some of the non-vector keywords are subsumed by more than one vector, thereby generating a lattice structure with rather complex properties. This is both a weakness and a strength; A weakness since it will generate the same set of vertices more than once and a strength since such keywords are excellent candidates for finding correlations and interesting data points in later Data Mining operations. It would be rather simple to update CHIC so that it excludes all keywords after they have been used for the first time in a dimension if that would fit a certain problem.

Previous version of CHIC generated a conceptual hierarchy for each dimension even though the data indicated that a lattice would be more fitting (see Figure 8.5 on the facing page for examples of both). The reason for this was the topological sorting done in the last step of GenerateDimensions (Algorithm 8.6 on page 89). Removing the correct vertices from the set to generate a lattice was fortunately not that hard (see Algorithm 8.7 on page 90); Subsumption is a transitive function, e.g. if keyword *a* subsumes both *b* and *c* while *b* subsumes *c* then the vertex (a,c) can safely be removed from the resulting set. This technique proved to be very useful (especially when the keywords are already in decreasing cardinality order after Algorithm 8.1 on page 85 is done), but requires that the chosen Data Mining system can handle lattices rather than hierarchies.

The time required for generating the concept hierarchy for our biggest dataset seems rather high (just over 80 minutes of computation, see Section 8.6.2), but since this is done only once for each dataset before doing data mining we believe it to be satisfactory anyway. One way of speeding this up is to create an extra dataset that contains no duplicate records and use this dataset when constructing the concept hierarchy. The time for concept hierarchy generation on our large dataset (with 38187 unique records) falls to around 11 minutes execution time. Data mining is often done on static databases, e.g. data warehouses, so the cost

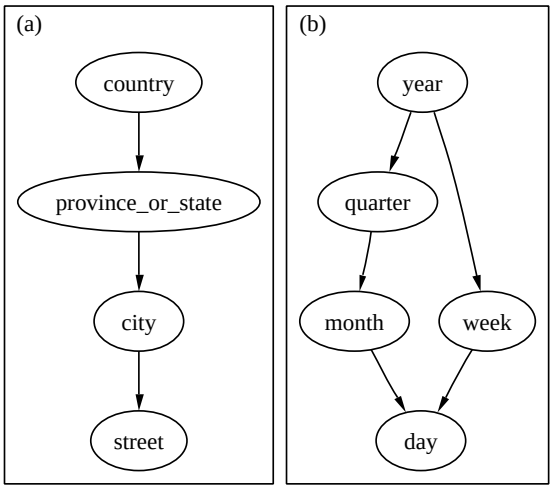


Figure 8.5: Hierarchical (a) and lattice (b) structures of attributes.

of concept hierarchy generation should generally be amortized over the number of times that the resulting data is later used. Our approach should probably not be used in a constantly evolving database with major upgrades going on simultaneously.

8.9 Experiences

All work so far on the prototype have been fruitful in the form of data relations both directly from concept hierarchy generation and also from later data mining. The concept hierarchies generated have generally been of good quality (i.e. logically connected keywords tend to be in the same dimension or even subsumed, etc.) with some minor glitches and we hope to see future use of our algorithm in the works of others.

The divisions of two multinational companies that have used CHIC have been very pleased with the results obtained from the program, and have found other, more novel, uses for it as well. One of the comments that we have received is that "*The only other option [available to us] would have been to hire a very expensive expert in the field, and she would probably have come up with something remarkably similar to what we get from* CHIC."

8.10 References

[1] BORGIDA, A. Description Logics in Data Management. *IEEE Trans. Knowledge and Data Engineering 7*, 5 (Oct. 1995), 671–682.

[2] BÖRSTLER, J. FOCS: A Classification System for Software Reuse. In *Proceedings of the 11th Pacific Northwest Software Quality Conference* (PNSQC, Beaverton, OR, June 22-24, 1993), pp. 201–211.

[3] FUJIHARA, H., SIMMONS, D. B., ELLIS, N. C., AND SHANNON, R. E. Knowledge Conceptualization Tool. *IEEE Trans. Knowledge and Data Engineering 9*, 2 (Mar.–Apr. 1997), 209–220.

[4] HAN, J., CAI, Y., AND CERCONE, N. Data-driven discovery of quantitative rules in relational databases. *IEEE Transactions on Knowledge and Data Engineering 5*, 1 (Feb. 1993), 29–40.

[5] HAN, J., AND KAMBER, M. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, Inc., San Francisco, California, 2001.

[6] JAIN, A. K., MURTY, M. N., AND FLYNN, P. J. Data Clustering: A Review. *ACM Computing Surveys (CSUR) 31*, 3 (Sept. 1999), 264–323.

[7] KNUTH, D. E. Estimating the efficiency of backtrack programs. *Mathematics of Computation 29* (1975), 121–136.

[8] KNUTH, D. E. *The Art of Computer Programming: Vol 3, Sorting and Searching*, second ed. Addison-Wesley, Reading, Massachusetts, 1998.

[9] LAWRIE, D., CROFT, W. B., AND ROSENBERG, A. Finding topic words for hierarchical summarization. In *Proceedings of the 24th ACM/SIGIR International Conference on Research and Development in Information Retrieval* (Sept. 9-12, 2001), pp. 349–357.

[10] MANBER, U. *Introduction to algorithms*. Addison-Wesley, Reading, Massachusetts, 1989.

[11] PRIETO-DÍAZ, R. Implementing Faceted Classification for Software Reuse. *Communications of the ACM 34*, 5 (May 1991), 88–97.

[12] SALTON, G. Manipulation of trees in information retrieval. *Communications of the ACM 5*, 2 (Feb. 1962), 103–114.

[13] SANDERSON, M., AND CROFT, B. Deriving concept hierarchies from text. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval* (Berkeley, California, United States, 1999), ACM Press, pp. 206–213.

[14] YE, H., AND LO, B. W. N. Towards a self-structuring software library. *IEE Proc. Soft. 148*, 2 (Apr. 2001), 45–55.

[15] YOO, J. P., PETTEY, C. C., AND YOO, S. A hybrid conceptual clustering system. In *Proceedings of the 1996 ACM 24th annual conference on Computer science* (Philadelphia, Pennsylvania, United States, 1996), ACM Press, pp. 105–114.

[16] YOON, S. C., SONG, I. Y., AND PARK, E. K. Intensional query processing using data mining approaches. In *Proceedings of the Sixth International Conference on Information and Knowledge Management* (Las Vegas, Nevada, United States, 1997), ACM Press, pp. 201–208.

[17] ÅGREN, O. Automatic Generation of Concept Hierarchies for a Discrete Data Mining System. In *Proceedings of the International Conference on Information and Knowledge Engineering (IKE '02)* (Las Vegas, Nevada, USA, June 24-27, 2002), pp. 287–293. Paper II on page 59.

IV

Chapter 9

Propagation of Meta Data over the World Wide Web

Paper appears with kind permission from CSREA Press.

Abstract

In this paper we propose a distribution and propagation algorithm for meta data. The main purpose of this is to tentatively allocate or derive meta data for nodes (in our case sites and/or web pages) for which no meta data exists.

We propose an algorithm that depends on 1) meta data given to a node, site and/or web page, 2) how pervasive we percieve this meta data, and 3) the trust that we give to this meta data. We will also show that PICS labels can be used to hold the meta data even for distant web pages and sites.

Keywords: meta data, automatic propagation, PICS, spatial linking

9.1 Introduction

Meta data embedded in web pages are one of the most reliable means of acquiring meta data for web pages. Web developers have however a tendency to omit meta data in their pages. Those that do add meta data tend to either put in very sparse meta data or everything that *might* occur further down in the web hierarchy that they control. In this work we propose a way to use the meta data already available in hierarchies of web pages to automatically propagate said information to other web pages.

Meta data that has been well thought out can be used to infer content domain (and in some cases sub-domain) for web pages. Some of the embedded meta data that we have found while looking at a large number of web pages can indeed be used as substitute for the content of the web page when doing searches.

We postulate that searching and browsing can be performed on the meta data level directly first. We have furthermore found that the connection between web pages and meta data can be viewed in the same way as PICS labels (i.e. meta data is either bound to a specific web page or to an entire (sub) hierarchy of the web).

9.1.1 PICS

W3C has devised a protocol called *Platform for Internet Content Selection* (PICS), with which organizations and companies can provide filters that allow only suitable pages to be presented. The PICS standard does not say how a rating shall be done, nor how it should be used or presented. That is completely up to the software providers and this has led to an abundance of different rating systems, none of which are compatible with the others. The good part with this is that a user of these systems can choose not only among suitable ratings but also which services to trust [9, 10, 4].

There are few providers of rating services that use PICS. All of them use it for client side blocking. That means that the browser first asks the service provider for its rating of a certain page, and if that rating is within predefined boundaries it starts to down-load the actual web page. Parents, teachers and/or employers can choose which rating service to trust and what levels of the ratings are suitable for down-loading. Providers of these types of services include NetNanny, Cybersitter, Cyber Patrol, and Surf Watch [13, 14, 17].

9.1.2 Web-Based Meta Data

PICS labels can be used to store any kind of textual meta-information about an Uniform Resource Identifier (URI). One way to use a PICS label bureau (an off-site provider of PICS labels) is as a meta data repository/warehouse. This data can then be searched much more effectively than searching through all the corresponding web pages [4, 19].

The PICS based labels are either for a specific web page (e.g. `http://www.com/-users.html`) or for a web path and everything beneath it (e.g. `http://www.com/`). If we have a label bureau set up and a request comes for information pertaining to a specific web page/address (e.g. `http://www.com/research/users.html`) the bureau must first try to locate information using the exact web address given, and if that fails try with each path that is a prefix of the web address (e.g. first for the `http://www.com/research/` path and then `http://www.com/`) until the required data is found [9] (see Figure 9.1).

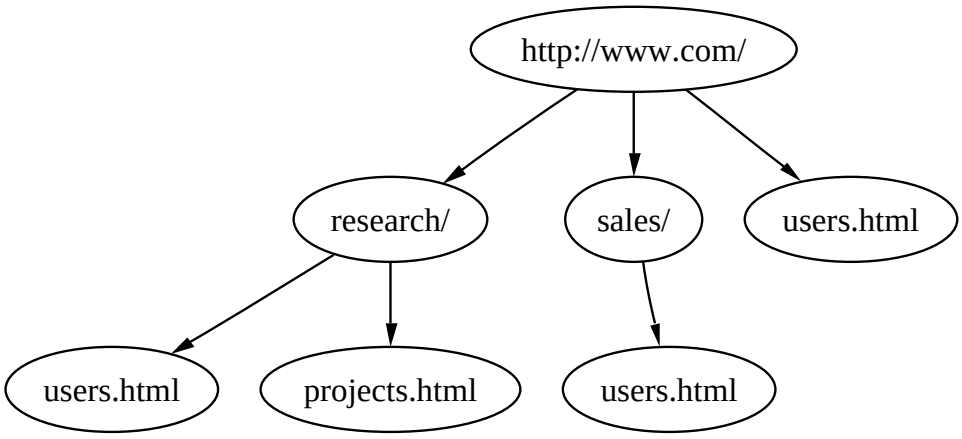


Figure 9.1: A small sample of a web hierarchy.

9.2 Propagation Algorithm

Looking at the web as a collection of nodes (web pages) and edges (links) yields the graph corresponding to the part of the WWW that we want to model. The nodes in the graph are further augmented with meta data for the corresponding web page. This graph can be used to generate meta data for web pages lacking actual meta data using the algorithms given later in this paper.

Meta data propagation is controlled by the trust level and pervasiveness that we attribute to a each node in the graph. Pervasive meta data might propagate further than one step. Propagation will stop when reaching nodes with higher trust levels.

We propose the following rules for meta data extraction and propagation:

1. If we have meta data extracted from a web page we would retain this data for the corresponding node (possibly reformatted and altered).

2. If meta data exists for a path that is a prefix of the requested web page then it is used for the node instead.

3. Depending on how much we trust the meta data from a certain node we have different ways of propagating the meta data given: The higher trust we put in the meta data of a node, the higher the probability that it will be propagated to those nodes that there are edges to. We have found that the *intersection* of all meta data represented in those web pages that have a edge to a node (looking at only those that have the highest trust level among the incoming links) will yield a system that is consistent and well behaved.

   The different trust levels are not defined here. We view trust levels as an enumerated value ranging from "no trust" to "full trust". Conversion to and from these values must be defined in the domain that they are to be used depending on the requirements of the surrounding systems.

4. Moreover, some sites may be marked as pervasive, meaning that meta data from that site will propagate recursively to nodes further than one link away. It is held in check by pages already marked in the steps 1 or 2 of this algorithm, and will not propagate to pages marked in 3 unless the new meta data has a higher trust than already given.

We do *not* advocate setting all meta data systems to pervasive and fully trusted. Pervasive propagation should only be used when the propagation can be controlled in some other way. Uncontrolled propagation would yield an increasing amount of data to propagate to all nodes in the graph, diminishing the value and truthfulness of the meta data. The resulting graph from using pervasive classification of all instances of given meta data will yield a graph where all nodes will be marked with (possible diluted and/or wrong) meta data. The main reason for this is that everything that one can find a link chain to without data given by rules 1 or 2 will be set to the meta data of the source.

Given the small example in Figure 9.2 we can show the differences between trust levels, i.e. meta data propagation using rules 3 and 4. If *B* is marked as a low trust system we can find the following propagations:

- *D* will be marked with *x*.
- If *A* is more trusted than *C* then *E* would be marked with *x*.
- If *C* is more trusted than *A* then *E* would be marked with *y*.
- If *A* and *C* are equally trusted then *E* would be marked with the intersection of *x* and *y*.
- If *A* is marked as pervasive and has at least the same trust as *B* then *F* would be marked with *x*, since the value given to *D* would continue to propagate.

  It would not propagate if *B* is more trusted than *A*, since *D* would get the meta data from *B* rather than *A*. Just because *A* and *B* have the same meta data is not sufficient reason for further propagation of the meta data to *F*.[1]

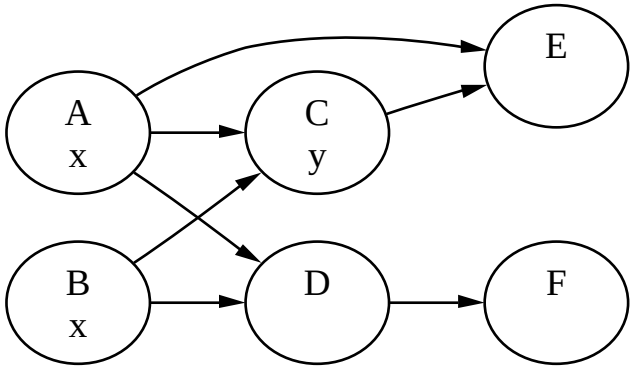


Figure 9.2: Sample graph with nodes ($A \ldots F$) marked with meta data ($x \ldots y$).

[1]This might seem counterintuitive (since both A and B agree on the evidence) but our algorithm uses only the highest level of trust.

9.3 Definitions

The following definitions are the ones typically found in works about compilers, e.g. [6, 18, 2], but adapted to the needs of this paper.

DEFINITION 9.1 A **semilattice** is a set L with a binary **meet** operation $\wedge$ such that for all $a,b,c \in L$:

1. $a \wedge a = a$ (idempotent)
2. $a \wedge b = b \wedge a$ (commutative)
3. $a \wedge (b \wedge c) = (a \wedge b) \wedge c$ (associative)

DEFINITION 9.2 A semilattice has a **zero element 0** iff $a \wedge \mathbf{0} = \mathbf{0}$ for every $a \in L$. L has a **one element 1** iff $a \wedge \mathbf{1} = a$ for every $a \in L$.

COROLLARY 9.1 If **0** exists, then it is unique. This holds true for **1** as well.

DEFINITION 9.3 If $(L, \wedge)$ is a semilattice and a and b are arbitrary elements L then we can define a relation $\leqslant$ in L:

$$a \leqslant b \Longleftrightarrow a \wedge b = a$$

The $<$, $>$ and $\geqslant$ relations can be defined in a similar way.

COROLLARY 9.2 Let $(L, \wedge)$ denote a semilattice and $\leqslant$ the relation introduced in Definition 9.3. Then $\leqslant$ is a partial order on L.

DEFINITION 9.4 A **chain** is a sequence $a_1, a_2, \ldots$ of elements from a semilattice L iff $a_i > a_{i+1}$ for all $i = 1, 2, \ldots$

DEFINITION 9.5 A semilattice L is **bounded** iff for every $a \in L$ there exists a $c_a \in \mathbb{N}$ such that the length of every chain beginning with a is at most c_a.

DEFINITION 9.6 A total function $f : L \rightarrow L$ is **monotonic** iff for all $a,b \in L$: $f(a \wedge b) \leqslant f(a) \wedge f(b)$.

DEFINITION 9.7 A **monotone data flow system** (MDS) is a tuple $\Omega = (L, \wedge, F, G, FM)$, where:

1. $(L, \wedge)$ is a bounded semilattice with **0** and **1**.
2. F is a monotonic function space for L.
3. $G = (N, E)$ is a directed graph modelling the web, with web pages as nodes and the links between the web pages as edges. This would normally also contain a start node *s*, but since the WWW is not a totally connected graph we will ignore start nodes and ordering between nodes in the system.
4. $FM : N \to F$ is a total function over N.

9.4 A Monotone Data Flow System on Meta Data over Web Pages

We can now look at the web as a directed graph $G = (N, E)$, with web pages and paths as the nodes N and the links represented by the edges E. There are a few basic rules that must apply in order to get a functioning and stable system:

1. Nodes that contains meta data that is valid for a path (including an entire web server) and everything beneath it in the site tree must be seen as highly trusted systems and have links to *at least* the web pages beneath it in the tree.
2. The meta data directly attributed to a specific node *n* in the system must be marked as such, and will not be changed later on by the algorithm.

Furthermore, all meta data must be marked with the trust given to it and how pervasive it is. The data to be distributed over the graph is the meta data given to the system at start-up. In our system we have:

$$(L, \wedge) = (\mathbb{P}(\text{meta data}), \cap),$$
$$\mathbf{0} = \varnothing \text{ and}$$
$$\mathbf{1} = \text{meta data}.$$

The definitions in Section 9.3 can then be used to model our web of meta data and web pages using algorithm 9.1 on the following page and 9.2 on page 113 (a heavily rewritten *general iterative algorithm* [6]).

The result of the algorithms are found in *trust* (only used as an intermediate result between the two algorithms) and *INF*. *INF* is meant to supersede the given value of *FM* in the final MDS.

ALGORITHM 9.1 (HANDLES ALL NON-PERVASIVE DATA)

Input: *An MDS* $\Omega = (L, \wedge, F, G, FM)$ *with* $G = (N, E)$
- $INF(n)$: *The actual meta data associated with node n. Given by rules 1 or 2, otherwise undefined.*
- $stabled(n)$: *Array of booleans marking that this node got its value by rules 1 or 2*
- $trust(n)$: *The trust level of the meta data given to the current node, enumeration or other values*
- $pervasive(n)$: *The data of this node is pervasive*

Output:
- $INF(n)$: *See above, but updated by the algorithm*
- $trust(n)$: *See above, but updated by the algorithm*

Variables:
- $n \in N$: *The node that we are currently looking at*
- $new(t)$: *Possible meta data for current node, per trust level*
- $N' \in \mathbb{P}N$: *Stable but not pervasive nodes*
- t, hi : *Temporary variables of trust levels*

begin
  Initialize
  foreach $n \in N \bullet undefined(INF(n))$ **do**
    INF(n) $\leftarrow$ **0**;
    trust(n) $\leftarrow$ *no trust;*
  od
  $N' \leftarrow \{n \in N \mid stabled(n) \wedge \neg pervasive(n)\}$;
  Handle all non-pervasive meta data once
  foreach $n \in N \bullet \neg stabled(n)$ **do**
    Initialize required data
    foreach $t \in$ *trust levels* **do**
      $new(t) \leftarrow$ **0**;
    od
    $hi \leftarrow$ *no trust;*
    Find intersection of incoming links with highest trust
    foreach $n' \in N' \bullet (n', n) \in E$ **do**
      $t \leftarrow trust(n')$;
      $new(t) \leftarrow new(t) \bigcap INF(n')$;
      if $t > hi$ **then** $hi \leftarrow t$; **fi**
    od
    Update INF if a change has been found
    if $t >$ *no trust*
      then $INF(n) \leftarrow new(t); trust(n) \leftarrow t$;
    fi
  od
end

ALGORITHM 9.2 (MODIFIED ITERATIVE ALGORITHM)

Input:	*An MDS* $\Omega = (L, \wedge, F, G, FM)$ *with* $G = (N, E)$	
	$INF(n)$,	
	$trust(n)$:	*As given by algorithm 9.1 on the facing page*
	$stabled(n)$,	
	$pervasive(n)$:	*As in algorithm 9.1 on the preceding page*
Output:	$INF : N \to L$,	*A total function*
Variables:	n, t, hi, $new(t)$:	*As in algorithm 9.1 on the facing page*
	$N' \in \mathbb{P}N$:	*Pervasive nodes*
	$stable \in Boolean$:	*Have we reached a stable state?*

```
begin
    Repeat data propagation until stable
    stable ← false;
    while ¬stable do
        stable ← true;
        N' ← {n ∈ N | pervasive(n)};
        Check for incoming meta data
        foreach n ∈ N • ¬stabled(n) do
            Initialize required data
            foreach t ∈ trust levels do
                new(t) ← 0;
            od
            hi ← no trust;
            foreach n' ∈ N' • (n', n) ∈ E do
                t ← trust(n');
                new(t) ← new(t) ⋂ |INF|(n');
                if t > hi then hi ← t; fi
            od                                              foreach n'
            if hi = trust(n)
                then new(hi) ← new(hi) ⋂ INF(n);
            fi
            if hi ≥ trust(n)
            then
                INF(n) ← new(hi);
                trust(n) ← hi;
                pervasive(n) ← true;
                stable ← false;
            fi
        od                                                  foreach n
    od                                                      while ¬stable
end
```

9.5 Related Work

This work builds on all previous forms of web mining [8] of semi-structured (i.e. HTML) data. Typical examples of this includes wrapper induction like STALKER [11], and information extraction like RSV [5] or WHISK/CRYSTAL [15, 16].

Extracting the information from the web was however only the first step; we are more interested in how meta data can be viewed outside of the web. The extracted data can either be seen as a data base over the web [12] or as a source for web structure mining such as HITS [7], Clever [3], PageRank and Google [1].

9.6 Discussion

We have used the algorithms described in this paper to model the web structure (and meta data content) of Umeå University. The university is a medium sized university in northern Sweden with approximately 25,300 undergraduate and 1,300 graduate students. Its web structure contains less than 100 official web servers with a total of more than 200,000 static web pages (counting only HTML pages, not pictures and other binary data).

Very reliable results from this data set has been obtained when none of the meta data has been marked pervasive. Trust levels were set, in decreasing order, according to 1) individual web pages containing meta data, 2) meta data for a sub-tree in a hierarchy, and 3) for the corresponding server.

We have checked the validity of the given meta data. Most (approximately 95%) of the checked individually marked web pages had correct meta data set. Almost all sub-trees had correct meta data (less than 1% contained errors) and the meta data given on the server level were 100% correct.

Looking at this data set we find that $\approx 3\%$ of the pages contain embedded meta data keywords. Applying rule 1 in Section 9.2 makes this value jump to $\approx 55\%$ and rule 2 increases this even further to $\approx 67\%$. Setting some of the nodes/web pages pervasive might yield an even higher percentage, depending on which nodes are marked pervasive.

9.7 References

[1] BRIN, S., AND PAGE, L. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems 30*, 1–7 (1998), 107–117.

[2] BURKE, M. An interval-based approach to exhaustive and incremental interprocedural data-flow analysis. *ACM Transactions on Programming Languages and Systems (TOPLAS) 12*, 3 (1990), 341–395.

[3] CHAKRABARTI, S., DOM, B. E., AND INDYK, P. Enhanced hypertext categorization using hyperlinks. In *Proceedings of SIGMOD-98, ACM International Conference on Management of Data* (Seattle, US, 1998), L. M. Haas and A. Tiwary, Eds., ACM Press, New York, US, pp. 307–318.

[4] EVANS, C., FEATHER, C. D., HOPMANN, A., PRESLER-MARSHALL, M., AND RESNICK, P. *REC-PICSRules-971229: PICSRules 1.1*. The World Wide Web Consortium, Cambridge, Massachusetts, Dec. 29, 1997.

[5] FREITAG, D. Information extraction from HTML: Application of a general machine learning approach. In *AAAI/IAAI* (1998), pp. 517–523.

[6] KAM, J. B., AND ULLMAN, J. D. Global data flow analysis and iterative algorithms. *Journal of the ACM (JACM) 23*, 1 (1976), 158–171.

[7] KLEINBERG, J. Authoritative sources in a hyperlinked environment. In *Proc. of ACM-SIAM Symposium on Discrete Algorithms* (1998), pp. 668–677.

[8] KOSALA, AND BLOCKEEL. Web mining research: A survey. *SIGKDD: SIGKDD Explorations: Newsletter of the Special Interest Group (SIG) on Knowledge Discovery & Data Mining, ACM 2* (2000).

[9] KRAUSKOPF, T., MILLER, J., RESNICK, P., AND TREESE, W. *REC-PICS-labels-961031: PICS Label Distribution Label Syntax and Communication Protocols*. The World Wide Web Consortium, Cambridge, Massachusetts, Oct. 31, 1996.

[10] MILLER, J., RESNICK, P., AND SINGER, D. *REC-PICS-services-961031: Rating Services and Rating Systems (and Their Machine Readable Descriptions)*. The World Wide Web Consortium, Cambridge, Massachusetts, Oct. 31, 1996.

[11] MUSLEA, I., MINTON, S., AND KNOBLOCK, C. A. Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems 4*, 1/2 (2001), 93–114.

[12] NESTOROV, S., ABITEBOUL, S., AND MOTWANI, R. Extracting schema from semistructured data. In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data* (New York, NY, USA, 1998), ACM Press, pp. 295–306.

[13] RESNICK, P. Filtering Information on the Internet. *Scientific American* (Mar. 1997), 106–108.

[14] RESNICK, P., AND MILLER, J. PICS: Internet Access Controls Without Censorship. *Communications of the ACM 39*, 10 (1996), 87–93.

[15] SODERLAND, S. Learning information extraction rules for semi-structured and free text. *Machine Learning 34*, 1-3 (1999), 233–272.

[16] SODERLAND, S., FISHER, D., ASELTINE, J., AND LEHNERT, W. CRYSTAL: Inducing a conceptual dictionary. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (San Francisco, 1995), C. Mellish, Ed., Morgan Kaufmann, pp. 1314–1319.

[17] WEINBERG, J. Rating the Net. *Hasting Communications and Entertainment Law Journal 19*, 2 (1997), 453–482.

[18] ZIMA, H., AND CHAPMAN, B. *Supercompilers for Parallel and Vector Computers*. Addison-Wesley, Reading, Massachusetts, 1990.

[19] ÅGREN, O. Reuse via the World Wide Web: How to Find the Software Required for Reuse. Master's thesis, Umeå University, Umeå, Sweden, Dec. 1998. UMNAD 242.98.

V

Chapter 10

Assessment of WWW-Based Ranking Systems for Smaller Web Sites

Abstract

A comparison between a number of search engines from three different families (HITS, PageRank, and Propagation of Trust) is presented for a small web server with respect to perceived relevance. A total of 307 individual tests have been done and the results from these were disseminated to the algorithms, and then handled using confidence intervals, Kolmogorov-Smirnov and ANOVA. We show that the results can be grouped according to algorithm family, and also that the algorithms (or at least families) can be partially ordered in order of relevance.

Keywords: Assessment, search engines, HITS, PageRank, Propagation of Trust, and eigenvectors

Received January 26, 2006, and accepted for publication on March 14, 2006, in INFOCOMP Journal of Computer Science.

10.1 Introduction

Finding the required information on the WWW is not a trivial task. Currently used search engines will usually give good advice on pages to look at, but are there more personalised tools that can be used instead? We will in this work compare the relative strength of some of the algorithms usable in a user defined personalisation environment, in order to find out how they behave over smaller networks (such as a single web server).

Two things that these algorithms have in common is that they operate on a connection matrix (or adjacency matrix), and they all use a set of pages that are known to be about a specific topic (called *known pages*) as the starting point. The algorithms belong to three different families:

Hypertext-Induced Topic Selection (HITS [8]) This family of algorithms does not work on the entire connection matrix, but will instead use a subset of this matrix called H. It includes the known pages together with pages pointing at one or more pages among the known pages as well as the pages pointed out by them. Each page in the entire set is given a start value in two categories, "hub" (denoting an important link page) and "authority" (denoting a page with valuable information on the given subject). These values are adjusted by iteration and normalisation over the simultaneous equations given in Eq. (10.1).

$$h_i = \sum_{(i,j)\in E} a_j \qquad a_j = \sum_{(i,j)\in E} h_i \tag{10.1}$$

The basic idea behind HITS is to use the inherent strength of the connection matrix. The starting point is to give a value to all pages in the known set, and then calculate the final result by propagating these values first in the forward direction of H (giving a partial result of the authority pages), then back into the hub value until the calculations are stable.

The two algorithms of the HITS family that we will use are the original **HITS** [8] algorithm as well as the **Randomized HITS** [11] algorithm, where some of the hub/authority value given to each page is dissapated to all pages in the set. There are other versions in this family, including:

Subspace HITS [11], where all stable eigenvectors found are multiplied with their relative eigenvalue strength and these are then superpositioned,

Clever [5], where the connection matrix is slightly changed by weighting according to the number of incoming/outgoing hyper-links as well as whether the pages resided on the same site or not[1],

MHITS [10], where the connection matrix is generated using web logs as well as more than one link away from the original starting page,

BHITS [1], where two things are used in order to make HITS more stable; outliers are filtered out and the weight of links between two servers are one divided by the number of links, and

Stochastic Approach to Link Structure Analysis (SALSA) [9], where H and H' are updated in order to get stochastic matrices (by dividing each value in a row with the number of values in the row). The main reason for doing this is to get a sound and stable system, but the final outcome is that for systems without dangling nodes (or weights) we get a result directly related to the in- (for authority value) and out-degree (for hub value). This can be computed much faster with other techniques.

Clever and BHITS were ruled out since all pages resided on one web server and Subspace HITS were removed since our search engine framework was unfortunately not able to support them fully. No web logs were available, thus ruling out MHITS, and SALSA did not give sufficiently fine-grained results for weighted disseminations. This left us with HITS and Randomized HITS from this family.

[1] A later version of Clever breaks up pages with a vast amount of outgoing links into micro-pages, each with its own fine-grained hub value [3]. These micro-pages are not seen as entirely separate entities, and a secondary aggregate hub/authority value can be calculated for them as well.

PageRank This is the main algorithm of Google [2], and is used to give a query independent importance number (called a *rank value*) to each web page according to the structure of hyper-links between web pages.

PageRank uses a random surfing model over the Internet. This means that it models the behaviour of a web surfer that follows one randomly chosen link in the current page, every once in a while this web surfer gets bored with the current chain of pages and skips to a random page on the Internet (called the *damping factor*). Each visit to a page would in theory indicate that a page gets slightly more interesting than before. Rather than letting a simulator mark each visited page a number of times, there are much more effective ways of simulating and calculating these values.

The probability that the web surfer will visit page w_j is given in Eq. (10.2), using $(1-\mu) < 1$ as the dampening factor, the graph $G = (V,E)$ where V is the set of pages and E is the set of hyper-links, $n = |V|$, and $d(w_i)$ is the out-degree of page w_i.

$$PR(w_j) = \frac{1-\mu}{n} + \mu \sum_{(i,j)\in E} \frac{PR(w_i)}{d(w_i)} \tag{10.2}$$

This is the same thing as using a connection matrix where each column sums to 1 modified by adding the dampening factor as can be seen in Eq. (10.3).

$$P = \left[\frac{1-\mu}{n}\right]_{n\times n} + \mu M \tag{10.3}$$

The PageRank is the dominating eigenvector of P: $P\pi = \pi, \pi \geqslant 0, ||\pi||_1 = 1$. This means that the i-th entry of π is the probability that a surfer visits page i, or the PageRank of page i.

The version that we will use in this work is **Topic-Sensitive PageRank** [7]. It uses the same general ideas and algorithm as the normal PageRank, except that skipping will be to one of the known pages; the dampening factor is only added (and scaled accordingly) if the corresponding page is known to be about that particular subject.

Propagation of Trust This set of algorithms builds on the algorithm found in [12]. The main idea is that the trust of known pages are distributed (and diminished by $1/\xi$, where $\xi > 1$) over each outgoing link, until the value is too small to make a difference any more. We will use three different versions of this algorithm in this work:

Basic Propagation of Trust (ProT) Given an initial score $\varpi(j,0) = 1$ (100%) for pages that are on-topic and zero otherwise, and using k as the iteration count as well as setting ξ to be a value just over the dominant eigenvalue of the corresponding connection matrix we can apply the algorithm in Eq. (10.4).

$$\varpi(j,k) = \frac{1}{\xi} \sum_{(i,j)\in E} \varpi(i,k-1) + \begin{cases} \varpi(j,k-1) & j \text{ is on-topic} \\ 0 & \text{otherwise.} \end{cases} \tag{10.4}$$

The final answer is given after normalisation of the k:th ϖ vector.

Superpositioned Singleton Propagation of Trust (S^{2}ProT) This algorithm is a much faster replacement for ProT. For each page among the known pages we calculate a singleton (or basic) vector using ProT, and then superposition these vectors to form the final answer. Calculating a singleton is usually much faster than using the set of known pages directly in ProT.

Hybrid Superpositioned Singleton Propagation of Trust (HyS^{2}ProT) This is a hybrid version of S^{2}ProT where each outgoing value is further decreased with the number of outgoing links (i.e. the out-degree of a page) in the same manner as for PageRank.

10.1.1 Hypotheses

We have two main hypotheses regarding smaller input sets (see Appendix 10.A on page 136 for a description of the input data set):

1. These algorithms will for this data set give top five results that are disparate when comparing different families of algorithms with each other.

2. The Propagation of Trust family gives better sets of top five results (with regards to relevancy) than any of the others for this data set.

10.2 Methods and Materials

In order to find out whether the algorithms in question yields roughly equally relevant results the following experiment was conducted: The top five results from applying the algorithms was gathered and combined for a number of keywords (see Section 10.2.1), yielding a total of between nine and 20 links per keyword. These links were manually examined for relevancy with regards to the corresponding keyword by those participating in the experiment, yielding a total of 307 tests of the ten keywords. The relevances were then propagated back to the corresponding algorithms as per Section 10.2.2 in order to find out how relevant the mean result of each algorithm are, both per question and in total. The total relevancy of each algorithm were then compared using confidence intervals (see Section 10.2.3) and Kolmogorov-Smirnov (see Section 10.2.4) to show whether there are differences between each algorithm in terms of relevancy.

10.2.1 Keyword Selection Criterias

The first criteria was that at least eight pages had to contain a keyword in order for it to be eligible for inclusion in this experiment. The reason for this is that there should be a basis for variability of the top lists of each algorithm. One keyword that appears in only one page was still kept here, since variation between HITS and Randomized HITS was apparent with this keyword.

The second criteria was that combining the results from the algorithms should yield at least nine unique links in the resulting set. Once more, this rule also stems from the variability of the top lists.

The third criteria was that the pages had to be present during the experiment. Some pages have been removed since the database was gathered, thus preventing some otherwise fitting keywords from being used.

The fourth criteria was that no student web page should be included in the top lists, so that no single individual of the student body should feel singled out.

The fifth criteria was that at most 20 links should be given when combining the results of the algorithms. The reason for this is that the workload of those participating in the experiment should be reasonably small.

The sixth and final criteria was that not all pages given by an algorithm should have exactly the same weight attributed to them, in order to use the weights to find differences between the data sets.

These criteria yielded a list of a few hundred suitable candidates for inclusion in the experiment. The final selection was done using two methods — direct selection of some keywords that we felt were well defined (in this case ‘aagren’, ‘jubo’, ‘kompilatorteknik’, and ‘ola’), while the others were selected at random.

10.2.2 Dissemination of Result to Algorithms

The relevance checking was done using blind reviews (i.e. no reference was given as to what or which algorithm(s) produced the link in the top five positions) on a five-graded scale. The first grade indicated that the reviewer was unable to say anything about the relevancy of the page regarding the current keyword, and these values were ignored in all calculations. The second grade corresponded to a complete lack of relevancy, i.e. 0. The third grade was indicative of some relevance, i.e. $\frac{1}{3}$. The fourth grade indicated a moderate amount of relevance, i.e. $\frac{2}{3}$. The fifth and final grade indicated that the page was very relevant, corresponding to 1 (or 100% relevance). This scaling will lead to an underestimation of the true relevance of the pages, but we are interested in the relative rather than exact relevance here.

10.2.2.1 Original Dissemination

The pages given in the top lists for each algorithm shows which pages should be included in each dissemination. The values corresponding to each grade were summed up and then divided by the number of grades that did not belong to the first grade, thereby forming a mean relevancy for that keyword and algorithm combination according to that reviewer.

As an example, consider the top-list P containing the five pages A, B, C, D and E. These were given the grades A - grade one, B - grade two, ..., E - grade five. This means that the mean relevancy for top-list P from this grader was

$$\frac{0+1/3+2/3+1}{4} = 0.5 \text{ or } 50\%.$$

10.2.2.2 Weighted Dissemination

The algorithms supply not only the list of pages, but also gives a weight $\in (0,1]$ for each page. For all pages with grades higher than the first, add up both the product of page weight and the corresponding page relevancy and the sum of the weights. The final number is given by dividing the sum of products with sum of weights. The rationale here is that the higher weight attributed to them by an algorithm, the more important that page should be for the final score.

We can continue the example above by saying that the weight corresponding to page A is $a = 0.6$, weight of page B is $b = 0.7$, ..., and weight of page E is $e = 1.0$. Given the same grading as in the previous section the weighted relevancy would be

$$\frac{0.7 \times 0 + 0.8/3 + 0.9 \times 2/3 + 1}{0.7 + 0.8 + 0.9 + 1} \approx 0.5490/54.90\%.$$

10.2.3 Confidence Interval Comparisons

The mean result can be used to rank the algorithms according to relevancy. Moreover, by forming a confidence interval around this mean it is possible to show whether the results from the disseminations are disparate.

10.2.4 Kolmogorov-Smirnov Comparisons

One of the most widely used goodness-of-fit tests available is the Kolmogorov-Smirnov. It uses the maximum difference D in y values between two curves plotted in a cumulative fraction plot, i.e. going in discrete steps of 1/#steps from 0 to 1 from left to right. This difference is then compared to a number that depends on both the chosen α level (in our case 0.001) and the number of samples in the set. The number of samples to use in the comparison is calculated from the original number of samples for each input set (n_1 and n_2, respectively):

$$n = \frac{n_1 \times n_2}{n_1 + n_2}$$

10.3 Results

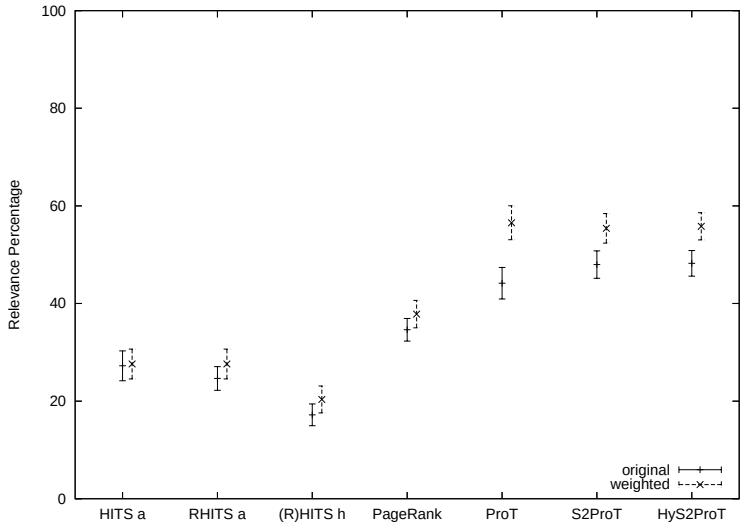


Figure 10.1: This graph shows the mean relevance values as well as 95% confidence intervals per algorithm, both unweighted and weighted.

10.3.1 Confidence Interval Comparisons

10.3.1.1 Original

Both the 95% confidence intervals plotted in Figure 10.1 as well as the 99.9% confidence intervals given in Table 10.1 on the next page leads to the same result; we can divide the algorithms into four groups. The top group contains HyS^2ProT, S^2ProT and ProT, with each confidence interval encompassing the mean value of the others. The second group contains only one algorithm, Topic-Sensitive PageRank. The third group contains HITS Authority and Randomized HITS Authority, and (Randomized) HITS Hub is the algorithm that is the sole member of the last group.

Table 10.1: The mean relevance values and their 99.9% confidence intervals, given in descending order.

Algorithm	Mean	99.9% conf. inter.
HyS^2ProT	0.4823	(0.4383,0.5263)
S^2ProT	0.4797	(0.4325,0.5270)
ProT	0.4416	(0.3872,0.4959)
Topic-Sensitive PageRank	0.3462	(0.3073,0.3851)
HITS Authority	0.2723	(0.2210,0.3237)
Randomized HITS Authority	0.2465	(0.2057,0.2873)
(Randomized) HITS Hub	0.1719	(0.1344,0.2094)

10.3.1.2 Weighted

The same four groups with overlapping confidence intervals can be found in the weighted result set as well as can be seen in Figure 10.1 on the facing page and Table 10.2. The first group consists of HyS^2ProT, S^2ProT and ProT, the second of Topic-Sensitive PageRank, the third of both versions of HITS Authority and the final group contains (Randomized) HITS Hub.

Table 10.2: The mean weighted relevance values and their 99.9% confidence intervals, given in descending order.

Algorithm	Mean	99.9% conf. inter.
ProT	0.5654	(0.5070,0.6237)
HyS^2ProT	0.5582	(0.5114,0.6050)
S^2ProT	0.5540	(0.5033,0.6047)
Topic-Sensitive PageRank	0.3783	(0.3312,0.4253)
HITS Authority	0.2761	(0.2248,0.3273)
Randomized HITS Authority	0.2761	(0.2248,0.3273)
(Randomized) HITS Hub	0.2034	(0.1569,0.2500)

10.3.2 Kolmogorov-Smirnov Comparisons

For each combination of result lists, we put up the following hypothesis:

H_0 : The distribution of the two lists are equal.

H_1 : The distribution of the two lists are not equal.

The Kolmogorov-Smirnov test is then applied to the combination in order to either reject or accept H_0.

10.3.2.1 Original

The results from these comparisons is that ProT, S^{2}ProT and HyS^{2}ProT have almost identical distribution, as does HITS Authority and both Randomized HITS Authority and (Randomized) HITS Hub. All other combinations are disparate at the 99.9% certainty level. The corresponding cumulative fraction plot can be seen in Figure 10.2. For full results see Appendix 10.D.

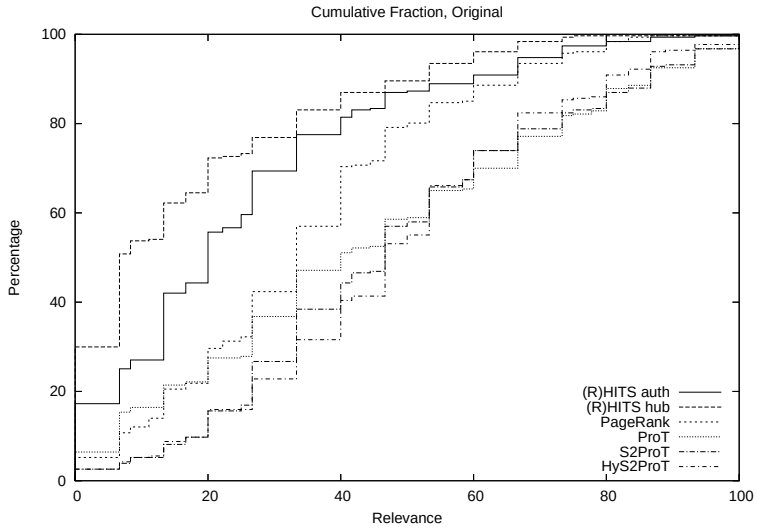


Figure 10.2: Cumulative fraction of answers that is at a certain level or lower.

10.3.2.2 Weighted

The results from these comparisons is that HITS Authority and Randomized HITS Authority have almost identical distribution, and S^2ProT has a distribution that is very close to both ProT and HyS^2ProT (while these two are disparate). All other combinations are disparate at the 99.9% certainty level. The corresponding cumulative fraction plot can be seen in Figure 10.3. For full results see Appendix 10.D.

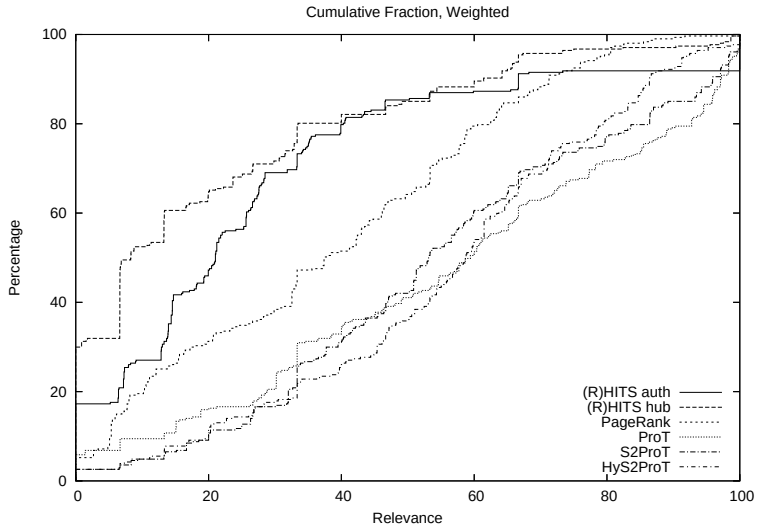


Figure 10.3: Weighted cumulative fraction of answers that is at a certain level or lower.

10.4 Discussion and Conclusions

Looking back at the two main hypotheses posed in the introduction (Section 10.1.1) we can see that both of them have been shown to be true with high significance:

1. The mean values from each algorithm of each family does neither appear in the confidence intervals of another (Section 10.3.1), nor will Kolmogorov-Smirnov retain H_0 of comparisons between families (Section 10.3.2) for both original and weighted values.

2. The values presented by the confidence intervals in Section 10.3.1 shows that a distinct relevance order can be seen among the algorithm families. The order is that Propagation of Trust yields better result than Topic-Sensitive PageRank, that in turn yields better result than HITS. This is also visible in Figure 10.2 on page 130 where this order (remember that a lower curve corresponds to better relevancy) can be clearly seen. This is also true for the largest part of Figure 10.3 on the previous page, even though some crossing of the graphs can be seen.

The Hub lists of HITS and Randomized HITS are identical in the unweighted version, while the weighted version shows some minor differences (D = 0.0749). A slightly larger difference can be seen when looking at the authority scores, with difference in distribution of (D = 0.07818) or $\alpha = 0.2890$.

There is a slightly more complex situation among the algorithms of the Propagation of Trust family. While looking at the original comparisons at the high significance level we are unable to reject that each mean value *could* come from one of the other algorithms. Looking at Figure 10.2 on page 130 and Figure 10.3 on the previous page gives a clear indication that there *is* in fact some minor differences between the algorithms. The only way to show this is to increase α, and the α required to show that the distributions are disparate can be seen in Table 10.3.

Table 10.3: This table shows the α that must be chosen in order to show that the distributions from the algorithms are disparate.

	Original		Weighted	
	S^2ProT	HyS^2ProT	S^2ProT	HyS^2ProT
ProT	0.0074	0.0024	0.0988	0.0004
S^2ProT		0.4526		0.0528

The results given in Section 10.3.1 has been confirmed by using ANOVA tests, where only tests between algorithms of the same family have p-values of 0.001 or higher. The relevant tests are between:

- HITS Authority and Randomized HITS Authority ($f = 1.6811, p = 0.1953$),
- ProT and S^{2}ProT ($f = 3.0072, p = 0.0834$),
- ProT and HyS^{2}ProT ($f = 3.6335, p = 0.0571$), and, finally,
- S^{2}ProT and HyS^{2}ProT ($f = 0.0169, p = 0.8966$).

Our conclusion of this experiment is that not only does the algorithms in the Propagation of Trust family yield good results even for smaller databases, they give better results than the competition. The main reason for the lower results of Topic-Sensitive PageRank is probably the relative lack of links, the more links (and pages) the better it seems to be working. There are on the other hand two reasons for the lower than expected results from HITS:

- The first reason is that some pages that came from the hub lists do not talk about a subject directly but have lots of links to pages that does.
- The second reason is that HITS suffer from mutually reinforcing relationships between pages among the included pages as well as topic drift, where a tight-knit community of pages can take over as the most important pages for a query.

The scaling could be improved on an intuitive level by using a more sensible scale (such as ignore, 0%, 50%, 75% and 100%) if more exact relevance number were required. We have opted to continue with this scaling, since rescaling would not affect the final result.

One thing that could be done to get even more information per keyword is to look at more than 5 links per list. This method have the drawback that the number of links to process for those participating in the experiment increases almost linearily, so that an increase from 5 to 10 links per algorithm yields roughly twice as many links to check.

Another test that should be done is to look at a much larger database, preferrably the entire Internet. Since this data is not available at this time this is hardly feasible, even though we do have much larger databases to work on (such as the entire web structure at Umeå University). It would however be much harder to choose keywords to use, since even more criterias (such as pages on more than one web server) could be applied.

Hiding the true souce of each link rather than comparing each list directly was first seen in [6] (comparing Clever and Yahoo), since they found a distinct problem in their earlier comparisons that showed the entire result lists from each search engine/algorithm [4]. One set of the included result lists in the older test contained annotations and one-line summaries, thus yielding better information for the classifier to use when assessing relevancy. We must agree that using blind examinations for relevancy yields an objectively better result and should be used in future studies.

10.5 Acknowledgements

Many thanks to Leif Nilsson, Helena Lewandowska, and Mårten Forsmark for valuable comments and suggestions that were useful for improving the quality of this paper, as well as to the approximately 80 participants of the assessment.

10.6 References

[1] BHARAT, K., AND HENZINGER, M. R. Improved algorithms for topic distillation in a hyperlinked environment. In *SIGIR '98: Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval* (New York, NY, USA, 1998), ACM Press, pp. 104–111.

[2] BRIN, S., AND PAGE, L. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems 30*, 1–7 (1998), 107–117.

[3] CHAKRABARTI, S. Integrating the document object model with hyperlinks for enhanced topic distillation and information extraction. In *WWW '01: Proceedings of the 10th international conference on World Wide Web* (New York, NY, USA, 2001), ACM Press, pp. 211–220.

[4] CHAKRABARTI, S., DOM, B., RAGHAVAN, P., RAJAGOPALAN, S., GIBSON, D., AND KLEINBERG, J. Automatic resource compilation by analyzing hyperlink structure and associated text. In *WWW7: Proceedings of the seventh international conference on World Wide Web 7* (Amsterdam, The Netherlands, 1998), Elsevier Science Publishers B. V., pp. 65–74.

[5] CHAKRABARTI, S., DOM, B. E., AND INDYK, P. Enhanced hypertext categorization using hyperlinks. In *Proceedings of SIGMOD-98, ACM International Conference on Management of Data* (Seattle, US, 1998), L. M. Haas and A. Tiwary, Eds., ACM Press, New York, US, pp. 307–318.

[6] CHAKRABARTI, S., DOM, B. E., KUMAR, S. R., RAGHAVAN, P., RAJAGOPALAN, S., TOMKINS, A., GIBSON, D., AND KLEINBERG, J. Mining the web's link structure. *Computer 32*, 8 (1999), 60–67.

[7] HAVELIWALA, T. H. Topic-sensitive PageRank. In *Proceedings of the eleventh international conference on World Wide Web* (2002), ACM Press, pp. 517–526.

[8] KLEINBERG, J. Authoritative sources in a hyperlinked environment. In *Proc. of ACM-SIAM Symposium on Discrete Algorithms* (1998), pp. 668–677.

[9] LEMPEL, R., AND MORAN, S. SALSA: The stochastic approach for link-structure analysis. *ACM Trans. Inf. Syst. 19*, 2 (2001), 131–160.

[10] MILLER, J. C., RAE, G., SCHAEFER, F., WARD, L. A., LOFARO, T., AND FARAHAT, A. Modifications of Kleinberg's HITS algorithm using matrix exponentiation and web log records. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval* (New Orleans, Louisiana, United States, 2001), ACM Press, pp. 444–445.

[11] NG, A. Y., ZHENG, A. X., AND JORDAN, M. Stable algorithms for link analysis. In *Proceedings of the Twenty-fourth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Sept. 2001).

[12] ÅGREN, O. Propagation of Meta Data over the World Wide Web. In *Proceedings of the International Conference on Internet Computing (IC '03)* (Las Vegas, Nevada, USA, June 23-26, 2003), vol. 2, pp. 670–676. Paper IV on page 103.

10.A Test Database

The database used in this experiment contains a subset of all the pages available at www.cs.umu.se, the web site of the Department of Computing Science, Umeå University. This database was collected in January 2003.

The database contains 7312 pages, of which 2728 are HTML pages with outgoing links. There are a total of 22970 hyper-links, yielding an average of approximately 8.42 outgoing hyper-links per HTML page and just over 3.14 incoming links per page.

A total of 57823 keywords are present at least once in the entire set of pages. These keywords are present in on average 9.87 pages, for a total of 570870 page occurrences. 33854 of these keywords appear in the same set of pages as another keyword, so that only 23969 unique ranking lists are required for the entire set per algorithm.

Looking at the web site as a undirected graph we find that it contains 130 components, with 6885 pages in the largest component.

10.B Keywords

The keywords used in the assessment can be seen in Table 10.4.

Table 10.4: The keywords used in the tests.

Keyword	English	#pages in DB	#pages in test	#tests
aagren	Ågren[a]	227	17	34
choklad	chocolate	15	17	33
exempelrapport	sample report	1[b]	9	34
jubo	Jürgen Börstler[a]	110	19	30
kallin	Kallin[a]	161	19	37
kompilatorteknik	compiler construction/-techniques	23	18	31
konstant	constant	18	16	26
matrismultiplik	matrix multiplic[c]	8	16	19
ola	Ola[a]	251	17	37
relation	relation	17	14	26
			Sum:	307

[a] Proper name.

[b] Breaks the first selection criteria, but was included since all other criterias were met and it manifested a real difference between the authority lists of HITS and Randomized HITS.

[c] This keyword has been truncated by stemming.

10.C Confidence Intervals per Keyword

The 95% confidence intervals for each keyword given in Appendix 10.B on the preceding page (and Table 10.4 on the facing page) can be seen in Figure 10.4 and Figure 10.5 on the following page.

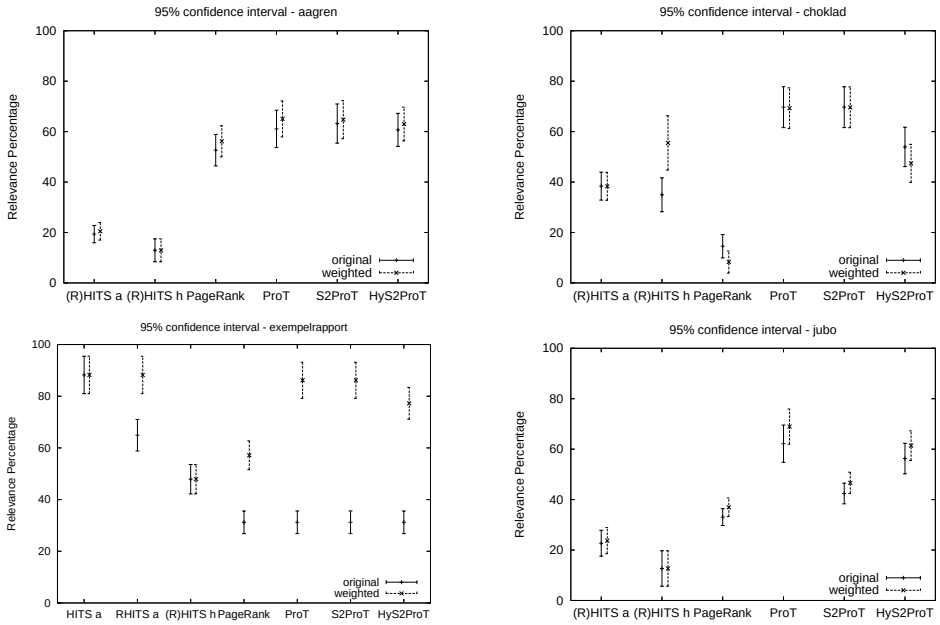


Figure 10.4: Mean relevance and 95% confidence intervals for the first four keywords.

10.D Full Kolmogorov-Smirnov Results from Comparisons

Table 10.5 on page 139 contains all results from applying Kolmogorov-Smirnovs tests on each of the 21 possible combinations of algorithms (counting the results from HITS Hub and Randomized HITS Hub values as equal). These values are given for both the original and for the weighted Kolmogorov-Smirnov tests.

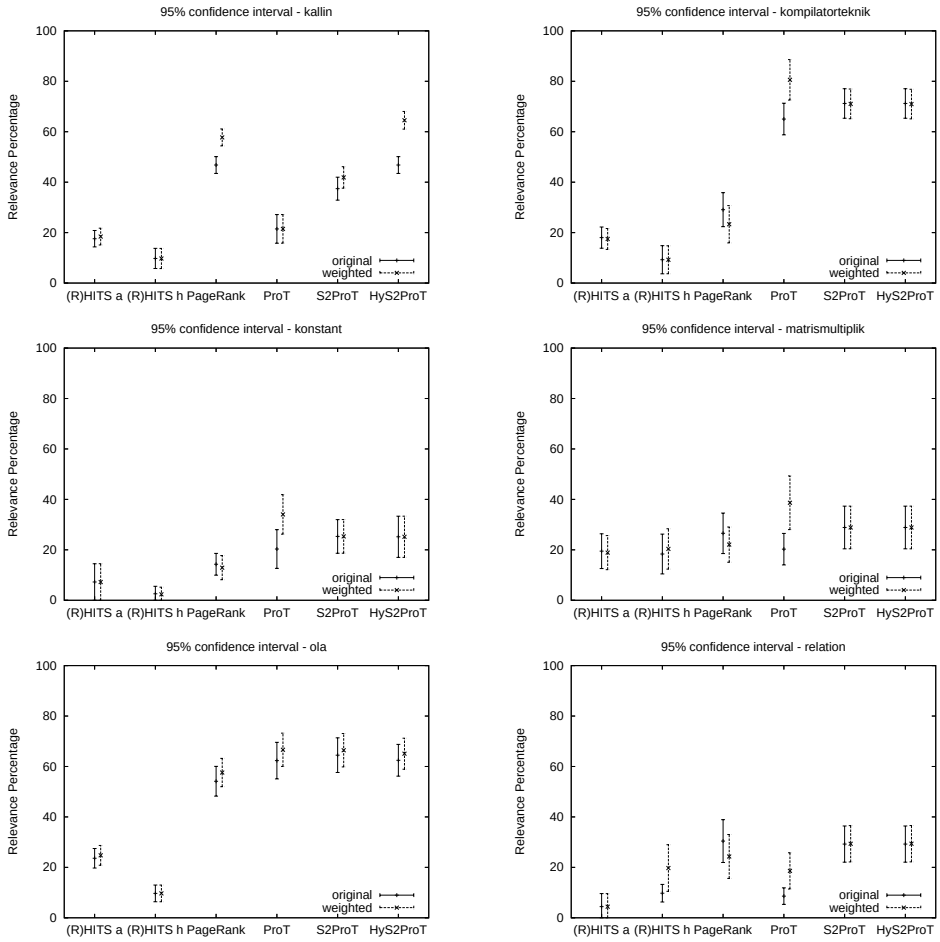


Figure 10.5: Mean relevance and 95% confidence intervals for the last six keywords.

Table 10.5: The full results from the Kolmogorov-Smirnovs tests. Both tables use a cut-off of 0.157 ($n = 154, \alpha = 0.001$) .

Original	Randomized HITS Authority	(Randomized) HITS Hub	Topic-Sensitive Page-Rank	ProT	S^2ProT	HyS^2ProT
HITS Authority	D = 0.0782 < 0.157, H_0 accepted	D = 0.2704 > 0.157, H_0 rejected	D = 0.2704 > 0.157, H_0 rejected	D = 0.3290 > 0.157, H_0 rejected	D = 0.4235 > 0.157, H_0 rejected	D = 0.4625 > 0.157, H_0 rejected
Randomized HITS Authority		D = 0.2704 > 0.157, H_0 rejected	D = 0.2736 > 0.157, H_0 rejected	D = 0.3322 > 0.157, H_0 rejected	D = 0.4267 > 0.157, H_0 rejected	D = 0.4658 > 0.157, H_0 rejected
(Randomized) HITS Hub			D = 0.4267 > 0.157, H_0 rejected	D = 0.4560 > 0.157, H_0 rejected	D = 0.5700 > 0.157, H_0 rejected	D = 0.5733 > 0.157, H_0 rejected
Topic-Sensitive PageRank				D = 0.2052 > 0.157, H_0 rejected	D = 0.2606 > 0.157, H_0 rejected	D = 0.3029 > 0.157, H_0 rejected
ProT					D = 0.1336 < 0.157, H_0 accepted	D = 0.1466 < 0.157, H_0 accepted
S^2ProT						D = 0.0684 < 0.157, H_0 accepted

Weighted	Randomized HITS Authority	(Randomized) HITS Hub	Topic-Sensitive Page-Rank	ProT	S^2ProT	HyS^2ProT
HITS Authority	D = 0.0782 < 0.157, H_0 accepted	D = 0.2932 > 0.157, H_0 rejected	D = 0.3225 > 0.157, H_0 rejected	D = 0.4951 > 0.157, H_0 rejected	D = 0.5244 > 0.157, H_0 rejected	D = 0.5505 > 0.157, H_0 rejected
Randomized HITS Authority		D = 0.2932 > 0.157, H_0 rejected	D = 0.3225 > 0.157, H_0 rejected	D = 0.4951 > 0.157, H_0 rejected	D = 0.5244 > 0.157, H_0 rejected	D = 0.5505 > 0.157, H_0 rejected
(Randomized) HITS Hub			D = 0.3550 > 0.157, H_0 rejected	D = 0.5342 > 0.157, H_0 rejected	D = 0.5700 > 0.157, H_0 rejected	D = 0.5831 > 0.157, H_0 rejected
Topic-Sensitive PageRank				D = 0.2899 > 0.157, H_0 rejected	D = 0.2410 > 0.157, H_0 rejected	D = 0.3062 > 0.157, H_0 rejected
ProT					D = 0.0977 < 0.157, H_0 accepted	D = 0.1629 > 0.157, H_0 rejected
S^2ProT						D = 0.1075 < 0.157, H_0 accepted

VI

Chapter 11

S^{2}ProT: Rank Allocation by Superpositioned Propagation of Topic-Relevance

Received February 19, 2008, was revised and accepted on May 29, 2008 for publication in *International Journal of Web Information Systems.*

Purpose - *The purpose of this paper is to assign topic-specific ranks to web pages.*

Methodology/Approach - *The paper uses power iteration to assign topic-specific rating values (called* relevance*) to web pages, creating a ranking or partial order among these pages for each topic. Our approach depends on a set of pages that are initially assumed to be relevant for a specific topic, the spatial link structure of the web pages, and a net-specific decay factor designated* ξ.

Findings - *The paper finds that this approach exhibits desirable properties such as fast convergence, stability and yields relevant answer sets. The first property will be shown using theoretical proofs, while the others are evaluated through stability experiments and assessments of real world data in comparison with already established algorithms.*

Research limitations/implications - *In the assessment, all pages that a web spider was able to find in the Nordic countries were used. It is also important to note that entities that use domains outside the Nordic countries (e.g., .com or .org) are not present in the paper's datasets even though they reside logically within one or more of the Nordic countries. This is quite a large dataset, but still small in comparison with the entire World Wide Web. Moreover, the execution speed of some of the algorithms unfortunately prohibited the use of our large test dataset in the stability tests.*

Practical implications - *It is not only possible, but also reasonable, to perform ranking of web pages without using Markov chain approaches. This means that the work of generating answer sets for complex questions could (at least in theory) be divided into smaller parts that are later summed up to give the final answer.*

Originality/value - *This paper contributes to the research on Internet Search Engines.*

Keywords *World wide web, Information retrieval, Spatial data structures, Search engines*

Paper type *Research paper*

11.1 Introduction

Web search is normally performed using search engines that process text queries, where each query is built up by a combination of terms (called *topics* in this paper). Since the web has grown to such an enormous size, we can expect the number of pages matching each text query to be huge as well. This means that there should be some way of deciding the order (or *ranking*) of the resulting pages as a support for the user. While many different methods of ranking pages on the internet exist, some have been more prolific than others. Probably the most important method relies on the fact that people tend to put in links to pages that they find interesting on roughly the same topics as their own pages, e.g. pages on table tennis tend to have links to other pages and sites on table tennis [10]. These links can be used in different ways to calculate relevance values. We see two major approaches for computing relevance values based on the structure of links between pages; topic-independent and topic-specific.

Topic-independent approaches generates one set of rankings, and then uses a subset of this to answer each query. This means that answering a question about pages that contain a specific search term boils down to looking up the ranking for the pages that contain that term. The challenge is to find a ranking order that says something for each and every topic, since the ranking algorithm does not care whether a search term is part of the page or not. The most prominent example of this approach is PageRank [23].

Topic-specific approaches instead generate one (or more) set of rankings for each topic. The set of rankings is tailor-made to this specific topic by using either weighting of certain pages or by starting with smaller sets of pages around the pages containing the search term. While this can often leads to smaller solutions sets, the challenge is still to create sets of rankings for a sufficiently large number of topics because the computational cost of generating each topic-specific ranking is usually quite large. Typical examples of this approach are Topic-sensitive PageRank [13] and HITS [16].

We propose a new approach for generating topic-specific rankings. The main idea of our approach is to start with an initial set Θ of pages that are assumed to be relevant for the topic. For instance, Θ may be the set of all pages containing a specific keyword. Each page in Θ is initially assigned a certain relevance value (1 in our tests), whereas all other pages are assigned the relevance value 0. The relevance values are then propagated and decreased in a controlled fashion over the network of links at hand (be it a single site, the entire Internet, or anything in between). This gives a relevance value for each page, that can be used to generate a ranking for these pages.

We have implemented a number of different algorithms based on our approach. Two of these will be described in detail and another in a cursory manner in this paper.

One thing that is important for all ranking algorithms is that they should preferably give roughly the same answer even if small perturbations such as missed links occur. The less susceptible to perturbations a given algorithm is, the more stable it is said to be. Our results show that our algorithms are very stable, yielding good results even when operating on small, site-specific data sets. We will show the stability using statistics from actual data. The more advanced version of the algorithm is also very fast and scalable as we will show.

Disadvantages of our approach are that it requires an initial set Θ of pages, and depends on a parameter that we denote by ξ. The former can be found by using, e.g. web directories such as Yahoo or by checking the content of a web page word by word (which is the method we used in our tests). The parameter ξ is an algorithm dependent decay factor; our basic algorithm requires ξ to be a close approximation of the dominant eigenvalue of the underlying network while the advanced version works better and faster when larger values of ξ are chosen, i.e. four or five times larger than the dominant eigenvalue.

11.1.1 Layout of this paper

Section 11.2 contains background material and definitions that are essential for the technical details of (but not central to) this paper. Section 11.3 describes related work. The section following it, Section 11.4, describes our algorithms in detail. The behaviour patterns of each algorithm are discussed in Section 11.5. Section 11.6 contains empirical results from running the algorithms, as well as comparisons of these results with PageRank and Topic-sensitive PageRank. The last section, Section 11.7, contains a discussion and some concluding remarks.

11.2 Preliminaries

This section sets up notations and terminology that are required for the article, even though it is not central to the work described herein.

11.2.1 Webs as Graphs

Throughout this paper, we will identify a web with a graph (as far as its link structure is concerned). For this, let V be the set of web pages and E be the set of hyper-links (and thus directed edges) between them. The pair (V,E) denotes the unweighted[1] directed graph over these web pages. A hyper-link from page $i \in V$ to $j \in V$ is a pair $(i,j) \in E$. Please note that we remove all self-referential links from this graph, i.e. $\forall i \in V : (i,i) \notin E$.

The adjacency matrix of this graph is obtained as usual, based on an arbitrary but fixed ordering of the set V.

11.2.2 Nomenclature

We use the following naming conventions in order to minimise misunderstandings:

Data type	Form	Example
Matrices	Italic upper case letters	A
Vectors and parameters	Greek lower case letters	τ
Constants	Italic lower case letters	m

Let A be a square matrix. The transpose of A is denoted A'. Its dominant eigenvalue is denoted by $\lambda_1(A)$ and the second largest eigenvalue by $\lambda_2(A)$, etc. The corresponding eigenvectors are denoted by $\pi_1(A), \pi_2(A)$, etc. The spectrum of A is the set $\Lambda(A) = \{\lambda_1(A),\ \lambda_2(A),\ \ldots\}$.

DEFINITION 11.1 (RATING) A *rating function* is a total function $\rho : V \to [0,1]$. For each individual page $i \in V$, the value $\rho(i)$ is called the *rating* of i.

As output, all algorithms in this paper yield rating functions.

DEFINITION 11.2 (RANKING) A *ranking* is a partial order of pages according to a rating function, where pages with rating values larger than a certain cut-off value ε are ordered in decreasing order with respect to their rating.

[1] This is a simplification used in this paper, our algorithms works with weighted graphs as well, e.g. from multiplicity of links.

Note that a ranking does not need to contain all elements in V, as only pages with large enough rating are included. We have used $\varepsilon = 10^{-6}$ when nothing else is said in this paper.

DEFINITION 11.3 (RANKING ORDER) A *ranking order* assigns to each page i the index σ_i of that page within a ranking, where $\sigma_i = 1$ for the highest ranked page. If the range is restricted, then the ranking order is adapted accordingly, e.g. removing page b from $\{\sigma_a = 3, \sigma_b = 2, \sigma_c = 1\}$ yields $\{\sigma_a = 2, \sigma_c = 1\}$.

Ties because of equal ratings have been handled by using the fixed ordering among the elements of the set V (see Section 11.2.1 on the preceding page).

11.2.3 Metrics

We use four different measurements when comparing the algorithms:

DEFINITION 11.4 (n-VALUE) Given $n = |\Theta|$ pages that are initially assumed to be relevant for a topic, the *n-value* is the percentage of these appearing in the n top elements in the output ranking.

An n-value corresponds to both a recall and a precision value, and is usually called *R-Prec* when using exact rather than assumed number of relevant pages [4].

DEFINITION 11.5 (TOTAL-VALUE) The *total-value* is the percentage of the pages in the given input set that received a value larger than ε after running an algorithm.

DEFINITION 11.6 (SFD) The *Spearman Footrule Distance* [29, 30, 11] shows how different rankings are. This value is between 0 (identical ranking orders) and 1 (inverted or heavily permuted ranking orders) with 0.5 meaning totally random orders with no correlation. It is defined as follows: Given two ranking orders σ and τ with s elements in common,

$$\mathsf{SFD}(\sigma,\tau) = \frac{2}{s^2}\sum_{i=1}^{s}|\sigma_i - \tau_i|.$$

DEFINITION 11.7 (ORDER %) Given two rankings, the *order %* is the probability that two pages that are consecutive in the first ranking will have the same relative order in the other, considering only pages that are common to both rankings.

11.2.4 Small Test Dataset

The first database used in our experiments consists of a subset of the pages available at: `http://www.cs.umu.se/`, the web site of the Department of Computing Science, Umeå University. This dataset was collected in January 2003.

The database contains 7,312 pages, of which 2,728 are HTML pages with outgoing links. A total of 4,486 of the pages contain text. There are a total of 22,970 hyper-links, yielding an average of approximately 8.42 outgoing hyper-links per HTML page and just over 3.14 incoming links per page. The corresponding unweighted adjacency matrix has a dominant eigenvalue of $\lambda_1 \approx 25.813242$.

A total of 57,823 distinct stemmed words are present in the entire set of pages. These words are present in on average 9.87 pages, for a total of 570,870 page occurrences. Among these words, 33,854 appear in the same set of pages as another word. Thus, only 23,969 unique rankings per algorithm are required for the entire dataset.

11.2.5 Large Test Dataset

The dataset used in the assessment consists of all web pages found within the Nordic countries, i.e. Denmark, Finland, Iceland, Norway, and Sweden, by using a web spider. It was collected in January and February 2007. This dataset was chosen since these countries have had access to the Internet for a long time, they contain a mix of old and new, academic and commercial, web servers, and provide fast access from our location.

This database contains 3,087,531 web addresses, of which 478,985 contain hyper-links that point to another server. It contains 37,245,054 hyper-links, of which 3,889,216 are non-local. The corresponding unweighted non-local adjacency matrix has a dominant eigenvalue of $\lambda_1 \approx 49.135476$.

All in all, 8,054,200 stemmed words of at least four characters appear 221,259,520 times in 727,757 of the pages, leading to an average of just over 304 unique words per page in that set.

11.2.6 Performance Details

All running times reported throughout this paper have been measured while running the algorithms on a 1.7 GHz Pentium M laptop with 1.5 GB RAM. While this is a quite modest machine, given the newest machines available in the market (especially stationary machines with 64 bit processors), it will still give a rough estimate of the time required to run the algorithms.

11.3 Related Works

Web link mining is the discovery of useful knowledge from the structure of hyper-links. This link structure has been exploited in several techniques to say something about the importance of different web pages, such as in HITS [16], CLEVER [6], PageRank and Google [3], cGraph [17], the Intelligent Surfer [26], as well as [25] and [22]. The strength of using links in this way is that neighbouring web pages (when using hyper-links to define distance) can be used to either deduce or corroborate information about a web page.

Almost all of these techniques use a (modified) connection matrix corresponding to the graph that they are working on. Each algorithm yields one or more eigenvector(s) corresponding to the eigenvalue(s) of the connection matrix. Such eigenvectors can be seen as a tuple of values that define a rating for the corresponding pages.

The algorithms, as well as their underlying methodologies differ considerably between various approaches within this domain. Most of the link mining systems available today are based on either PageRank or HITS. Both of these families of algorithms use a connection matrix, but in quite different ways.

The PageRank algorithms first performs normalisation on the matrix in order to get a zero-sum propagation of data when multiplying with a random vector whose values form a Markov chain [3]. This matrix has to be updated further, since cross referrals would accumulate more and more of the value in each iteration. The way this is handled in PageRank is to simulate a jump to a random page with a certain probability, also known as the *damping factor*. This also fulfils the requirement on a Markov chain, that each page must have both incoming and outgoing links. The final answer appears when the changes between two iterations of the algorithm have become small enough, and corresponds intuitively to the probability that a random user would look at the corresponding web page. This is called the *PageRank*. We will also use a version of PageRank whose random jumps only lead to a starting set of pages, called Topic-sensitive PageRank [13]. Using $1-\mu$ as the damping factor and n is the number of pages, we can define PageRank as:

$$PR(j) = \frac{1-\mu}{n} + (\mu) \times \sum_{(i,j)\in E} PR(i)/\text{out-degree}(i) \tag{11.1}$$

The HITS algorithms, on the other hand, yield not one value, but two; Authority – indicating how interesting the information on the page relative to the given query is, and Hub – indicating how good the pointers from the page are [16]. These numbers are obtained by multiplying the connection matrix with the Hub values – yielding the new set of Authority values – followed by a multiplication of the transpose of the connection matrix with the Authority values – resulting in a new set of Hub values – and finally a normalisation of both Authority and Hub values so that they remain within a reasonable interval. We will not compare our algorithms with the HITS algorithms in this paper, since our algorithms are more similar to the PageRank family.

For more information on ranking systems see [18].

11.4 Propagation of Topic-relevance

The basic idea of the algorithms proposed in this paper is that each page is assigned a relevance value for each topic. This relevance value propagates along hyperlinks, while decreasing for each link travelled. This decrease is controlled using a parameter ξ, called the *decay factor*. Therefore, the method is called *Propagation of Topic-relevance* (ProT). The method is quite similar to *spreading activation* [24, 7, 8, 1], but uses multiplicative rather than additive activation in each iteration.

To implement this idea, we use an iterative approach similar to the algorithms described in Section 11.3. Starting with an initial assignment of relevance values, iterated updates and normalisations are made until a fixed point is reached (or, more precisely, until the changes are smaller than a certain threshold that we call *cut-off*). Let ϖ_j^k be the relevance value for page $j \in V$ in iteration $k \geqslant 0$. The initial values depend on the set of pages initially assumed to be on-topic, i.e. the set Θ:

$$\varpi_j^0 = \begin{cases} 1 & \text{if } j \in \Theta \\ 0 & \text{otherwise.} \end{cases} \tag{11.2}$$

The ProT algorithm is then given by normalisation of

$$\varpi_j^k = \left(\frac{1}{\xi} \sum_{(i,j) \in E} \varpi_i^{k-1} \right) + \begin{cases} \varpi_j^{k-1} & \text{if } j \in \Theta \\ 0 & \text{otherwise.} \end{cases} \tag{11.3}$$

This is the same thing as using the wide-spread power method [12] on a matrix $\hat{A}$, consisting of a standard adjacency matrix A divided by ξ with the addition that diagonal elements corresponding to pages in the set Θ are set to 1 (and using the values in the diagonal as the starting vector). Each iteration computes the next answer vector ϖ^1, ϖ^2, etc. The final result is given after a suitably large number of iterations (i.e. $\lim_{k \to \infty} \varpi^k = \pi_1(\hat{A})$), but the algorithm usually converges quite quickly as long as normalisation is done after each iteration (see Section 11.4.1 and the more advanced version of the algorithm given in Section 11.4.4).

NOTATION 11.1 Let D_Θ denote the diagonal matrix where the only non-zero positions are the diagonal elements corresponding to a member in Θ; these elements are set to one.

OBSERVATION 11.1 (EIGENVALUES OF A DIAGONAL MATRIX) Given that $|\Theta| = k$, then

$$\Lambda(D_\Theta) = \{\lambda_1(D_\Theta) = \ldots = \lambda_k(D_\Theta) = 1, \lambda_{k+1}(D_\Theta) = \ldots = \lambda_{|V|}(D_\Theta) = 0\}. \tag{11.4}$$

11.4.1 Resulting Values

Using Eq. (11.3) without normalisation leads to values ϖ_i^k much larger than 1, as soon as,

- there exists a path from one on-topic page to another, and/or,
- the sum of predecessor values for a page *i* is larger than ξ.

Even though we are interested in relative (rather than absolute) values, we need to keep these numbers in check, since they will accumulate over each iteration performed. These values can otherwise become arbitrarily large until they no longer can be represented in floating point format (see also HITS [16, 6]). The easiest way to handle this is by using normalisation after applying Eq. (11.3) once to all pages (or one iteration of the power method, as the case might be). Normalisation will, moreover, make it easier to test if changes compared with previous results are below the cut-off.

This leads to a rather interesting question, namely which normalisation to use. Since we are using relevance values and we want to consider at least one page as relevant, it seems natural to assign full relevance (i.e. 100%) to the most relevant pages. This indicates that a normalisation using $||\cdot||_\infty = \max(\cdot)$ should be used, leading to Algorithm 11.1.

ALGORITHM 11.1 (PROT)

Parameters: A, ξ, D_Θ *(as defined previously)*
$\hat{A} = \frac{A}{\xi} + D_\Theta$
$\forall i \in V : \varpi_i^0 = \hat{A}_{(i,i)}$
for $k = 1, 2, \ldots$
  $\varpi^k = \hat{A}\varpi^{k-1}$
  $\varpi^k = \varpi^k / ||\varpi^k||_\infty$
  if $||\varpi^k - \varpi^{k-1}|| < \varepsilon$, ***stop***
end for

This algorithm will converge, as long as the two largest eigenvalues of $\hat{A}$ are not equal [12].

11.4.2 Expected Results

The intuitive view is that the results given by the algorithm should correspond to two authoritative sources: Θ and the links of the web. If an appropriate set Θ was chosen, all pages that belong to Θ should appear fairly early in the resulting rankings. Moreover, pages that are pointed to from many of the pages in Θ should be given a high relevance value as well. If ξ is increased, this should intuitively strengthen the relative effect of '$+D_\Theta$'.

There is always the trade-off between these mutually conflicting expectations. As will be seen below, we can tune the tendencies towards the different expectations by setting the decay factor ξ. The larger ξ, the closer the results go towards pages in Θ, and vice versa.

11.4.3 The Size of ξ

There is a rather complex relationship between the parameter ξ and the behaviour of ProT, as we will show. In general:

- If ξ is too small (e.g. zero) then the primary eigenvector of the original matrix is given, regardless of Θ. This often leads to the situation where none of the pages in Θ are given as a result for a search term.
- Increasing ξ strengthens certain eigenvectors of $\hat{A}$. While this is a desired effect, it may lead to very slow convergence if ξ gets too large.

Let us explain this in more detail. While the eigenvalues of a matrix are given by the matrix, their behaviours are complex when the matrix is changed. Increasing multiple values in the diagonal might result in an increase of multiple eigenvalues, thereby inhibiting the convergence of ProT (see Conjecture 11.1 on page 181).

Values added to diagonal positions in the adjacency matrix that do not belong to the eigenvectors of the non-zero eigenvalues results in the creation of an eigenvector with this element as its only member. The corresponding eigenvalue is equal to the value added in the diagonal, see Theorem 11.1 on page 181.

All other diagonal additions result in the increase of at least one eigenvalue, and possibly shifting (and resizing) of others so that all eigenvalues will continue to be linearly independent of each other.

11.4.3.1 Using Too Small ξ

Let B be the sum of D_Θ multiplied by ξ and the adjacency matrix A:

$$B = \xi D_\Theta + A. \tag{11.5}$$

B is after normalisation identical in all respects (except the absolute size of the eigenvalues) to the matrix $\hat{A}$ used elsewhere in this paper, except at $\xi = 0$ or $\xi = \infty$.

It is obvious that $\lambda_1(B) = \lambda_1(A)$ when $\xi = 0$. What happens when ξ is increased is more complex. It depends on both the layout of the web and the number as well as the position of elements in Θ.

However, if one or more of the pages in Θ coincide with the non-zero elements of the dominant eigenvector $\pi_1(B)$, enlarging ξ will strengthen the corresponding eigenvalue (while shifting the eigenvectors slightly toward these pages). Unless another eigenvector gets an even stronger boost from multiple pages in Θ, this will continue to be the strongest eigenvector.

On the other hand, if some other eigenvector is strengthened to the point where its eigenvalue is equal to the (possibly increased) dominant eigenvalue of A, they will compete for position as the strongest eigenvalue. We call the ξ where this happen the *peak value*, since it corresponds to a distinct local maximum in the number of iterations required to reach a stable value (see Figure 11.3(b) in Section 11.5.1 for typical examples from the test database).

The exact value required to reach the peak value is somewhere in $(0, \lambda_1(A)]$, depending on A and Θ. We are currently not able to predict exactly where the peak values are.

A typical behaviour of ProT when given too small values for ξ can be seen in Figure 11.1 on the following page. When ξ is smaller than the peak value, the dominant eigenvector of the connection matrix is given. The strengthened eigenvalue competes with the dominant eigenvalue of the original connection matrix, and then takes over ($15.2 \leqslant \xi \leqslant 19.1$). Multiple eigenvalues are increased at the same time, resulting in slower convergence, as can be seen in right half of Figure 11.1 on the next page when ξ is increased above 19.1.

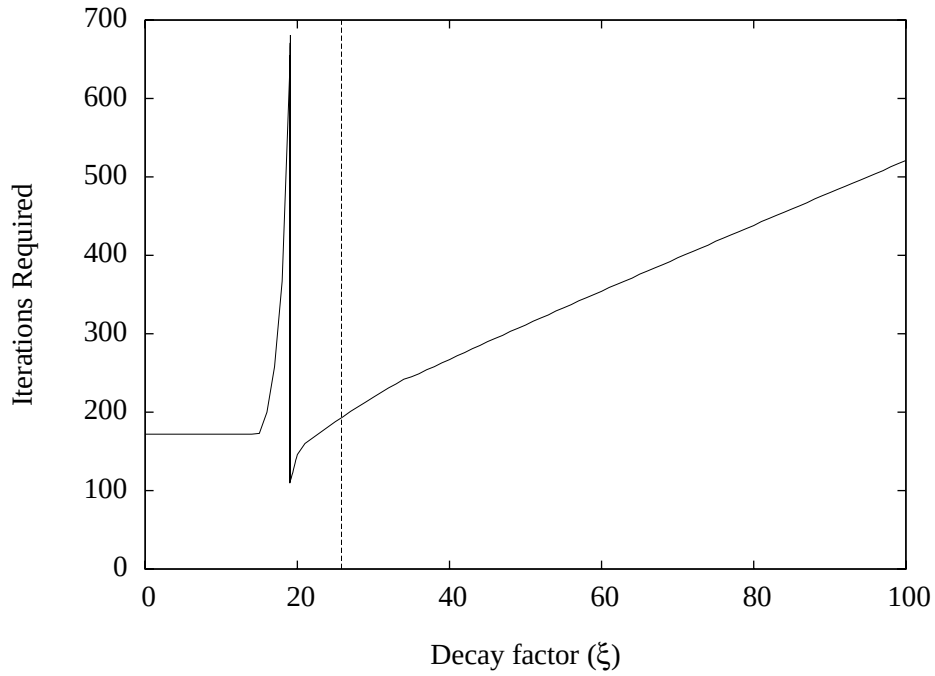


Figure 11.1: This figure shows the relationship between decay factor and minimum number of iterations required to reach a stable result for ProT. The vertical line corresponds to the dominant eigenvector of the adjacency matrix.

11.4.3.2 Using Too Large ξ

The situation is quite different when ξ is large and $|\Theta| > 1$. Multiple eigenvectors will be affected by the values in the diagonal, indicating that it will be very hard to find the correct eigenvectors using the power method. This leads us to Observation 11.2.

OBSERVATION 11.2 (TOO LARGE ξ) Using a very large ξ in ProT will prohibit convergence of the power method.

Let the matrix $\hat{A}_{\xi,\Theta}$ be the sum of the adjacency matrix A divided by ξ and D_Θ. This gives, per definition, that

$$\lim_{\xi\to\infty} \hat{A}_{\xi,\Theta} = \lim_{\xi\to\infty} \frac{1}{\xi} A + D_\Theta = D_\Theta. \tag{11.6}$$

This means that the $|\Theta|$ largest eigenvalues in the spectrum of $\hat{A}_{\infty,\Theta}$ are equal to 1. The convergence rate of the power method is linear to $|\lambda_2/\lambda_1|$ [12] and here we have $\left|\frac{\lambda_2(\hat{A}_{\infty,\Theta})}{\lambda_1(\hat{A}_{\infty,\Theta})}\right| = 1$ leading to no convergence at all.

One observation that can be made here is that this is not true when $|\Theta| = 1$, since there is only one non-zero eigenvalue when $\xi \to \infty$. This leads us to the version of our algorithm as described in Section 11.4.4.

11.4.3.3 Selection of ξ

A moderate solution is to use $\xi = \lfloor \lambda_1(A) + 1 \rfloor$. This is large enough to be over the peak value, since the peak value appears in $(0, \lambda_1(A)]$. This value is still not so high that slow convergence is a problem for smaller web sites. Examples of this can be seen in Figure 11.1 on the preceding page.

11.4.4 Superpositioned Singleton Propagation of Topic-relevance (S^{2}ProT)

A further development of the ProT algorithm using the same general idea but a slightly different approach is the S^{2}ProT algorithm. Instead of trying to generate the entire eigenvector at once, it creates one vector for each page in Θ and then performs additive superpositioning of these followed by normalisation, thus resulting in the vector that yields the returned rating. The rationale for this is that even though many different calculations need to be performed, this is offset by much faster convergence for each subproblem and reuse of vectors whenever a page is on-topic for more than one topic.

The reason for the fast propagation is that each such vector calculation can be viewed as a propagation with decreasing strength, i.e. a topological ordering with minor changes because of back links.

DEFINITION 11.8 (SINGLETON MATRIX) Let $S_{(i)}$ be the $V \times V$ matrix with a single non-zero value equal to 1 in the diagonal. This corresponds to a self-reference of the page $i \in V$. Such a matrix is called a *singleton matrix* and satisfies,

$$\Lambda(S_{(i)}) = \{1, 0, \ldots, 0\}.$$

ALGORITHM 11.2 (S^{2}PROT)

Parameters: A, ξ, Θ *(as defined previously)*
$\forall i \in \Theta$ *:* $rating_i$ *= ProT(*$A, \xi, S_{(i)}$*)*
$rating = \sum_{i \in \Theta} rating_i$
return $rating/||rating||_\infty$

Note that, when computing ratings for sets $\Theta_1, \ldots, \Theta_k$, we only need to compute $rating_i$ once, for each $i \in \cup_{1 \leqslant j \leqslant k} \Theta_j$. In practise, this will be very useful as it allows one to consider a large collection of topics. Moreover, S^{2}ProT has excellent convergence rate when $\xi > \lambda_1$, proportional to ξ/λ_1, and will finish in $\frac{\log(\varepsilon)}{\log(\lambda_1(A)) - \log(\xi)}$ iterations or less (see Theorems 11.3 on page 182 and 11.4 on page 183, respectively).

This means that each individual page rating $rating_i$ can be calculated in just a few iterations. PageRank (using $\mu = 0.85$[2]) requires approximately 114 iterations to yield an eigenvector with a maximum error of less than 10^{-8}, while each S^{2}ProT calculation (using $\xi = 2 \times \lambda_1(A)$) will get the same margin of error in at most 26 iterations. Using an even larger ξ will yield stable results even faster, as can be seen in Theorem 11.4 and Figure 11.4 on page 164 (e.g. $\xi = 4 \times \lambda_1(A) \rightsquigarrow 14$ iterations and $\xi = 10 \times \lambda_1(A) \rightsquigarrow 8$ iterations).

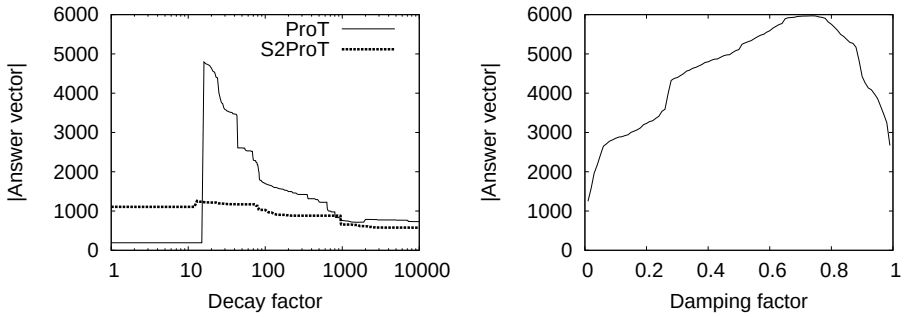
Almost as important is the fact that we do not need to concern ourselves with pages further away from a starting node than the maximum number of iterations required. This means that we can speed up S^{2}ProT very effectively by increasing ξ, leading to calculations on a small subset of the original web with only a few iterations required. The downside of using large decay factors is that fewer pages not in Θ will be included in the answer, and they will be given later in the ranking order.

[2]The value of μ recommended by the PageRank authors, where μ corresponds to the non-dampened part of the PageRank calculation (which plays a somewhat similar role as our ξ).

11.5 Comparison of Algorithm Behaviours

In this section we describe the general behaviour of our algorithms when applied to our small test database, using the well-known Topic-sensitive PageRank as a basis of comparison. We look at the number of non-zero elements in the rating functions, scalability, execution times, and similarities between our algorithms and Topic-sensitive PageRank.

Our algorithms have very different behaviours regarding how they are affected by the input data size (handled below) and their different decay factors (Figure 11.2). The number of non-zero elements in the rating functions is important when assessing relevance; the precision (see Eq. (11.7) in Section 11.6) goes down drastically when too many pages are given in the results.



(a) ProT and S^2ProT (b) Topic-sensitive PageRank

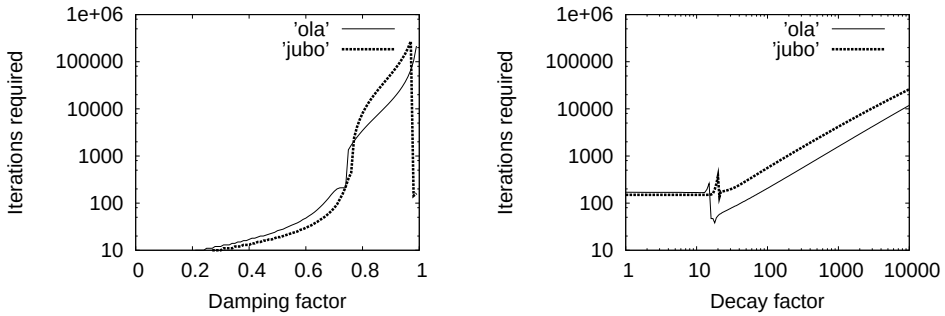
Figure 11.2: Typical relationships between input parameters (ξ and μ, respectively) and number of non-zero elements in the resulting vectors using actual data. Topic-sensitive PageRank added for comparison.

ProT The basic ProT algorithm yields reasonable results, but has a number of disadvantages when it comes to efficiency. For a large web, the number of iterations required to reach a stable state is typically in the same range as for Topic-sensitive PageRank. Thus, ProT does not scale well. Choosing an appropriate decay factor is imperative; too small and the result is misleading at best, and too large and no answer will be given in due time (if ever). The number of non-zero elements in the resulting vectors depends on the decay factor, as can be seen in Figure 11.2(a) on the facing page. The rather complex behaviour of the basic algorithm with respect to the decay factor can be seen in Figure 11.3(b) on the next page.

S^{2}ProT This version scales well, and yields reasonable results as long as the decay factor is high enough, i.e. larger than the dominant eigenvalue of the adjacency matrix. The larger the decay factor, the smaller the resulting set and the faster the convergence, as can be seen in Figure 11.2(a) on the facing page and Figure 11.3(c) on the next page.

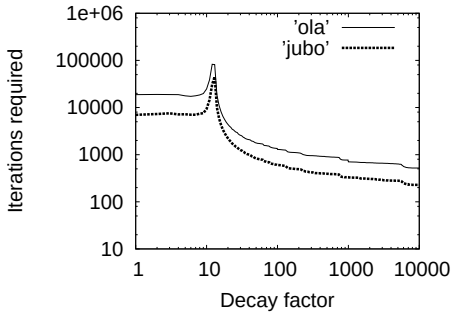
Topic-sensitive PageRank We implemented Topic-sensitive PageRank [13] and tested it on the same data as ProT and S^{2}ProT. It turns out that this algorithm has roughly the same scalability as ProT. The number of non-zero elements in the resulting vector is very large when using the recommended values of μ (0.7–0.85). Moreover, it requires a lot of iterations to reach a stable state, as can be seen in Figure 11.3(a).

Applying the approach of Jeh and Widom [15, 14] would require fewer iterations, but still much more than for S^{2}ProT. The main problem with their approach is that the choice of hub nodes to use in the calculation is critical.



(a) Topic-sensitive PageRank

(b) ProT



(c) S^2ProT

Figure 11.3: Typical relationships between input parameters (μ and ξ, respectively) and total number of iterations required to reach a stable state for each algorithm. One thing to note here is that, even though the total number of iterations required for the superpositioned algorithm might seem a bit high, the work done pays off, because the individual page ratings (denoted $rating_i$ in Section 11.4.4 on page 158) can be reused for other queries (see the discussion in Section 11.4.4 on page 158). A total of 251 individual page ratings are created for the search term 'ola' and 110 for 'jubo' by S^2ProT.

11.5.1 Execution Time

Using the small test database, we found the execution time of the various algorithms (Table 11.1 and Figure 11.3 on the preceding page). Each individual word available within any of these pages constitutes its own query, e.g. the word "ladok" exists on 23 pages and these 23 pages belong to the Θ of the query for "ladok". The cut-off for termination was set to $\varepsilon = 10^{-6}$ for all algorithms. Table 11.1 shows the total number of iterations and the total CPU time required to compute all queries.

Table 11.1: Actual number of iterations and CPU time required for our small database per algorithm.

Algorithm	*Iterations*	*CPU time*
Topic-Sensitive PageRank	115 051 531	7.5 days
ProT	557 646 907	10 days
S^{2}ProT	87 825	158 secs

11.5.1.1 Topic-Sensitive PageRank

A total of 115 M iterations were required for an average of just under 4,700 iterations per search term using $\mu = 0.85$. The maximum required number of iterations for a single query is just over 4 M. Calculating the answer set for all queries amounts to approximately 7.5 d on the test computer with only minimal optimisation done on the code. A typical behaviour of this algorithm with respect to the damping factor can be seen in Figure 11.3(a) on the preceding page.

11.5.1.2 ProT

A total of 557 M iterations were required for an average of approximately 23,200 iterations per search term with $\xi = 26$. The maximum required number of iterations for a single query is just over 7 M. Calculating the answer set for all queries amounts to 10 d on the test computer with only minimal optimisation done on the code. The behaviour of the basic algorithm depends heavily on the decay factor (including the unstable peaks at 15-20), as can be seen in Figure 11.3(b) on the facing page.

11.5.1.3 S^{2}ProT

A total of almost 88 k iterations were used to create the complete set of individual page ratings for the 4,486 pages with text when the decay factor was set to 26. This amounts to an average of 19.6 iterations per page, with a maximum of 1586

iterations for one page. This corresponds to a total of 1.52 iterations/search term, since already computed page ratings can be reused for other queries, as discussed earlier.

Calculating the answer vectors for all questions takes a total of 158 s of user time on the test computer, including superpositioning and all overheads and with no optimisation made.

Figure 11.3(c) on page 162 shows the typical behaviour pattern for this algorithm with respect to the decay factor. Please note the peak values when the decay factor is between 10 and 20, i.e. just under λ_1.

Figure 11.4 shows that the number of iterations is well below the upper bound as given in Section 11.4.4 on page 158, specifically Theorem 11.4 on page 183.

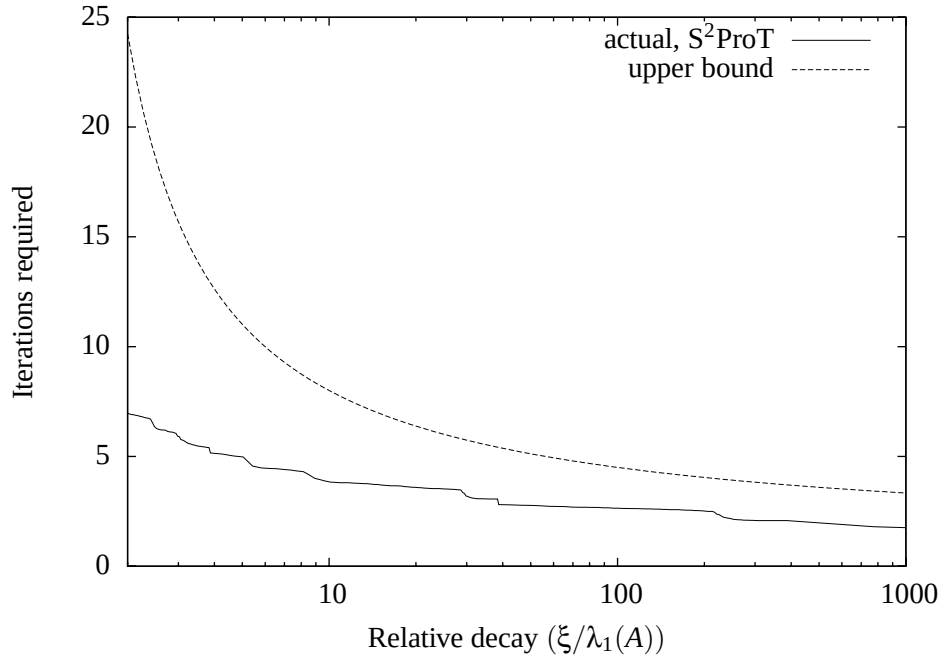


Figure 11.4: The relationship between $\xi/\lambda_1(A)$ and the number of iterations required to reach a stable state, looking at both practical values as well as upper bound (as given in Theorem 11.4 on page 183) using a cut-off of 10^{-6}.

11.5.2 Summary

In this section, we compared the execution behaviour of ProT and S^{2}ProT to that of Topic-sensitive PageRank. Specifically:

- In the beginning of Section 11.5, we showed that the number of non-zero elements in the resulting vector depends on μ (for Topic-sensitive PageRank) or ξ (for ProT/S^{2}ProT). We also discussed the scalability of each algorithm in terms of CPU usage, with S^{2}ProT being the most scalable of the three.
- In Section 11.5.1 we showed that the ProT algorithm is roughly comparable to Topic-sensitive PageRank when it comes to execution speed, while S^{2}ProT is *much* faster. The practical rate of convergence for S^{2}ProT was also shown to be well below the theoretical upper bound for our test database.

This means that S^{2}ProT will yield results in a timely fashion, making it suitable even for very large web systems. This is especially true for query systems with many search terms, since page ratings can be reused for all terms for which this page occurs as a starting page, i.e. is in Θ.

11.6 Empirical Results

In the previous section we evaluated the general behaviour of our algorithms compared to Topic-sensitive PageRank for a small (but real) example database. In this section we evaluate the quality of the resulting ratings and the stability of our algorithms compared to PageRank and Topic-sensitive PageRank, based on perceived relevance and stability.

This is the second study of the ProT algorithm. The first study was done on a single web server and was reported in a resent paper [31], while the one reported here used all web pages our web spider was able to find in the Nordic countries (see Section 11.2.4 on page 149 and Section 11.2.5 on page 149, respectively).

11.6.1 Assessment

The relevance assessment was done on the large dataset (see Section 11.2.5 on page 149). In particular, the web considered in this assessment did not consist of pages mainly taken from the academic environment as in [31], but covered commercial, private, and public service sites as well. The three algorithms that were considered in this assessment were PageRank, Topic-sensitive PageRank and S^2ProT.

11.6.1.1 Relevance

One of the most important factors of a search engine is how relevant the resulting sets of pages are, especially the pages given early. The two measurements typically used to judge web search and Information Retrieval systems are called *precision* and *recall*, as defined in equations 11.7 and 11.8. These cannot always be computed, since the set of relevant items is not always given in such a system. High precision indicates that most of the items retrieved are relevant, while high recall indicates that most of the available relevant records in the database have been retrieved.

$$\text{Precision} = \frac{|\text{Retrieved Relevant Items}|}{|\text{Retrieved Items}|} \tag{11.7}$$

$$\text{Recall} = \frac{|\text{Retrieved Relevant Items}|}{|\text{Relevant Items}|} \tag{11.8}$$

In the assessment we will instead use perceived relevance, i.e. how relevant and appropriate each page was for the given search term, as judged by a group of people making individual assessments.

11.6.1.2 Choice of Search Terms

The search terms were chosen according to the following selection criteria:

1. At least 20 pages had to contain the search term;
2. The tenth result given by each algorithm had to be above the base level ($\frac{1-\mu}{n}$ for PageRank, etc.);
3. At least three different rating values had to exist in each top ten list;
4. The search terms should be spread over a wide range of topics; and
5. There should be no risk that the search terms could be considered offensive by the participants.

The reason behind the first criterion was to get enough variability. The next two criteria ensured that the rankings from the algorithms would determine the result, rather than the alphabetical order. The last two criteria ensured that the assessment would be as fair and free of bias as possible.

There was a very large set of possible search terms in this experiment, so the final choice came down to whether a search term was well defined and had a clear cut-off around the tenth or eleventh element in the top lists of Topic-sensitive PageRank and S^{2}ProT. The latter was used to assure that the alphabetical order that we used as arbitration between pages with equal ranking would not affect the result one way or the other. This resulted in a list of 85 search terms. The complete list is given in Appendix 11.B on page 184.

11.6.1.3 Setup of the Assessment

The basic setup of the assessment was that participants assessed the subjective relevance of each given page with respect to each search term. These pages were given in alphabetical order without any marking as to what algorithm or which algorithms yielded the page in the top list. There were five different grades among which the test subjects were asked to choose the most appropriate one for each page:

- The first grade indicated that the reviewer was unable to say anything about the relevance of the page regarding the current keyword. These assessments were ignored in all calculations.
- The second grade corresponded to a complete lack of relevance.
- The third grade was indicative of some relevance.
- The fourth grade indicated a moderate amount of relevance.
- The fifth and final grade indicated that the page was very relevant, corresponding to 100% relevance.

The second, third, fourth, and fifth grades were assigned the numerical values 0, 0.5, 0.8, and 1, respectively. This scaling was used to avoid an underestimation of the true relevance of the pages, since we were using terms for the grades that intuitively should have had a higher relevance, e.g. “relevant” for the fourth grade (was $\frac{2}{3}$ in [31] compared to 0.8 in this paper). The relevance numbers were then disseminated back to each algorithm by averaging the grades that were given by the graders. This leads to a system where the subjective perceived relevance values can be computed for the results from each algorithm.

11.6.1.4 Results

A total of 587 valid assessments (i.e. with at least one grading being from grades two to four) resulted in between 577 and 581 answer sets per algorithm or approximately 7.6 answer sets per search term.[3] The participants showed a preference for the results of S^{2}ProT over Topic-sensitive PageRank and for Topic-sensitive PageRank over PageRank, as can be seen in Figure 11.5 and Figure 11.6 on the following page.

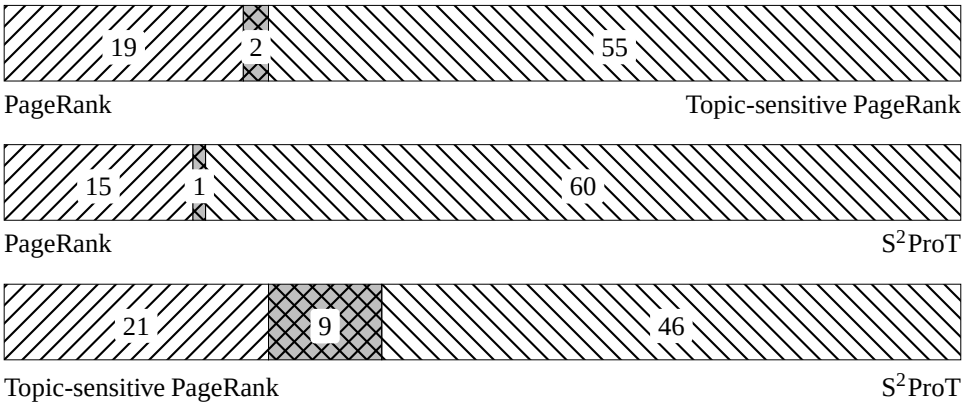


Figure 11.5: Preference of relevance per search term using pairwise algorithm comparisons. Thus, e.g. for 19 search terms, the graders preferred the rankings returned by PageRank over the one returned by Topic-sensitive PageRank. For 55 search terms, Topic-sensitive PageRank was preferred over PageRank, and the remaining two search terms resulted in a tie.

The resulting values were then compared using ANOVA tests. The results showed that the S^{2}ProT algorithm gives better relevance results than Topic-sensitive PageRank with statistical significance ($p < 0.05$), that in turn had higher relevance than PageRank with very high statistical significance ($p < 0.001$). The average values for the entire test as well as the 95% confidence intervals can be seen in Figure 11.6 on the next page, and the ANOVA answers are given in Table 11.2 on the following page.

[3] 76 of the search terms had at least one assessment.

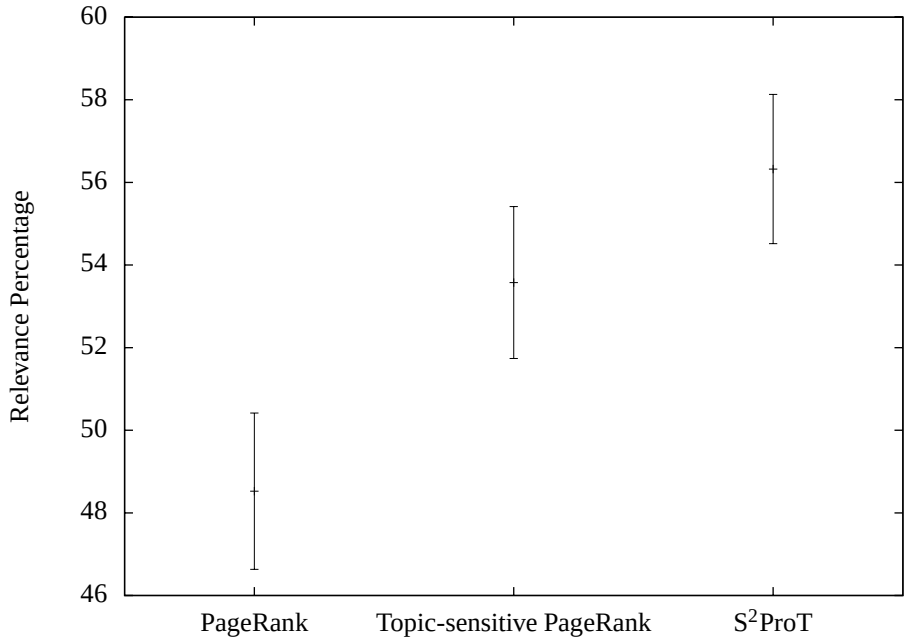


Figure 11.6: The average 95 percent confidence intervals for each algorithm for the assessment.

Table 11.2: ANOVA test results from the assessment.

Compared results	*F-value*	*p-value*
All three	F(2,1733) = 17.6475	p = 2.5870×10^{-8}
PR - TsPR	F(1,1153) = 14.0671	p = 0.00018516
PR - S^2ProT	F(1,1157) = 34.1410	p = 6.6571×10^{-9}
TsPR - S^2ProT	F(1,1156) = 4.3672	p = 0.036855

11.6.2 Stability

Another really important aspect of a web search engine is that the results should not change too much even if small changes are applied to the underlying web. The users are expecting valid results even if some information is not available to the search engine when generating answer vectors. Since all of the algorithms that we talk about in this paper work on graphs and most of them use a set Θ of starting pages, we can find two different ways in which we can disturb the system in order to check its stability:

1. removal of pages from Θ (thus affecting the starting sets per topic), and
2. removal of links (affecting the graph).

We will show empirical results from both types of perturbations in this section.

11.6.2.1 Missing Pages in Θ

Using our small test data set, it was possible to remove a number of pages and still get a valid number of remaining pages (between 20% and 30% of the original Θ removed) for 16,681 of the 57,823 words.

By running our algorithms as well as Topic-sensitive PageRank on the diminished dataset, we found some interesting measurements when we compared the results of the different algorithms. In each case we compared the results from using the diminished dataset with the original input dataset, given in *n*-value and total-value (see Section 11.2.3 on page 148).

The results from applying each algorithm to the diminished data sets can be seen in Table 11.3 and (for S^{2}ProT) Figure 11.7 on the next page.

Table 11.3: Recall values for various algorithms using a diminished data set as basis.

Algorithm	*n-value*	*total-value*
Topic-sensitive PageRank	53.95	91.96
ProT	37.81	93.42
S^{2}ProT (using $\xi = 30$)	77.90	91.00

Both the *n*-value and the total-value are quite high for Topic-sensitive PageRank. The total-value in particular indicates that the results are relevant even when some pages have not yet been indexed.

ProT does not have as good an *n*-value as Topic-sensitive PageRank. It does, however, have a better total-value even though it on average returns only 594.5 pages per search term while Topic-sensitive PageRank returns on average 2024.53 pages (a factor of over 3.4 times as many pages for Topic-sensitive PageRank).

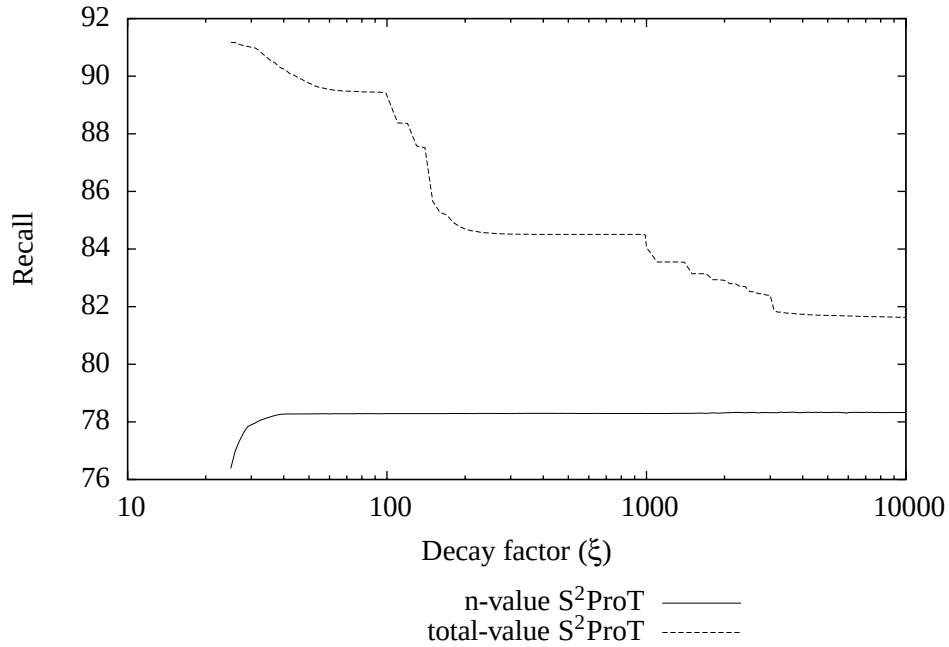


Figure 11.7: Recall values for the top *n* results and in all values given from the S^2ProT algorithm using the diminished sets with regard to the original Θ sets. S^2ProT values are not valid unless the decay factor is larger than the dominant eigenvalue of the matrix, in this case ≈ 25.813242.

The precision (and recall) over the first *n* values of S^2ProT are much higher than for the other algorithms. The total-value recall is only slightly lower than Topic-sensitive PageRank and ProT. It does, however, depend heavily on the chosen decay factor, as is clearly visible in Figure 11.7. Using a larger ξ means faster results but fewer pages in the resulting sets.

11.6.2.2 Link Removal

One of the major concerns for some of the other approaches to searching has been their partial lack of stability if links are missing [21, 20]. Our approach is very stable: the resulting ranking order is almost identical even when 10% of the links have been dropped at random. In test databases with between 5000 and 15000 web pages, we have found an average SFD of less than 0.1 when 10% of the links were dropped. That means that our algorithms are in the same order of stability as Topic-sensitive PageRank as far as missing links are concerned. Table 11.4 shows the results from applying each algorithm to the small test database, and the stability is shown when just over one in ten (10.2%) of the links have been removed randomly.

It is unfortunately not that easy to compare the results from these algorithms, since their stability varies considerably depending on how it is measured. The Spearman footrule distance indicates that ProT is the most stable algorithm, followed by Topic-sensitive PageRank, and, finally, S^{2}ProT. Order % on the other hand gives a different picture, according to which S^{2}ProT is the most stable algorithm, followed by ProT, and, finally, Topic-sensitive PageRank.

Table 11.4: Stability of the algorithms when removing 10.2% of the links on the small test dataset, showing the Spearman footrule distance and order % (including their variance), as well as the average number of pages given per search term.

Algorithm	avg SFD	σ^2	order %	σ^2	avg #pages
ProT	0.05831	0.01387	89.833	0.03035	568.5530
S^{2}ProT	0.08865	0.00647	94.523	0.01079	914.1436
Topic-sensitive PageRank	0.08279	0.01307	84.295	0.03655	1448.3610

We have tested the link removal stability of each algorithm for 10 of our queries in the two test databases. The total scatter plots as well as least squares best fit correlations for 320 different link removals are shown in Figure 11.8 and Figure 11.9 on the next page, respectively. Even though all of the algorithms are somewhat sensitive to link removals, both Topic-sensitive PageRank and S^2ProT give average ranking orders, in the form of Spearman Footrule Distance, lower than the removal rate for these datasets. S^2ProT increases in stability as the number of pages in Θ are increased.

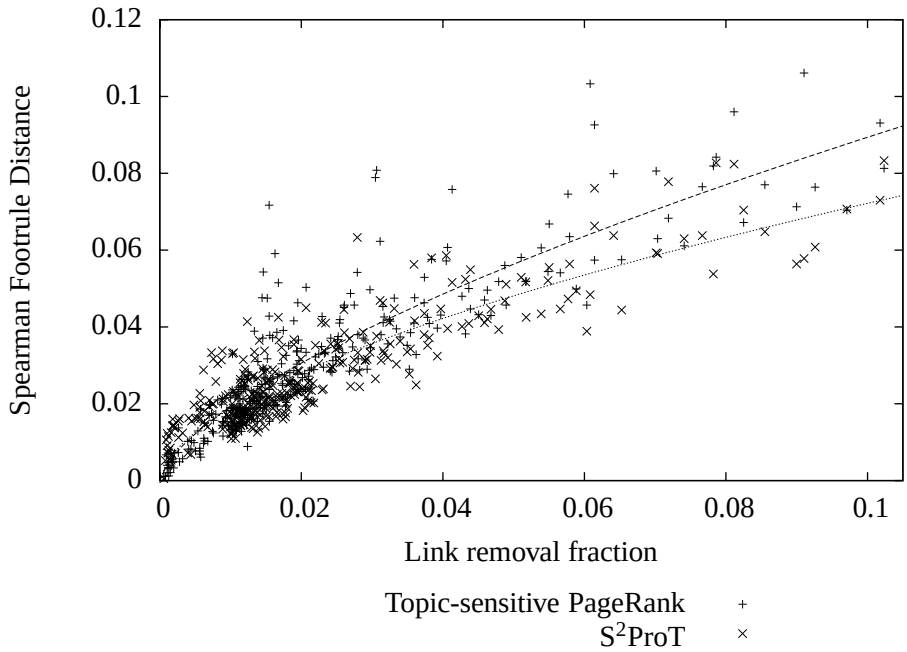


Figure 11.8: Stability of some of the mentioned algorithms on the smaller test database when removing a certain amount of links, given as average Spearman Footrule Distance from ten search terms with various removals as well as least-squares best fit of $a \times x^b$ for each algorithm.

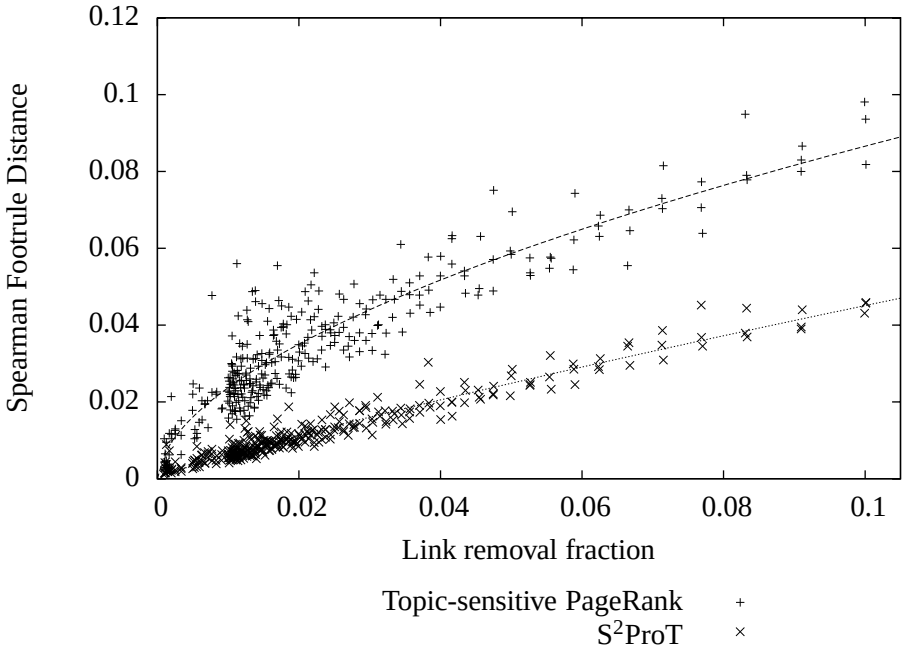


Figure 11.9: Stability of some of the mentioned algorithms on the larger test database when removing a certain amount of links, given as average Spearman Footrule Distance from ten search terms with various removals as well as least-squares best fit of $a \times x^b$ for each algorithm.

11.6.3 Summary

In this section, we compared the perceived relevance and stability using actual queries on actual datasets. Our results show that our algorithms compare well with algorithms in use today. Specifically:

- In Section 11.6.1, we showed that the results given by S^2ProT are good on a typical larger dataset, by comparing it to PageRank and Topic-sensitive PageRank.
- The stability of our algorithms was studied in Section 11.6.2, specifically:
  - In Section 11.6.2.1, we showed that our algorithms are stable when Θ is diminished. This indicates that our algorithms can be used even if some pages have not yet been indexed or have been incorrectly indexed.
  - In Section 11.6.2.2, we showed that our algorithms give more or less the same result even if some of the links are removed from the datasets. This indicates that our algorithms are useful even if some of the links are missing from the database or some of the pages have not yet been traversed.

This means that our algorithms will yield good results even when applied to datasets containing small errors and omissions. As can be expected, the more errors in the input, the less good the results will be.

11.7 Discussion

11.7.1 Efficient Implementation

Using a value of one in the starting vector at the positions corresponding to the elements of Θ ensures that the resulting vector will always be non-zero, as this value is always non-zero in the dominant eigenvector. Moreover, it is for S^{2}ProT usually very close to the direction of the dominant eigenvector, thus leading to even faster convergence.

Another way of speeding up the calculation of ProT or S^{2}ProT is to keep a list of currently reached pages $R \subset V$, i.e. $i \in R$ in iteration k indicates that $\varpi_i^k > 0$. This means that only the important pages are used in the calculations, reducing the number of elements to calculate to a bare minimum. The downside of doing this is that a stride of larger than 1 will be used when going through the database, thus increasing the likelihood of cache misses and possibly even page faults for large databases. This can be diminished by using a storage order that as much as possible is derived from the link structure. This is, however, not trivial to implement since most webs have a lot of circular structures and back links.

Using S^{2}ProT on a parallel computer or in a distributed environment is straightforward. The individual page ratings can be calculated at the same time, since they do not depend on each other. The precalculated topic vectors can also be stored efficiently, as was shown in [26].

11.7.2 Hybrid S^{2}ProT

Another version that we have tried is what was called the *Hybrid Superpositioned Singleton Propagation of Topic-relevance* (HyS^{2}ProT) in [31]. It uses singletons as in S^{2}ProT (see Section 11.4.4), but each outgoing value is further decreased with the number of outgoing links (i.e. the out-degree of a page) in the same manner as in PageRank (hence the name). It turns out to be slightly more stable than S^{2}ProT when missing links are concerned and works with a matrix having a fixed dominant eigenvalue of less than or equal to one, but is otherwise inferior to S^{2}ProT.

11.7.3 Future Work

We are currently developing a complete search system based on our algorithms. The system is supposed to handle everything from the initial retrieval of the data to the actual searching, using a user-friendly web interface.

A number of open questions remain:

- The exact decay value required to overcome the dominant eigenvector of the original matrix when using ProT depends on both the underlying matrix and the set Θ used (how many pages as well as their position in the dataset). Further study regarding the most suitable choice of ξ would be interesting.
- The results and stability of ProT are promising, but using power iteration is often too CPU intensive on even moderately large webs to be practically usable. Better methods could be explored, such as Arnoldi iteration [28]. This problem does not exist when using S^2ProT, as can be seen in Section 11.4.4.
- Our algorithms should be compared to some new development using spatial link structures, such as [5].
- It is possible to have the same relevance value for many pages in the resulting set when running these algorithms, thus resulting only in a partial order among the pages. Some possible ways to discriminate among these pages could be to use weighing according to the term frequency [27] or the corresponding eigenvalue when carrying out the superpositioning. This was not a major issue in our datasets, so this has not been pursued further at this time.

11.7.4 Conclusions

We have shown that topic-specific answer sets can be generated very quickly using S^2ProT (Section 11.4.4 and Section 11.5) and that the results are both relevant and stable (Section 11.6). This indicates that our algorithms are very useful for generating relevance values and rankings for web pages in a topic-sensitive manner.

ACKNOWLEDGMENTS

We thank the referees for their valuable comments that have led to numerous improvements on the structure and content of the paper. We also thank all the participants of the assessment.

11.8 References

[1] ASWATH, D., AHMED, S. T., D'CUNHA, J., AND DAVULCU, H. Boosting Item Keyword Search with Spreading Activation. In *Web Intelligence* (2005), A. Skowron, R. Agrawal, M. Luck, T. Yamaguchi, P. Morizet-Mahoudeaux, J. Liu, and N. Zhong, Eds., IEEE Computer Society, pp. 704–707.

[2] BERMAN, A., AND PLEMMONS, R. J. *Nonnegative Matrices in the Mathematical Sciences*. SIAM, Philadelphia, PA, 1994.

[3] BRIN, S., AND PAGE, L. The Anatomy of a Large-scale Hypertextual Web Search Engine. *Computer Networks and ISDN Systems 30*, 1–7 (1998), 107–117.

[4] BUCKLEY, C., AND VOORHEES, E. M. Evaluating evaluation measure stability. In *SIGIR '00: Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval* (New York, NY, USA, 2000), ACM, pp. 33–40.

[5] CHAKRABARTI, S. Dynamic personalized pagerank in entity-relation graphs. In *WWW '07: Proceedings of the 16th international conference on World Wide Web* (New York, NY, USA, 2007), ACM, pp. 571–580.

[6] CHAKRABARTI, S., DOM, B. E., AND INDYK, P. Enhanced hypertext categorization using hyperlinks. In *Proceedings of SIGMOD-98, ACM International Conference on Management of Data* (Seattle, US, 1998), L. M. Haas and A. Tiwary, Eds., ACM Press, New York, US, pp. 307–318.

[7] CRESTANI, F. Application of Spreading Activation Techniques in Information Retrieval. *Artif. Intell. Rev. 11*, 6 (1997), 453–482.

[8] CRESTANI, F., AND LEE, P. L. WebSCSA: Web Search by Constrained Spreading Activation. In *IEEE Forum on Research and Technology Advances in Digital Libraries (ADL '99)* (1999), pp. 163–170.

[9] CVETKOVIĆ, D., ROWLINSON, P., AND SIMIĆ, S. *Eigenspaces of Graphs*. Cambridge University Press, 1997.

[10] DAVISON, B. D. Topical locality in the web. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval* (Athens, Greece, 2000), ACM Press, pp. 272–279.

[11] DWORK, C., KUMAR, R., NAOR, M., AND SIVAKUMAR, D. Rank aggregation methods for the web. In *WWW '01: Proceedings of the 10th international conference on World Wide Web* (New York, NY, USA, 2001), ACM Press, pp. 613–622.

[12] GOLUB, G. H., AND VAN LOAN, C. F. *Matrix computations (3rd ed.)*. Johns Hopkins University Press, 1996.

[13] HAVELIWALA, T. H. Topic-sensitive PageRank. In *Proceedings of the eleventh international conference on World Wide Web* (2002), ACM Press, pp. 517–526.

[14] JEH, G., AND WIDOM, J. Scaling personalized web search. Tech. Rep. 2002-12, Stanford University Database Group, 2002.

[15] JEH, G., AND WIDOM, J. Scaling Personalized Web Search. In *WWW '03: Proceedings of the 12th international conference on World Wide Web* (New York, NY, USA, 2003), ACM Press, pp. 271–279.

[16] KLEINBERG, J. M. Authoritative sources in a hyperlinked environment. *J. ACM 46*, 5 (1999), 604–632.

[17] KUBICA, J., MOORE, A., COHN, D., AND SCHNEIDER, J. cGraph: A fast graph-based method for link analysis and queries. In *Proceedings of the 2003 IJCAI Text-Mining & Link-Analysis Workshop* (Acapulco, Mexico, Aug. 9, 2003).

[18] LANGVILLE, A. N., AND MEYER, C. D. A Survey of Eigenvector Methods of Web Information Retrieval. *The SIAM Review 47*, 1 (2005).

[19] MINC, H. *Nonnegative matrices*. John Wiley and Sons, New York, 1988.

[20] NG, A. Y., ZHENG, A. X., AND JORDAN, M. Link analysis, eigenvectors, and stability. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)* (2001).

[21] NG, A. Y., ZHENG, A. X., AND JORDAN, M. Stable algorithms for link analysis. In *Proceedings of the Twenty-fourth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Sept. 2001).

[22] NIE, L., DAVISON, B. D., AND QI, X. Topical link analysis for Web search. In *SIGIR* (2006), E. N. Efthimiadis, S. T. Dumais, D. Hawking, and K. Järvelin, Eds., ACM, pp. 91–98.

[23] PAGE, L., BRIN, S., MOTWANI, R., AND WINOGRAD, T. The PageRank citation ranking: Bringing order to the web. Tech. rep., Stanford Digital Library Technologies Project, 1998.

[24] PIROLLI, P., PITKOW, J. E., AND RAO, R. Silk from a Sow's Ear: Extracting Usable Structures from the Web. In *CHI* (1996), pp. 118–125.

[25] RAFIEI, D., AND MENDELZON, A. O. What is this page known for? computing Web page reputations. *Computer Networks 33*, 1-6 (2000), 823–835.

[26] RICHARDSON, M., AND DOMINGOS, P. The Intelligent Surfer: Probabilistic Combination of Link and Content Information in PageRank. In *NIPS* (2001), T. G. Dietterich, S. Becker, and Z. Ghahramani, Eds., MIT Press, pp. 1441–1448.

[27] SALTON, G., AND YANG, C. S. On the specification of term values in automatic indexing. *Journal of Documentation 29*, 4 (Dec. 1973), 351–372.

[28] SCOTT, J. A. An Arnoldi code for computing selected eigenvalues of sparse, real, unsymmetric matrices. *ACM Trans. Math. Softw. 21*, 4 (1995), 432–475.

[29] SPEARMAN, C. The proof and measurement of association between two things. *American Journal of Psychology 15* (Jan. 1904), 72–101.

[30] SPEARMAN, C. 'Footrule' for measuring correlations. *British Journal of Psychology 2* (June 1906), 89–108.

[31] ÅGREN, O. Assessment of WWW-Based Ranking Systems for Smaller Web Sites. *INFOCOMP Journal of Computer Science 5*, 2 (June 2006), 45–55.

11.A Theorems and Proofs

DEFINITION 11.9 (PROPER EIGENVECTOR) An eigenvector corresponding to a non-zero eigenvalue is called a proper eigenvector.

THEOREM 11.1 (ADDING A NEW EIGENVALUE TO THE SPECTRA OF A MATRIX) Adding a value to a diagonal element that does not belong to a proper eigenvector of an adjacency matrix with zero diagonal results in the creation of a new eigenvector that contains at least that element. The corresponding eigenvalue will be as large as the value added to the diagonal.

PROOF. That an element a belongs to a cycle within the matrix (i.e. there is a nontrivial path from a back to a) or on a direct path from such a cycle implies that a belongs to a proper eigenvector, and vice versa [2, Chapter 2, Theorem 3.20].

An element a that does not belong to a proper eigenvector has zero value for all nonzero eigenvectors of the matrix. Adding the diagonal element creates a new local cycle that contains at least a.

None of the other eigenvalues are affected and the sum of the eigenvalues is equal to the trace of the matrix, indicating that the eigenvalue corresponding to the newly added eigenvector is equal to the value added in the diagonal. □

THEOREM 11.2 (INCREASING ONE EIGENVALUE OF A MATRIX) Adding a value to a diagonal element that belongs to a proper eigenvector of an adjacency matrix with zero diagonal results in the increase of the nonnegative eigenvalue of that eigenvector of at least the added value.

PROOF. There is no connection between components of a graph in the corresponding matrix and updates of the diagonal of the matrix do not change this, so we can consider only this component without loss of generality.

The result follows from the fact that only one eigenvalue can be increased by the addition and that $\sum_{i \in n} \lambda_i(A) = \mathrm{tr}(A)$. The corresponding eigenvector is nonnegative [19, Chapter 1, Theorem 4.2], and this is the eigenvector/eigenvalue pair found when using the power method, if the starting vector is nonnegative and nonzero. □

CONJECTURE 11.1 (INCREASING MULTIPLE EIGENVECTORS) Adding multiple values in the diagonal of a nonnegative matrix A increases the eigenvalue of the corresponding component (or nonnegative eigenvector of that principal submatrix [9, Theorem 2.1.5]) that each element belongs to.

Worst case scenario is that multiple eigenvalues are increased to the same size. This yields a spectrum with dominating real eigenvalues of multiplicity higher than

one, with corresponding nonnegative eigenvectors. The result of this is that the power method will not converge at all. This situation is fortunately not that common, but should not be ignored when using ProT.

THEOREM 11.3 (RATE OF CONVERGENCE OF S^{2}PROT) The rate of convergence of S^{2}ProT depends on λ_1/ξ (as long as $\xi > \lambda_1$). The smaller this number the faster the convergence.

PROOF. Let $\overline{A}$ be the adjacency matrix A (with zeroes in the diagonal) divided by $\xi > \lambda_1$. This matrix has a spectrum of

$$\Lambda(\overline{A}) = \frac{1}{\xi}\Lambda(A) = \{\lambda_1(A)/\xi, \lambda_2(A)/\xi, \ldots\}.$$

Let $S_{(i)}$ be a singleton matrix with a spectrum of $\Lambda(S_{(i)}) = \{1, 0, \ldots, 0\}$.

The composition of these two matrices results in the matrix $\hat{A}_{(i)} = \overline{A} + S_{(i)}$, which is formed by ProT for each call from the definition of S^{2}ProT. There are two distinct possibilities for this matrix:

1. If the page i is not found in any of the proper eigenvectors of A (and thus $\overline{A}$), then $\Lambda(\hat{A}_{(i)}) = \{1, \lambda_1(A)/\xi, \ldots\}$ as per Theorem 11.1 on the preceding page.

2. In all other cases $\lambda_1(\hat{A}_{(i)}) > 1$ and $\lambda_2(\hat{A}_{(i)}) \leqslant \lambda_1(A)/\xi$, since the addition of $S_{(i)}$ will increase the strength of one of the eigenvalues of $\overline{A}$ that it coincides with (see Theorem 11.2 on the previous page). All other eigenvalues of this component might be adjusted (and possibly diminished) to still be linearly independent of this new dominant eigenvector.

Recall that the rate of convergence of the power method depends on the relative size of the two largest eigenvalues, i.e. the smaller $|\lambda_2/\lambda_1|$, the faster the convergence [12]. For $\hat{A}_{(i)}$ this corresponds to at most $\frac{\lambda_1(A)/\xi}{1} = \lambda_1(A)/\xi$. This factor corresponds to the relative diminishing effect of applying each power iteration on all eigenvalues except $\lambda_1(\hat{A}_{(i)})$. Thus: the larger ξ is, the faster the convergence. □

THEOREM 11.4 (MAXIMUM NUMBER OF ITERATIONS REQUIRED FOR S²PROT) Given a page $i \in V$, a starting vector that is not orthogonal to the dominant eigenvector of the matrix $\pi_1(\hat{A})$ (formed by the adjacency matrix A using $\hat{A} = A/\xi + S_{(i)}$), and a maximum cut-off of ε, ProT($\hat{A}$) yields a stable result within

$$I_{max} = \frac{\log(\varepsilon)}{\log(\lambda_1(A)) - \log(\xi)} \text{ iterations.}$$

PROOF. The relative strength of an eigenvector $\pi_i(\hat{A})$ (where $i > 1$) will be decreased with a factor of $\lambda_i(\hat{A})/\lambda_1(\hat{A})$ relative to the strength of $\pi_1(\hat{A})$ in each iteration of the power method. After k iterations this means that the relative strength of $\pi_i(\hat{A})$ is

$$s = \left| \frac{\lambda_i(\hat{A})}{\lambda_1(\hat{A})} \right|^k \tag{11.9}$$

We are interested in the number of iterations required to let the relative strength of $\pi_2(\hat{A})$ drops below the cut-off (ε), and this happens when

$$\varepsilon = \left| \frac{\lambda_2(\hat{A})}{\lambda_1(\hat{A})} \right|^{I_{max}} \tag{11.10}$$

We have already established that $\frac{\lambda_2(\hat{A})}{\lambda_1(\hat{A})} \leqslant \lambda_1(A)/\xi$, and this together with Eq. (11.10) yields

$$\varepsilon = (\lambda_1(A)/\xi)^{I_{max}} \tag{11.11}$$

$$\log(\varepsilon) = I_{max}(\log(\lambda_1(A)) - \log(\xi)) \tag{11.12}$$

$$I_{max} = \frac{\log(\varepsilon)}{\log(\lambda_1(A)) - \log(\xi)} \tag{11.13}$$

Q.E.D. □

11.B Search Terms for the Assessment

Search terms that received no assessments are given in *italic*.

Danish: *Boghandler*, musikvidenskab, rigsarkivet, *uddannelse*, and undervisningsministeriet.

English: Helicopter, infection, jubo[a], railway & station, swegrid, table & tennis, and trout & fish.

Finnish: Aineisto, elämä, ihminen, jakelu, korkeakoulu, osaaminen, tietoa, tietokone, toimitusjohtaja, and vastuu.

Icelandic: Áhersla, bókasafn, *bókmenntir*, greinar, *háskóla*, heilbrigði, *kennara*, *landafræði*, *pappír*, *smiði*, stærðfræði, and *tölva*.

Norwegian: Akershus, datamaskin, fartsprøve, forbrukerombudet, karrieresenter, kringkastingssjef, and nasjonalbiblioteket.

Swedish: Ågren, ansökningshandlingar, arkitekturmuseet, arkivcentrum, centrumbildningar, civilingenjör, datatermgruppen, datavetenskap, doktorandhandbok, energimyndigheten, fakultetsnämnden, forskningsdatabas, forskningsområden, genusforskning, geovetenskap, grundskolor, hemvärnet, humlab, idéskolor, idrottshögskolan, informationsteknik, källkritik, kammarkollegiet, lammstek, styrmekanism, kommunikationsteknik, länsbiblioteket, lärarutbildning, marklära, matematikdidaktik, miljöanalys, minoritetsspråk, naturvetenskaplig, överprövning, punktskriftsböcker, rymdfysik, sökmotoroptimering, and språkverkstaden.

Nordish: Billedkunst, Kaspersen, kulturstudier, science & fiction, Sturlasson, and universitet.

[a] This is the nick name of Jürgen Börstler, an associate professor and director of studies at the Department of Computing Science at Umeå University, Sweden.

A

Appendix A

Users' Guide to CHIC

Abstract

This appendix outlines how to use programs in the CHIC *system, as well as input, output and data base format used. For information about the algorithms used by* CHIC*, see [3, Paper II] and [5, Paper III]. It uses sets of natural numbers in the same manner as described in [4].*

Keywords: CHIC, data base, inverted index, format

A.1 Introduction

The Concept HIerarchy Constructor (CHIC) is a system for automatic generation of concept hierarchies from large discrete databases. The system contains two programs: One that takes an input file of a specific format and creates a database, and one that uses the information in the database to create the hierarchies. These programs are called `words` and `chic`.

A.2 Using the `words` Program

The program for data import is called `words`. This program does not use any command line parameters. It reads each input line from standard input and adds it to the database.

A.2.1 Input Format

Each line of the input data corresponds to one record in the database. There are three different fields on each line:

Record name — a number of characters, may not contain the separator.

Separator — marks the end of the record and start of the list of keywords. Default value for `words` is ':'. Changes must be done in the file called `words.l` before recompilation of the program.

List of keywords — Each keyword contains a number of characters or numbers, but never a space.

A typical example of such input data is given in Figure A.1 on the facing page.

A.2.2 Creating and Populating a Data Base

Make sure that the current working directory is where the new data base should have its root, using `cd` to get to the right place in the file system. Create the data base directory by executing the command

```
mkdir db
```

If the information in Figure A.1 on the next page was stored in the file called "example.in" it could be added to the database using the command

```
words < example.in
```

The result should be a database that matches what is described in Section A.4.

```
Makefile:   make commands text
clean:      Bourne shell script text
combinator: ELF 32-bit MSB executable, SPARC, version 1, dynamically linked, not stripped
db:         directory
fest.txt:   International language text
fil.aux:    LaTeX auxiliary file
fil.dot:    ASCII text
fil.eps:    PostScript document text conforming at level 2.0
fil.log:    TeX transcript text
fil.tex:    LaTeX 2e document text
input:      English text
lex.yy.c:   C program text
main.c:     C program text
main.o:     ELF 32-bit MSB relocatable, SPARC, version 1, not stripped
words.l:    lex description text
words.o:    ELF 32-bit MSB relocatable, SPARC, version 1, not stripped
```

Figure A.1: A short example of discrete meta-data, in this case output of the "file" command.

A.2.3 Known Bugs

The program can unfortunately not add more information to an existing database, it must currently start from a clean state. This will probably be fixed in a later version.

A.3 Using the `chic` Program

CHiC is a terminal based program that implements all algorithms given in [5]. It reads a database with the layout and format given in Section A.4.

A.3.1 Usage

`chic [-bdfltx] [-c <cut off depth>] [-C <cut off>] [-o <filename>]`

The command line parameters have the following meaning:

- **-b** — block order, pick the keyword with as the highest number of neighbours as possible among the vectors.
- **-d** — turn on debug output. Warning, can be very verbose.
- **-f** — filling, try to reuse vectors from earlier dimensions if possible.
- **-l** — generate a lattice, rather than a normal strict hierarchy.
- **-t** — timed execution of each step, mostly for debugging and testing purposes.
- **-x** — generate dot format output, see Section A.3.2 for a description of the format.
- **-c** <**cut off depth**> — cut everything beneath a certain depth from the hierarchy/lattice.
- **-C** <**cut off**> — remove all keywords that appears in less than <cut off> % of the records in the database.
- **-o** <**filename**> — redirect the output to the file given as argument.

The source code for CHiC is written in such a way that it should be easy to add routines for other databases and output formats.

A.3.2 Dot Format (as Generated by CHiC)

The standard output format of CHiC is the directed graph input format used by `dot` and `dotty` [2, 1].

The generated output file will contain some extraneous code lines as soon as the vectors cannot fit in one dimension. These lines are very easy to remove, depending on whether information is required for the entire graph as a whole or each dimension on its own.

A.3.2.1 Basics

The top level graph of the dot format is **digraph name** { **statement-list** }.

The important statements include:

n_0 [**$name_0$=val_0,$name_1$=val_1, ...**]**;** Adds the node n_0 (unless already created) and sets the its attributes according to the optional list.

n_0 -> **n_1** -> **...** -> **n_n** [**$name_0$=val_0,$name_1$=val_1,...**] ; Creates edges between nodes n_0, n_1, ... n_n and sets their attributes according to the optional list. Creates nodes as necessary.

label=text sets the label of a node to the text field given, where text may include escaped newlines \n, \l, or \r for centre, left, and right justified lines. This is the only attribute generated by CHiC.

Comments are in either **/*C-style*/** or in **//C++-style**.

For more information about `dot` and `dotty` input format see either [2, 1] or the manual page for `dot`.

A.3.2.2 CHiC Dot Output Specifics

The number used for each node is the decimal representation of the corresponding keyword within the database, see Section A.4.

The extraneous lines generated by CHiC contain specific comments to simplify usage of the output file.

If a complete graph with all dimension in one picture is wanted, then remove all lines in the output file that contain the comment `/* INTERNAL */`.

If a specific dimension *nn* is required, then remove all lines before the comment line containing `/* Start Dimension`*nn*` */` and after `/* Stop Dimension`*nn*` */`.

Both options yield valid `dot` input files.

A.4 Contents of Each Data Base File

The database is stored in a directory called `db`. This directory contains a number of files when the data has been added to the database. The examples given in the text below are when the input as given in Figure A.1 on page 189 has been processed.

DB This file contains the keywords in the order of their first appearance in the input file, one per line. The first keyword is the one enumerated with 1024, the second 1025, etc. See Figure A.2 on the next page for an example.

names The names of each record in the input file is stored in this file, one line per record. See Figure A.3 on the facing page.

stdin This file contains the relationships of the database, one line of enumerated numbers per record. Between each hexadecimal number of the line is a space character, and the numbers are in the order of the keywords as given. See Figure A.4 on page 194.

400 upwards Each file corresponds to the inverted indices for the keyword with that enumerated number, i.e., those given in DB. Some of the inverted indices can be seen in Figure A.5 on page 194.

The information in these files can be accumulated into a adjacency matrix, i.e. Table A.1 on page 195. This is however not feasible when the data base size increases, since the size of the table is number of records times number of keywords. Test data base three described in Table 8.1 on page 97 in Paper III [5] would need a table with 1,841,378,616 squares in it, but very sparsely filled in.

A.5 An Example Data Base

We will be using the same example as the one found in [3, 5], i.e. Figure A.1 on page 189.

A.5.1 Creating a Data Base

Given a database created by following the steps in Section A.2.2 we will find that each keyword in the input file is assigned an enumerated value starting with 1024 (hexadecimal 400). The reason for starting the enumeration at 1024 was to retain space for single characters and important words.

A.5.2 Content of the Data Base

The contents of each file in the data base will be:

DB As in Figure A.2, but please remember that it will in reality be a file with 39 lines with one word per line.

names As in Figure A.3, with one record name per line.

stdin Exactly as in Figure A.4 on the following page.

400–426 Figure A.5 on the next page contains some of the inverted indices. It would take up too much space to add all of them to this document.

Table A.1 on page 195 contains a adjacency matrix for this database.

```
make            sparc           latex           tex
commands        version         auxiliary       transcript
text            1               file            2e
bourne          dynamically     ascii           english
shell           linked          postscript      c
script          not             document        program
elf             stripped        conforming      relocatable
32-bit          directory       at              lex
msb             international   level           description
executable      language        2.0
```

Figure A.2: Keyword file (db/DB).

```
Makefile        fest.txt        fil.log         main.c
clean           fil.aux         fil.tex         main.o
combinator      fil.dot         input           words.l
db              fil.eps         lex.yy.c        words.o
```

Figure A.3: Name file (db/names).

```
400 401 402
403 404 405 402
406 407 408 409 40a 40b 40c 40d 40e 40f 410
411
412 413 402
414 415 416
417 402
418 419 402 41a 41b 41c 41d
41e 41f 402
414 420 419 402
421 402
422 423 402
422 423 402
406 407 408 424 40a 40b 40c 40f 410
425 426 402
406 407 408 424 40a 40b 40c 40f 410
```

Figure A.4: Relation file (`db/stdin`).

```
1
2
5
7
8
9
a
b
c
d
f
```
(a) text (`db/402`)

```
1
```
(b) make (`db/400`)

```
3
e
10
```
(d) elf (`db/406`)

```
f
```
(f) lex (`db/425`)

```
3
e
10
```
(e) msb (`db/408`)

```
6
a
```
(g) latex (`db/414`)

(c) executable (`db/409`)

Figure A.5: Sample inverted indices.

Table A.1: Adjacency matrix between rows in database and keywords. Each row can be seen as the inverted file for that keyword and each column represents each file record in sorted order.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
make	×															
commands	×															
text	×	×			×		×	×	×	×	×	×	×		×	
bourne		×														
shell		×														
script		×														
elf			×											×		×
32-bit			×											×		×
msb			×											×		×
executable			×													
sparc			×											×		×
version			×											×		×
1			×											×		×
dynamically			×													
linked			×													
not			×											×		×
stripped			×											×		×
directory				×												
international					×											
language					×											
latex						×				×						
auxiliary						×										
file						×										
ascii							×									
postscript								×								
document								×		×						
conforming								×								
at								×								
level								×								
2.0								×								
tex									×							
transcript									×							
2e										×						
english											×					
c												×	×			
program												×	×			
relocatable														×		×
lex															×	
description															×	

A.6 References

[1] GANSNER, E. R., AND NORTH, S. C. An open graph visualization system and its applications to software engineering. *Software: Practice and Experience 30*, 11 (2000).

[2] NORTH, S. C., AND KOUTSOFIOS, E. Applications of Graph Visualization. In *Graphics Interface '94* (Banff, Alberta, Canada, May 1994), pp. 235–245.

[3] ÅGREN, O. Automatic Generation of Concept Hierarchies for a Discrete Data Mining System. In *Proceedings of the International Conference on Information and Knowledge Engineering (IKE '02)* (Las Vegas, Nevada, USA, June 24-27, 2002), pp. 287–293. Paper II on page 59.

[4] ÅGREN, O. BITSET: Implementing Sets of Natural Numbers Using Packed Bits. Tech. rep., Umeå University, Umeå, Sweden, Oct. 2002. UMINF 02.10, ISSN 0348-0542.

[5] ÅGREN, O. CHIC: A Fast Concept HIerarchy Constructor for Discrete or Mixed Mode Databases. In *Proceedings of the Fifteenth International Conference on Software Engineering and Knowledge Engineering (SEKE'03)* (San Francisco, California, USA, July 1-3, 2003), pp. 250–258. Paper III on page 77.

Index

Colophon

Typesetting of this thesis was done using the LaTeX2ε macro system by Leslie Lamport on top of the TeX formatting engine by Donald E. Knuth. Times New Roman is the main font face found in the thesis, with the rare exceptions of some characters in equations and small/rotated pictures (where Computer Modern and Helvetica have been used, respectively).

A total of 28 LaTeX macro packages were directly imported. In reality, this number increases to 43 style files, seven definition files and four configuration files because of internal transitivity. Despite all these macro files, I made 47 definitions, created three new and reconfigured three old environments, created 18 new commands and reconfigured 15 others. I did, however, have the help of such wonderful tools as **aspell**, BibTeX and **makeindex**.

The figures were created using **dot** from the *GraphViz* collection of tools (for directed graphs), **gnuplot** (for all plottable graphs), or **xfig**. I used a ton of small shell scripts written in Bourne, awk or Perl together with larger programs in either MATLAB/Octave or R for calculations and statistics. The software of ALGEXT, CHIC and PROT are, however, written in C. All of this text as well as all programs have been written using **vi(m)**, still the fastest and best text editor out there.

All of this was of course controlled by a large number of makefiles. In fact, **make** had to first run LaTeX, then BibTeX eight[1] times, then LaTeX, then **makeindex**, then LaTeX again, and, finally, **dvips** whenever I wanted a new PostScript version of the thesis. This file was then converted to PDF using **ps2pdf** when required.

One thing has not changed, while at the same time changing rapidly, over the years; all of the work except the final submission of Paper VI were done on a computer using some sort of UNIX or UNIX-clone. The specifics have changed considerably, since I've had the pleasure of using SGI Irix, Sun Solaris, and GNU Linux (e.g. Debian and Ubuntu). They have all worked for me, rather than against me.

[1] BibTeX runs once on each paper, once for the front matter and once for the Appendix.