Using Web Graph Structures to Personalize Search

by

Francisco Tanudjaja

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2001

Author .....
Department of Electrical Engineering and Computer Science
May 23, 2001

Certified by .....
Peter Szolovits
Professor, MIT Department of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Using Web Graph Structures to Personalize Search

by

Francisco Tanudjaja

Submitted to the Department of Electrical Engineering and Computer Science
on May 23, 2001, in partial fulfillment of the
requirements for the degree of
Master of Engineering

Abstract

Personalization is a topic of active research in the information retrieval community. There are many applications developed to personalize users' web experience. In addition, recent advances in graph based search techniques derived from Kleinberg's work [22, 23] has been impressive. This thesis looks at these two aspects from a single perspective. First, we analyze variants of the graph based search algorithm that adapt to user preference [7] and extend them. Second, we develop a persistent user profiling technique, utilizing a web ontology provided by the *Open Directory Project*. In achieving these goals, we implement a real-time personalized search engine and run sets of controlled experiments to analyze the performance of our system.

Thesis Supervisor: Peter Szolovits
Title: Professor, MIT Department of Electrical Engineering and Computer Science

Acknowledgments

I would like to thank Professor Szolovits for his support and guidance in foreseeing my thesis. I am forever indebted to Lik Mui, who is the guiding light in my research. I thank him for the infinite amount of time he spared to our discussions and meetings. His indefatigable spirit always churns with new ideas; his focus and dedication push my research into the direction it must be going. I owe much thanks to James Lu and Waikit Koh, who helped me iron out many theoretical issues. Finally, I thank my parents for supporting me through all these years.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

The flux of information into the World Wide Web has grown rapidly ever since its conception. Due to the proliferation of personal computing and rapid advances in telecommunications technology, the Internet is widely available in a distributed setting, making such an environment ideal to function as a general platform of information exchange. A classic topic of information retrieval research is to devise methods for allowing users to tap relevant data efficiently from this vast resource.

Search engines index large numbers of documents and let users query desired documents. However, most search engines are not tailored to meet individual user preferences. [36] noted that almost half of the documents returned by search engines are deemed irrelevant by their users. There are several aspects to the problem. First is the problem of synonyms and homonyms. Synonyms are two words that are spelled differently but have the same meaning. Homonyms are words that are spelled the same but have different meanings [e.g the verb quail and the noun quail]. Along this line are words that have different *senses*. 'Rose' can be interpreted in the context of 'flower' or as 'wine.' Without prior knowledge, there is no way for the search engine to predict user interest from simple text based queries. Secondly, search engines should be deterministic in that it should return the same set of documents to all users with the same query at a certain time. Therefore it is inherent that

search engines are not designed to adapt to personal preferences.

Current information retrieval research shows much interest in enhancing the user's web experience. These efforts branch in several directions. One direction is to create a better structural model of the web, such that it can interface more efficiently with search engines. A second approach is to model user behavior as to predict users' interests better.

Along the lines of the former are efforts at better defining the meaning of queries themselves. The *Wordnet* project at Princeton University is an online lexical reference system that organizes English words into synonym sets [13]. A similar approach is to build a taxonomy of words. A taxonomy comprises of a tree structure in which a word belongs to a certain node, each with parents and children. A node's parent serves a general category that encompasses all of its children. A node may have children that are sub-categories of itself. An example of such word taxonomies are the *Open Directory Project* [http://dmoz.org] and the *Magellan* hierarchy [http://magellan.excite.com]. Yet another approach is to create a semantic structure in machine readable format. As opposed to classifying content from a person's point of view, this method embeds meta data for classification, allowing document content to be machine readable. There are currently efforts at standardizing these classification, for example OIL (*Ontology Interchange Language*) and DAML (*DARPA Agent Markup Language*). Haystack [1] is an ongoing project in semantic meta data indexing.

Along the lines of the latter approach, various research in data mining and knowledge representation have build models to record user interest and predict user behavior. Ultimately, these user models interface with a system so as to give it *a priori* knowledge regarding user preferences.

Beyond search engines, there are currently many available systems that allow for personalization to some extent. Some systems allow users to specify page content [e.g *My Yahoo*], while others allow for filtering and finding news information from the Internet. Yet another type of system recommends web pages, books, music, etc. There are two common paths to match user interests. One is to compare against individual profiles. Another is to match against communities of profiles, also known as *collaborative filtering*. [37] contains a survey of many of the available systems and their methodologies.

Clearly, work in user profiling is closely related to building better personalized systems.

Different methods of gathering user data is often coupled with various personalization systems. We found that the combinations that are available in the context of personalized search are unsatisfactory. We propose a novel approach in building a better system.

1.2 Thesis contribution

This thesis seeks to accomplish two things. One is to extend existing theory with regards to personalized search. Second is to design a better user model. We note that current search methods lie under two main categories:

- *Content based indexing*

  Content based indexing is the traditional approach used in search engines. All documents are indexed in terms of their content, using vector space keyword extraction method such as *term frequency - in document frequency* (TF-IDF) - which basically analyzes the number of times a certain word occur in a document. *Inktomi* [http://inktomi.com] is an example of such a system.

- *Structure based indexing*

  Structure based indexing emphasizes on the link structure of documents. This method stems from the study of *bibliometrics*, a field which investigates citation structures of sets of documents. The idea is to create a web graph structure from such documents, exploiting information inherent in the links to predict the authority of documents [22]. *Google* [http://google.com] is an example of such a system.

Recent advances in graph based search technique derived from Kleinberg's work [22, 23] have been impressive. There are several variants of the algorithm that allow for biasing documents to take into account user preferences [7]. This thesis project extends further one of the variants and analyze its relative performance.

To support our argument, we have built an implementation of a personalized search engine. The system wraps a personalization module onto an existing search engine, and refines search results using the proposed extension of the graph based algorithm.

At its core, the proposed system utilizes a taxonomy of user interest and disinterest. We use a tree coloring method to represent user profiles. Visited nodes are 'colored' by the number of time it is visited, whether the user rate it as positive or negative, and URL's that it associates to.

In addition, we run sets of controlled experiments to analyze the performance of each of the existing variants. The experimental results verify our predictions and confirm that the proposed extension performs better.

The power of the system implementation lies in its modularity and open standards. Many of the pieces are reusable for various applications. Though in this particular implementation the system input comes from a search engine, with minor changes it is easily replaceable by some other data repository. One interesting application is to deploy it onto a *peer* to *peer* (P2P) network. A P2P network allows and facilitates collaboration among multiple users across different platforms (e.g ICQ, Napster, Groove). The P2P network can utilize the users' profiles to match similar interests. The taxonomy based profile lends additional power to the system in predicting common interests more intelligently.

In summary, this thesis offers theoretical improvement over existing methods, as well as a prototype that serve as a proof of concept. The prototype is a real time system that serves to complement our experimental results.

The following offers a road map to this document. In chapter 2, we review related works in user profiling, personalization, and graph based filtering. We describe existing approaches and available systems. We review the graph based techniques and explain the underlying mathematical intuition.

In chapter 3, we propose an extension to existing theory in graph based techniques. We present the theory along with in depth mathematical analysis and discussion. We provide the motivation and description of the tree coloring scheme used to model user profile.

In chapter 4, we lay out the design and implementation of the system. We describe the components of the system as well as sample use cases.

In chapter 5, we discuss methods of validation and the results of our experiments. We compare and contrast simulations using the original graph based techniques and its variants. We devise analysis metrics to evaluate simulation results.

We conclude in chapter 6 with a number of directions for future work.

Chapter 2

Literature Review

This chapter reviews relevant work in user modeling, personalization, and graph based search techniques. Often times user modeling is a subset of personalization, but we discuss the two separately in turn. The discussion on user modeling focus on areas of work that concentrate solely on finding methods to model user behavior. The discussion on personalization focus on areas of work that apply these models into products or systems that accomplish a specific task. Finally, the section on graph based search techniques reviews mathematical background and methods of analysis.

2.1 Related Work in User Profiling

The tasks of modeling user behavior lie beneath two major axes: gathering data, and interpreting data. Work in gathering data normally lies in using either explicit feedback or implicit feedback. Interpreting data has a broader spectrum, ranging from probabilistic learning models, neural network algorithms, genetic algorithms, vector space models, as well as various machine learning techniques. The representation models are usually tied to their respective interpretation models.

In explicit feedback models, the system requires direct user interaction. Users specify their preferences, and the system records these preferences in the user profile. A very simple and common example of explicit feedback is web browser bookmarks: users specify a resource

of interest so that the system can later redirect them to that same resource.

A majority of user modeling approaches lie in the implicit feedback domain. Such models observe user behavior, and try to infer user interest from usage history. A common way to implement implicit feedback is by analyzing web server logs, from which information such as pages visited, time spent on each page, and temporal properties such as the time the page was requested. Since many web servers generate *cookies*, user can be identified from one another while keeping anonymity. Examples of such systems are discussed in [17, 16, 6, 21, 20].

After gathering user data, the system translates it to a model that represents user profile. Some systems are not interested in analyzing individual users *per se*, but rather the statistical pattern of users visiting a certain web page. Many of these applications are of commercial interest. For example, [17] is interested in whether a user who visits a certain web page would be interested in buying certain products. The system uses a probabilistic relational model (PRM) to infer relationships between products or information. The technique derives from Bayesian networks analysis, with additional expressiveness. The PRM data is gathered using implicit feedback and the goal is to determine statistical correlation between properties of entities (e.g entity *user* and entity *product A*).

[32] describes a system that infers demographic attributes of users, such as age, sex, marital status, and so son with up to sixty percent statistical confidence. The interpretation process uses a vector space learning technique combined with a neural network algorithm. This system also uses implicit feedback for data gathering.

Another work discussed in [12] designs a user profiling model to detect fraud in the cellular phone industry. A user is normally associated with an ID and sometimes this ID is 'stolen' by malicious eavesdroppers. The system builds a profile from observing usage pattern. From this data, a Bayesian estimator is used to determine the probability of fraudulent use given the time and location of the call. If the probability is above a certain threshold, the central server can block 'suspicious' calls.

User profiling is also used in matchmaking systems such as Yenta [15]. Such systems assume a set of users and try to match similar interests based on users' profile. Yenta looks at a user's email messages and personal files, and tries to find *similar* users and 'introduce' them to one another. LARKS [41] is another example of a matchmaking system.

The next section discusses user profiling that directly benefits the user in terms of personalizing content.

2.2 Related Work in Personalization

Personalization encompasses many fields of research, such as information retrieval, software agents [39, 43], and machine learning. Personalization applications cover a range of spectrum. On one end we have recommendation systems, in which software agents try to find information of possible interest to the user, given a profile. These types of systems actively search for information. On the other end of the spectrum, we have filtering systems, which filter input from an information resource, marking those of possible interest for the user. These types of systems wait for information to come in, in contrast to the previous approach.

FAB [3] is an example of a recommendation system. It fetches sets of articles from the web, personalizing content for each user. Fab combines *content based* and *collaborative* recommendation methods, the former taking into account user's past interest, the latter taking into account tastes of *other* users who are *similar*. The system combines several agents to collect and filter the results. Users then rate the returned material via explicit feedback. The system learns by incorporating this feedback into a genetic algorithm, in which *unfit* retrieval agents die out. Other examples of recommendation systems are Letizia [35] and Syskill & Webert [35].

A classic example of a filtering system is GroupLens [38]. It rates articles in a Usenet newsgroup, using collaborative filtering. As before, the system uses the heuristic that a group of users who had had similar interests in the past are likely to have similar interests in the future. The system also uses explicit feedback - users are asked to rate articles that they have read, the results of which are passed onto a rating server, which in turn finds clusters of people with similar interests.

Another filtering system, SmartPush [27] combines several novel ideas together. The system finds information by means of semantic meta data to filter news articles. In addition, it builds the user profile using a simple hierarchical concept model. For example, under the category news, there are the categories sports, literature, economics, etc. The model

records user preference by giving weightings to these nodes. This idea seeks to improve from the common *bag of words* approach in storing user profile. However, SmartPush requires news providers to provide the semantic meta data. Moreover, the concept hierarchy is also determined by the content provider.

In the context of web browsing itself, there are several examples with regards to personalization systems. For example [6] uses implicit feedback to profile users' browsing behavior. In particular, the system analyzes activity logs of a proxy server that intercepts requests coming out of a gateway and logs browsing information. Topics of interest are calculated using a page interest estimator coupled with vector space techniques. From these, the system extracts an *n-gram* set of words, namely bigrams and trigrams, that represents user interest. The idea is to capture in the user profile sets of words that may have different meaning when coupled together (e.g bar tender). The paper also offers suggestions on which sets of words should be given more emphasis, such those within the bold tag and italic tag in an HTML page.

Another work by [36] describes personalized search based on a taxonomy. It uses an existing taxonomy from *Magellan*, which classifies documents into approximately 4400 nodes. Profiles are stored as concept hierarchies, as in the SmartPush system. Each node is associated with a set of documents. Each document is represented by a weighted keyword vector, determined by the number of occurrences of keywords. User browsing behavior is analyzed from web server log, and documents that are browsed are calculated into keyword vectors, and matched against documents associated with nodes. The node that corresponds to the closest match is stored in the user's profile. The system also takes into account temporal effects, i.e how long a user browses each document.

With these sets of work in mind, we propose an improvement over existing methods in the next chapter.

2.3 Graph based search techniques

This section discusses motivation and background in lieu of existing theory. The Hyperlink Induced Topic Selection (HITS) algorithm is a well known approach in information retrieval.

We can transform the relationship between a set of documents into a directed graph, in which a node represent a document and a link from node i to node j represents a reference from document i to document j. Note that document here can be a interpreted as URL's, papers, etc. We provide a brief overview, then describe and compare two variants of HITS algorithm.

2.3.1 Kleinberg's HITS algorithm

The HITS algorithm assumes an unweighted directed graph of nodes labeled from 1 to n, then calculates the *authoritativeness* and *hubness* values of each node. Define the number of edges pointing to a node to be its *in-degree* value and the number of edges pointing out of a node to be its *out-degree* value. From any such graph, we can construct an adjacency matrix $\mathbf{M}$ that encodes the relationship between any two node in an n by n matrix. M_{ij} has the value one if there is a link from node i to node j and is zero otherwise.

The *authority* measure of a node is a function of a weighted sum of all the links pointing to the node. Similarly, the *hub* measure of a node is function of a weighted sum of all the links coming out of that node. The relationship can be summarized by the following update rules:

$$\begin{aligned} a_j^+ &= \sum_i M_{ij} h_i \\ h_j^+ &= \sum_i M_{ji} a_i \end{aligned} \tag{2.1}$$

a_j is the authoritativeness measure of node j, h_j is the hubness measure of node j. a_j^+ and h_j^+ are the *update values* of a_j and h_j, respectively. From simple substitution, it follows that:

$$\begin{aligned} a_j^+ &= \sum_k M_{kj} \sum_i M_{ki}\ a_i \\ h_j^+ &= \sum_k M_{jk} \sum_i M_{ki}\ h_i \end{aligned} \tag{2.2}$$

In matrix form, the equation can be written compactly as:

$$\begin{aligned} \mathbf{a}^{+} &= \mathbf{M}^{T}\mathbf{M}\,\mathbf{a} \\ \mathbf{h}^{+} &= \mathbf{M}\mathbf{M}^{T}\,\mathbf{h} \end{aligned} \tag{2.3}$$

The measure of *authority* and *hub* for each node is a recursive function. We can use an iterative approach known as the *power method* in linear algebra [40]. Given an n element vector $\mathbf{v}$ and an $n \times n$ matrix $\mathbf{A}$, we decompose $\mathbf{A}\mathbf{v}$ to be the weighted sum of the eigenvectors $\mathbf{u}_1$,$\mathbf{u}_2$,..., $\mathbf{u}_n$ of $\mathbf{A}$ with their associated eigenvalues ordered from largest to smallest such that $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_n$:

$$\begin{aligned} \mathbf{v}^{+} = \mathbf{A}\,\mathbf{v} &= \left(\sum_i \lambda_i \mathbf{u}_i\,\mathbf{u}_i{}^{T}\right)\left(\sum_i \alpha_i \mathbf{u}_i\right) \\ &= \alpha_1\lambda_1\mathbf{u}_1 + \alpha_2\lambda_2\mathbf{u}_2 + ... + \alpha_n\lambda_n\mathbf{u}_n \end{aligned}$$

$$\begin{aligned} \mathbf{v}^{+^m} = \mathbf{A}^m\,\mathbf{v} &= \alpha_1\lambda_1^m\mathbf{u}_1 + \alpha_2\lambda_2^m\mathbf{u}_2 + ... + \alpha_n\lambda_n^m\mathbf{u}_n \\ &= \lambda_1^m\left[\alpha_1\mathbf{u}_1 + \alpha_2\left(\frac{\lambda_2}{\lambda_1}\right)^m\mathbf{u}_2 + ... + \alpha_n\left(\frac{\lambda_n}{\lambda_1}\right)^m\mathbf{u}_n\right] \end{aligned} \tag{2.4}$$

Starting with arbitrary values of $\mathbf{v} \neq \mathbf{0}$ we iterate m times to generate $\mathbf{A}^m$ as above. Since λ_1 is the largest eigenvalue in the matrix, the value of $\left(\frac{\lambda_i}{\lambda_1}\right)^m$ will near zero as $m \to \infty$ for all i. Hence, the value of $\mathbf{v}^{+} = \mathbf{A}^m\mathbf{v}$ will approximately converge to the eigenvector $\mathbf{u}_1$ associated with the largest eigenvalue λ_1.

Using the power method to iteratively solve (2.3), we note that both matrices $\mathbf{M}^T\mathbf{M}$ and $\mathbf{M}\mathbf{M}^T$ are *symmetric*. Moreover, since the value of M_{ij} is either one or zero, they are also *positive definite* matrices. Symmetric positive definite matrices has the property that all its eigenvalues are positive and all its eigenvectors are orthogonal to each other [40].

Therefore, given that we normalize the resultant vector after each iteration, HITS *guar-*

antees convergence and stability. If we define the principal eigenvector to be the eigenvector associated with the largest eigenvalue, the value of $\mathbf{v}^+$ approaches the principal eigenvector with each iteration. If there are repeated 'largest eigenvalues' the resultant vector will converge to the eigenvector associated with *one* of those eigenvalues, depending on the initial value of the multiplicand vector. In this case the steady state solution to HITS is not unique.

The resultant steady state vectors $\mathbf{a}^\infty$ and $\mathbf{h}^\infty$ hold the authoritativeness and hubness measure, respectively, of all the nodes in a directed graph.

2.3.2 Variants of HITS algorithm that supports personalization

In the context of personalization, we would like to indicate a preference on certain documents. HITS assumes all nodes are equal; their authority and hub measure are determined essentially by the number of in-degrees and out-degrees. Consider an example in which there are two clusters of document, and we indicate that we like document j in the second cluster better. Ideally, we want to 'lift' the relative authoritativeness and hubness of documents in the second cluster in relation to the whole document collection.

Kleinberg describes a method of using the non-principal eigenvectors to extract multiple clusters of hubs and authorities [22, 23]. However, it does not provide a way to place emphasis or preference on a particular cluster. Two variants of HITS that deals with such 'lifting' are introduced and described in [7]. The following discusses and analyzes each method in turn.

Single node lifting

In single node lifting, the authority and hub measure of document j is lifted by augmenting the j^{th} component of the $\mathbf{a}^+$ and $\mathbf{h}^+$ matrix at each iteration. By directly changing the value of the j^{th} element, the nodes that are connected to it as are also affected. The method is as follows:

$$\begin{aligned} \mathbf{a}^+ &= \mathbf{M}^T\mathbf{M}\,(\mathbf{a}+\boldsymbol{\delta}) \\ \boldsymbol{\delta} &= \left[0\;0\ldots\gamma a_j\ldots 0\right]^T \end{aligned} \tag{2.5}$$

We start with an initial vector $\mathbf{a}_o$ and add $\boldsymbol{\delta}$ at each iteration loop. Because the steady state value of $\mathbf{a}^+$ only depends on the eigenvectors of $\mathbf{M}^T\mathbf{M}$, this amplification of the value a_j has to happen at each step of the iteration as to affect the original steady state. Intuitively, this makes sense: since we are interested in node j, we want to amplify its value at each iteration such that the original HITS steady state solution is somewhat shifted in favor of a_j.

How does the new value differ from the principal eigenvector of $\mathbf{M}^T\mathbf{M}$? Does he new solution offer the same guarantees as the original HITS? (i.e convergence and stability). We can rewrite $(\mathbf{a}+\boldsymbol{\delta})$ and rearrange (2.5) as follows:

$$
\begin{aligned}
(\mathbf{a}+\boldsymbol{\delta}) &= \begin{bmatrix} 1 & 0 & 0 & \dots & & 0 \\ 0 & 1 & & & & \\ 0 & & \ddots & & & \vdots \\ \vdots & & & 1+\gamma & & \\ & & & & \ddots & 0 \\ 0 & \dots & & & 0 & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_j \\ \vdots \\ a_n \end{bmatrix} \\
&= \mathbf{E}\,\mathbf{a}
\end{aligned}
$$

$$
\mathbf{a}^+ = \mathbf{M}^T\mathbf{M}\,\mathbf{E}\,\mathbf{a} \tag{2.6}
$$

At this point we can apply the power method as in the original HITS, with the exception that $\mathbf{M}^T\mathbf{M}$ has now become $\mathbf{M}^T\mathbf{M}\,\mathbf{E}$. This is essentially modifying the j^{th} row and column in $\mathbf{M}^T\mathbf{M}$ by some factor $(1+\gamma)$. Therefore, nodes directly linked to node j are directly affected by this amplification. This is in accordance with what we desire.

Rewriting $\mathbf{M}^T\mathbf{M} = \boldsymbol{\Phi}$ in (2.5), we can also express the new solution in terms of the original steady state solution, as follows:

$$
\mathbf{a}^+ = \Phi\,(\mathbf{a}+\boldsymbol{\delta})
$$

$$= \Phi \mathbf{a} + \Phi \boldsymbol{\delta}$$

$$\mathbf{a}^{+m} = \Phi^m \mathbf{a} + \Phi^m \boldsymbol{\delta} + \Phi^{m-1} \boldsymbol{\delta} + ... + \Phi \boldsymbol{\delta}$$

$$= \Phi^m \mathbf{a} + \sum_{i=1}^{m} \Phi^i \boldsymbol{\delta} \tag{2.7}$$

We observe that the first element of the right hand side is the original steady state solution, and the second element is the effect of perturbation matrix $\boldsymbol{\delta}$. The resultant vector is therefore a shifted steady state solution.

Note however, the matrix $\mathbf{M}^T\mathbf{M}\,\mathbf{E}$ from (2.6) is no longer symmetric, for any $\gamma \neq 0$. Therefore, the properties of convergence and stability is no longer guaranteed.

By symmetry, the same analysis holds for $\mathbf{h}^+$.

Brief overview of gradient ascent HITS

The gradient ascent HITS has a different approach in lifting the authority and hub measures for a node. The main idea behind gradient ascent is to find the values of M_{ki} in the adjacency matrix $\mathbf{M}$ that maximizes a_j or h_j. At each iteration, the algorithm takes a small step $\gamma \cdot \Delta M_{ki}$ for all k and i in a manner that increases the relative value of a_j or h_j. Accordingly, we have a new resultant matrix $\mathbf{M}^* = \mathbf{M} + \gamma \cdot \Delta\mathbf{M}$ at each iteration! This entails to *shifting the principal eigenvector* at each iteration in the direction of our document of interest.

An in depth mathematical analysis of gradient ascent HITS will follow in the next chapter.

Comparison and analysis

Both the single node lifting and gradient ascent provide a way to 'lift' an *individual* node in a cluster of documents. To generalize this technique even further, we can lift a *series* of nodes S, each node having some score that indicates a degree of preference. The lifting can be done one at a time; each lift will re-rank the authority and hub values, from which the final result is the aggregate of each individual result weighted by the the relative document

'score' in S.

[7] claims that gradient ascent is superior to single node lifting. The single lifting approach looks at a node of interest j and directly changes the adjacency matrix so as to affect all the nodes that link to it directly. As (2.6) shows, this means changing the values of the j^{th} row and column of $\mathbf{M}^T\mathbf{M}$.

In contrast, the gradient ascent node tries to readjust the values of *all* the nodes in the matrix $\mathbf{M}^T\mathbf{M}$ so as to lift a_j. It looks at all the values in the matrix and decides which ones should increase or decrease or stay the same so as to maximize the authoritativeness or hubness of a certain document.

Intuitively, the second approach is more elegant and robust and should therefore perform better. As our experiments in chapter 5 show, this is indeed the case.

Chapter 3

Theory

In this chapter, we introduce an extension to the gradient ascent variant of HITS and describe user profiling.

3.1 Extending the gradient ascent variant of HITS

3.1.1 Analysis of gradient ascent HITS

Given an adjacency matrix $\mathbf{M}$ and an authority vector $\mathbf{a}$, gradient ascent *climbs* the steepest direction by readjusting each of the values of $\mathbf{M}$ to increase the value of a_j. The following is a mathematical analysis of the method. As before, the analysis for a_j holds also for h_j by symmetry.

We can rearrange the equation (2.2) as follows:

$$\begin{aligned} a_j^+ &= \sum_k M_{kj} \sum_i M_{ki}\, a_i \\ a_j^+ &= \sum_i \left(\sum_k M_{ki} M_{kj} \right) a_i \end{aligned} \tag{3.1}$$

Let's consider adding an increment ΔM_{ki} for all the values k and i in M_{ki}, and call the

resulting vector $\mathbf{a}^*$. Its j^{th} element would have the value:

$$a_j^* = \sum_i \left(\sum_k (M_{ki} + \Delta M_{ki})\, M_{kj} \right) a_i \tag{3.2}$$

For a specific k and i, we can define $a_{\delta j}$ to be:

$$a_{\delta j}^+ = M_{ki}\, M_{kj}\, a_i$$

$$a_j^+ = \sum_{1 \le i \le k \le n} a_{\delta j}^+ \tag{3.3}$$

Comparing the difference between the j^{th} component in $\mathbf{a}^*$ and $\mathbf{a}$ for a certain ΔM_{ki} (i.e fixed k and i), we can find the derivative $\frac{\delta a_{\delta j}}{\delta M_{ki}}$ as follows:

$$a_{\delta j}^* - a_{\delta j}^+ = \Delta M_{ki}\, M_{kj}\, a_i$$

$$\frac{a_{\delta j}^* - a_{\delta j}^+}{\Delta M_{ki}} = M_{kj}\, a_i$$

$$\lim_{\Delta M_{ki} \to 0} \frac{a_{\delta j}^* - a_{\delta j}^+}{\Delta M_{ki}} = \frac{\delta a_{\delta j}^*}{\delta M_{ki}} = M_{kj}\, a_i \tag{3.4}$$

Now that we know the derivative of $a_{\delta j}$ with respect to M_{ki}, we are interested in knowing for what values of M_{ki} does a_j increase. Given adjacency matrix $\mathbf{M}$ and an authority vector $\mathbf{a}$, the first order Taylor series expansion of a_j is as follows:

$$a_j^* = a_j^+ + \sum_{1 \le k \le i \le n} \frac{\delta a_{\delta j}^*}{\delta M_{ki}} \cdot \Delta M_{ki}$$

$$a_j^* = a_j^+ + \sum_{1 \le k \le i \le n} (M_{kj}\, a_i) \cdot \Delta M_{ki} \tag{3.5}$$

We verify by observing $a_j^* = a_j^+$ for $\Delta M_{ki} = 0$. Choosing $\Delta M_{ki} = M_{kj}\, a_i$, we have:

$$a_j^* = a_j + \sum_{1 \le k \le i \le n} (M_{kj}\, a_i)^2 \tag{3.6}$$

Since $(M_{kj}\, a_i)^2 \geq 0$, choosing $\Delta M_{ki} = M_{kj}\, a_i$ has the effect of increasing the value of a_j, or in the worst case a_j stays the same. The gradient ascent algorithm is as follows: at each iteration, increase the value of M_{ki} by $\gamma \cdot \frac{\Delta M_{ki} = M_{kj}\, a_i}{\sum_i M_{kj}\, a_i}$, so as to maximize the value of a_j. The value $\frac{\gamma}{\sum_i M_{kj}\, a_i}$ is a normalizing factor, with γ being the step size of the ascend. The factor $(\sum_i M_{kj}\, a_i)^{-1}$ comes naturally, since the authoritativeness of node a_i is reflected in the i^{th} row of $\mathbf{M}$. For hub measures, the normalizing factor would be the sum over the i^{th} column. However, these constants have rather secondary effects as compared to δM_{ki}. A 'good' constant is one with a step size big enough to make significant progress in ascending, and one small enough as to avoid oscillation around a maxima. To summarize, the adjacency matrix is updated at each iteration as follows:

$$\begin{aligned} \mathbf{M}^* &= \mathbf{M} + \Delta\mathbf{M} \\ \mathbf{a}^+ &= \mathbf{M}^{*T}\mathbf{M}^*\,\mathbf{a} \end{aligned} \tag{3.7}$$

This variant of HITS suffers from the common malaise of gradient ascent algorithms, namely the trap of local maxima. At each step of the iteration, the algorithm tries to maximize *locally* what step to take. The value of $\mathbf{M}^*$ that produces a possible increase in a_j is maximized with respect to the previous $\mathbf{M}$, not the *original* $\mathbf{M}$. Therefore, it only has a local view and optimize in that sense. Nevertheless, the algorithm does shift the solution away from the original eigenvector of $\mathbf{M}^T\mathbf{M}$ towards a new value that tries to increase a_j in the resultant vector $\mathbf{a}$.

In terms of convergence and stability, the matrix $\mathbf{M}^{*T}\mathbf{M}^*$ is also a symmetric positive definite matrix, and hence the properties of the original HITS. However, since the value of $\mathbf{M}^*$ keep changing at each iteration, the steady state solution changes with it. Then convergence is not guaranteed. As mentioned above, depending on the step size, the algorithm may oscillate around a local maxima.

3.1.2 Extending gradient ascent HITS

We would like to extend the above algorithm with the following heuristic: *decrease the value of all nodes* $l \neq j$. From (3.6), we had derived an update rule for node j. We can use the

same analogy to derive an update rule for $l \neq j$:

$$a_l^* = a_l - \sum_{1 \leq k \leq i \leq n} (M_{kl}\, a_i)^2 \tag{3.8}$$

The above implies $\Delta M_{ki} = -M_{kj}\, a_i$.

We would like to lower the weights of nodes $l \neq j$. The motivation is that we would like to see a *faster rate of change* for readjusting the relative rankings of authorities and hubs, by doing both gradient ascent and descent at the same time. Adding this simple heuristic is a natural extension of existing theory. [7] mentions the use of gradient descent to reduce the authority of irrelevant documents, but claims that negative weight values of a_j complicates the analysis.

The following sections introduce two methods that explores this heuristic a step further.

Combination one

In this first extension, we note that the resultant ΔM_{ki} is the sum of all the ΔM_{ki_x} for all $1 \leq x \leq n$.

$$\begin{aligned} \Delta M_{ki} &= \Delta M_{ki_1} + \Delta M_{ki_1} + ... + \Delta M_{ki_1} \\ \Delta M_{ki} &= \Delta M_{ki_j} + \sum_{l, l \neq j} \Delta M_{ki_l} \end{aligned} \tag{3.9}$$

Note that the first part of the right hand side is positive, while the second element is negative to satisfy $\Delta M_{ki_l} = -M_{kl}\, a_i$. Here we will take preventive measures to minimize the possible cancelling effects by averaging the contribution from ΔM_{ki_l} in a set of N nodes:

$$\Delta M_{ki} = \Delta M_{ki_j} + \frac{1}{N-1} \sum_{l, l \neq j} \Delta M_{ki_l} \tag{3.10}$$

The average contribution of the nodes $l \neq j$ may still be greater than ΔM_{ki_j}. For this we apply the following rule: *if* $-\Delta M_{ki_j} \leq \frac{1}{N-1} \sum_{l, l \neq j} \Delta M_{ki_l}$, *then* $\Delta M_{ki} = \Delta M_{ki_j}$ *else use* (3.10). This is to say: if the effects of ΔM_{ki} from lowering the authorities of all nodes $l \neq j$ is such that the authority of node j is lowered, we *ignore* their contributions to ΔM_{ki}. We

note that the new ΔM_{ki} no longer satisfies the original equation (3.6) and we observe that as long as $\Delta M_k i \geq 0$, the second element of the right hand side of (3.6) will still lead to an increase the value a_j.

Combination two

In this second extension, we loosen our previous restriction, and for all cases, let:

$$\Delta M_{ki} = \Delta M_{ki_j} + \frac{1}{N-1} \sum_{l, l \neq j} \Delta M_{ki_l} \tag{3.11}$$

Now it is possible to have a negative value for ΔM_{ki}. The justification for this method is the fact that we essentially care only about the relative rankings of the node, and even if lowering node l contributes to lowering the *nominal* value of a_j or h_j, we allow it to happen because the *relative* value of a_j is still greater than the lowered node a_l.

For a gradient ascent and descent algorithm running at n iteration, we can do the following to attenuate possible negative values of ΔM_{ki}:

- At *even* values of n, calculate ΔM_{ki} as in (3.11).
- At *odd* values of n, calculate ΔM_{ki} as in the original gradient ascent (3.6).

Alternating between these two give the algorithm some time to readjust its values in the cases where M_{ki} is negative.

In general, we expect both combinations to have a faster rate of readjustment than the normal gradient ascent HITS. We analyze the performance of each of these variants in chapter 5.

3.2 Ontology based user profile modeling

3.2.1 Design criteria and choice

Gradient ascent HITS provides a method to bias certain documents in a graph based structure. What is lacking, however, is a method of keeping track the history of user interests. In

this respect, we propose a model of user profiling to complement our theoretical extension of graph based search.

We found that most user models lie in the common *bag of words* approach. Strings of words that represent user interests are kept in the form of web browser *cookies* or files. Moreover, most user models do not take into account what users *dislike.*

In the context of our search engine, this approach is inadequate for several reasons. Firstly, the bag of words approach does not consider the semantics of words. For example, when users indicate liking for 'cars', this approach does not consider other words such as 'automobile.' Likewise, when users indicate liking for 'rose' in the sense of 'wine,' as opposed to 'flower,' the bag of words approach lacks an efficient method to make a proper distinction. In other words, we run into the problem of homonyms and synonyms as introduced in chapter 1. Secondly, by ignoring what users dislike, the system does not learn from past mistakes. Though this approach of using only positive feedback is safer, it does not put the set of dislikes in proper perspective.

Other methods such as Bayesian network estimator and PRM as discussed in chapter 2 are not feasible. These techniques assume a large or statistically significant number of users. We would like our user profiling to be part of a stand alone application. Moreover, such techniques usually have a deterministic set of variables it is comparing to, such as time and location in [12], and a set of named entities in [17]. The same limitations apply to neural network learning models. Therefore, these models do not work well with our adaptive user profiling needs.

We propose an approach that uses a tree coloring technique. The tree is an *Open Directory Project* (ODP) taxonomy, which contains nodes that represent semantic contexts of web pages. We keep a record of visited nodes, and 'color' each by the number of times it has been visited, the number of times the user rates it positive or negative, and URL's that the node associates with.

We chose ODP for several reasons:

- The ODP is a collaboration of 35000 web enthusiasts working together in an effort to the web into a giant, readable structure. It covers a major subset of everything

available through the Internet.

- The ODP is an industry standard. Its structure is used by various popular search engines such as *Yahoo!* and *Lycos*. Given such endorsements, the validity of its function and use is unquestionable.

- It is open source, and hence all its implications. The RDF dump is available at [http://dmoz.org/rdf/].

ODP is a multiply connected tree. In the tree itself, ODP has only one parent, but its format allows multiple aliasing, so in effect a node may have multiple parents. In addition, each node is associated with an ID number and a set of 'leaves' which are the web pages associated with the node.

The user profile is a mapping from a context to a set of ODP nodes. A context is defined as a user query. For each query, the system generates a set of pages. Users can rate pages as being 'good' or 'unrelated.' Since each page is associated with a node in ODP, this feedback is updated into the user profile. Each entry in the mapping has the number of times the node has been visited, the number of positive and negative feedback for the node and the set of URL associated with it. Figure 3-1 (a) displays a schema of the user profile.

From our discussion in chapter 2, we found several methods that also uses this tree coloring scheme. Note, however, that this approach, although along the same lines, are distinct from these other methods. [36] uses a tree weighting scheme to calculate the vector space model of documents associated with each node. Documents of interests are processed and its vector space value matched against the vector space values of the nodes. The weights - in our case, colors - of the tree does *not* change. The other example, SmartPush [27], uses a taxonomy provided by news provider to determine which news articles to reccomend. This tree is dynamic, but does not cover the breadth and depth of our proposed system. Hence, although the idea of tree coloring is not new in itself, the way we combine it with the system is - at least to the author's knowledge - quite distinct.

The next section describes the mechanisms of the user profile in detail.

3.2.2 Use cases

The following summarizes our user modeling technique:

- *Data gathering*

  Our model gathers data using explicit feedback. Users are allowed to rate a certain context positively and negatively. Feedback will be recorded in the user's profile.

- *Representation*

  A user profile is a mapping from contexts to nodes. One context may map to several nodes. For example, the context 'car' may map to nodes that represent 'Honda', 'Volvo', etc. Each node has a 'color' that encodes the number of times it has been visited, rated positive, negative, and associated URL's.

- *Interpretation*

  The table is kept as the user profile. When a user submits a query, the system does a table lookup to find the context. The following happens:

  - If found, the system looks at the mapping of nodes, and accordingly give more or less weighting to its associated URL's to be filtered.
  - If not found, the system tries to associate that context with an ODP node. There may be several nodes that can be associated with the context. For each of these nodes, look up all the nodes in the table and check if either:

    1. The node associated with the query has the same parent as any node in the table.
    2. The node associated with the query is a child of any of the node in the table.

    If any of the above is true, return the associated URL's and their respective weights to be filtered; else, return nothing.

The results are passed back to the graph based search algorithm from the previous section. Nodes with positive weights are given positive bias, while nodes with negative weightings are given negative bias. Note that the current prototype implementation simply filters out

the negatively weighted URL's. It searches only up to one depth up and down the tree to look for parents and children when comparing nodes.

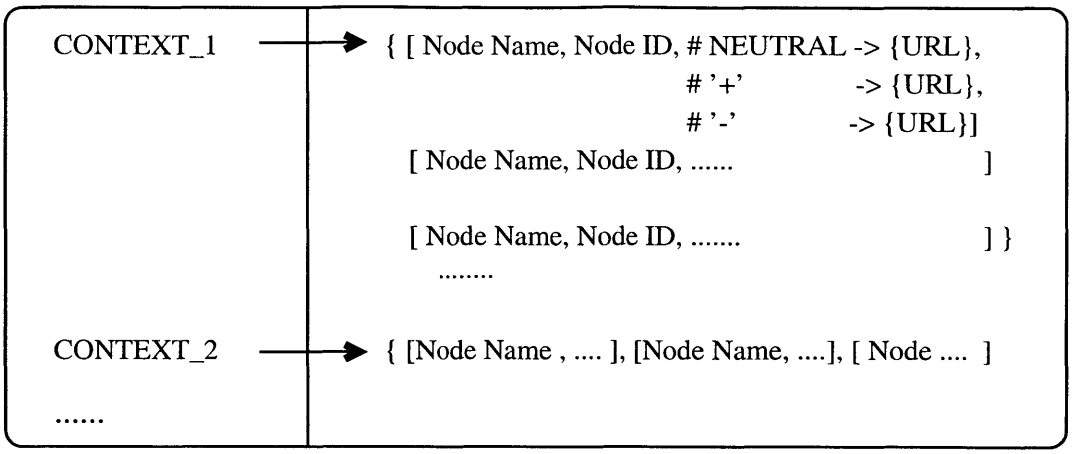
We note that there is much room for improvements. For example, generalizing the node searching mechanism up to n nodes up and down the classification tree, we observe that the nodes are more generic as they reach the root node. We can add the following simple heuristic: *the closer a node is to the root, the less depth the tree will be searched.* So instead of finding up to depth n for each node, the depth should be a function of how close a node is to a tree.

To illustrate clearly our profiling system, we give a simple example as drawn in Figure 3-1 (b). In the past, a user had queried 'vision' and was given two set of results, one relating health, another regarding computer science (machine vision). The user indicated that he or she preferred vision in the health sense, and rated vision in the computer science sense as negative. Next the user queries the word 'virus'. The system does not have any information regarding the user's preference on 'virus' and therefore looks at all the nodes in the profile table. Searching for 'virus' in ODP, the system finds two nodes, one in *health/virus* and another in *computer_science/virus.* Matching the user's profile shows that the user has indicated interest in the node *health/vision*, a negative weighting on *computer_science/machine_vision*, finds that *health/vision* has the same parent as *health/virus* and incoorporates this fact in returning the results.

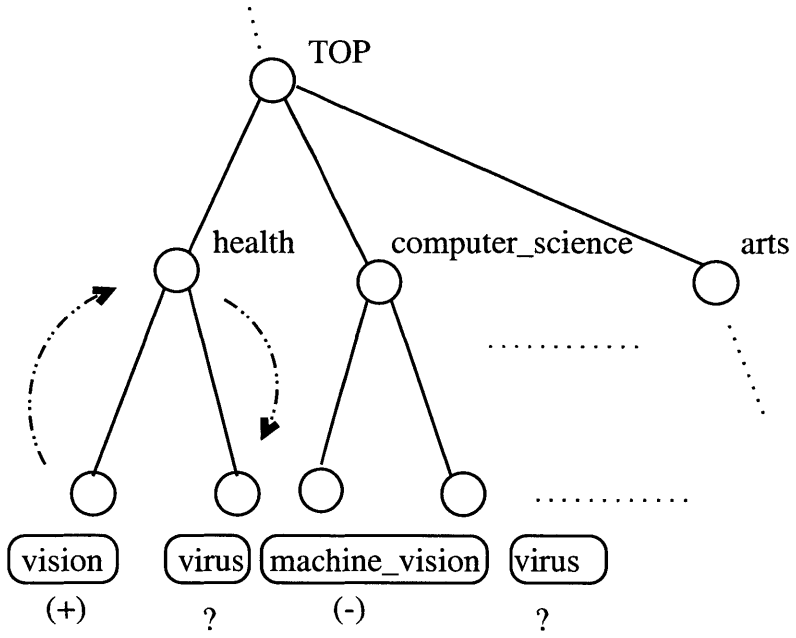
Clearly, this ontological approach is beneficial in the sense that the system can better predict user interest and further classify them into separate categories - leveraging on the semantics inherent in an ontology. Instead of trying to capture the meaning of words *per se*, an ontological profile captures the semantics of user queries, thus enabling it to find synonyms or related contexts as well as hierarchical relationships. Therefore we overcome many limitations of the standard bag of words approach.

Moreover, recent work by [24] introduces a technique for static matching of several users who had two 'colored' ontologies. This method has possible applications in the area of interest matching and collaborative filtering.

In summary, the proposed user model introduces improvements over the common bag of words approach and various other techniques. Given the design constraints, we feel that it



(a) User Profile



(b) Sample case

Figure 3-1: Building and using the user profile (a) depicts a schema of the user profile, (b) is an example of a use case

is the most feasible technique. The model uses open industry standards, endorsed by widely used products. The model is quite scalable and modular and is easily extendible to other applications.

Chapter 4

System Design and Implementation

This chapter describes the system design and implementation. We explain each of the components of the system, starting from the high level view to the low level view of our implementation.

4.1 System overview

The *Persona* system is a personalization module that wraps around a search engine. It lies between the search engine and the end user, refining the results coming out of a 'normal' search engine. Figure 4.1 shows a diagram of the system.

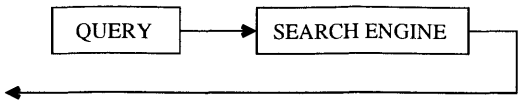
The system consists of a front end and a back end. The front end interfaces with the user, accepting queries and user feedback. These are then passed to the back end, which processes these queries and builds the user profile. The back end core of *Persona* consists of two main modules: a filtering mechanism and a user profiling system. The following sections describe each in turn.

4.1.1 Overview of filtering mechanism

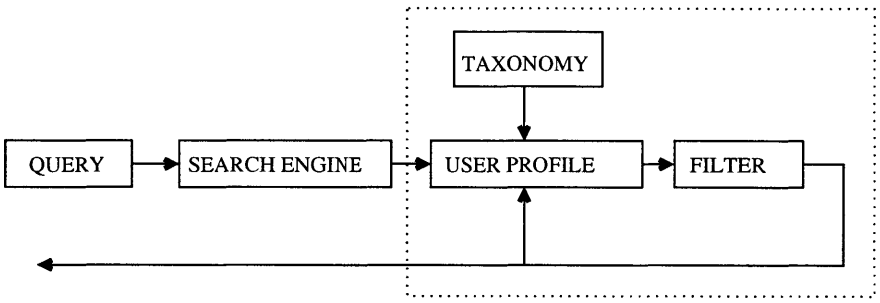
The brief overview of the filtering mechanism is as follows:

1. *Query input*

   The user inputs a query, which will then be outsourced to *dmoz*, an existing search



(a) 'normal' search engine



(b) Persona system wraps around existing search engine

Figure 4-1: System diagram

engine - the result of which will be filtered to leave the top n results. These n documents corresponds to nodes in the ODP taxonomy.

2. *Personalization Agent*

   First, the system consults the user profile to check the user's history. If there is a match, the system incorporates past user preference. If not, a 'normal' filtering module then processes these results. The HITS algorithm as described in [22] dictates the following:

   (a) Call the initial set of results the *root set.*

   (b) Expand the *root set* by order one distance, such that all web pages that point from and to the root set are included. This set of node is the *base set.*

   (c) Treating each web page as a node and each URL in that web page as a link, create a directed graph structure consisting of all members of the base set. In this calculation, the nodes that are from the same domain are not taken into account and are thus filtered out.

(d) Using the number of in-degrees and out-degrees from each node, the algorithm calculates the *authoritativeness* and *hubness* of each node. Based on a node's authoritativeness, the results are ranked accordingly.

(e) The ranked results are then updated against the user profile.

3. *Feedback-based Result Refinement*

   The system returns the filtered results to the user. The user may give positive or negative feedback on the returned set of documents. The system will then *refine* the current results based on these preferences, giving more weighting to the positively rated pages and less to the negatively rated pages.

   In addition, these user feedback are incorporated into the user profile. The user profiling is discussed in the next section.

4.1.2 User Profile

The user profile relies on relevance feedback. Each positive and negative feedback serves two function. First is to refine the set of searches and re-rank the results. Second is to build the user's profile. The user feedback is updated by 'coloring' nodes as we described in the previous chapter.

Each entry in the user profile consists of a context word, which consists of queries that the user types in. Each of this entry is associated with a set of nodes. Each node in the ODP has a unique identifier. The user profile keeps track of how many times each node in the profile is visited, the number of positive ratings, negative ratings, as well as the set of URLs associated with it.

In this manner, we do not keep the whole ODP taxonomy inside the user profile. We only keep track of the nodes that has been colored. The user profile then can be kept small, allowing for scalability.

The system consults the user profile at every new query. If a query exists in the user profile, it returns the set of URL's associated with the colored nodes. If there is no data on the query, the user profile finds the set of ODP nodes that closely matches the query and

tries to find nodes in the profile that may be its parents or children.

In the case of user feedback, the user profile simple colors related nodes and passes on the bias information to the variants of HITS. These variants will take the bias into account and return the set of most related documents.

We note that most search engines have the feature that lets users browse through 'similar pages.' We claim that this refining technique is different from ours. Finding 'similar pages' usually entails returning the closest document set that the search engine indexes. In contrast, our filtering mechanism expands a set of URL's and emphasize those with positive feedback. It expands up to depth three down to create a new graph structure, and lift those documents which are positive.

The following section discusses implementation details.

4.2 Implementation

Implementation is done mostly in Java, with some Perl script to process the front end. Figure 4.2.1 shows a high level Module Dependency Diagram for the current back end implementation. The design follows an object oriented methodology approach. The main pieces of the system are as follows:

1. *Daemon.* Listens for input from a port. The format of 'legal' input is specified in an XML Document Type Definition (DTD), and thus uses XML as an interface.

2. *Result Container Data Structure.* The input stream from the daemon is transformed into Java objects, in particular into data structures that encapsulate the necessary information.

3. *User Profile.* Captures the user's interest. Implemented as a table that maps keywords to context and its associated ODP nodes.

4. *Filter.* Encapsulates all the graph based search techniques we discussed in chapter 2 and 3. The filter module does all the calculation and filtering to determine authoritativeness and hubness.

5. *Controller.* Controls the transaction between all modules. Manages the logic and control flow among each of the components. The controller interfaces with the input daemon to pass back results.

6. *Ontology Parser.* Parses in and traverses the ODP taxonomy. The parser identifies each node in the ODP by its ID number and name. The parser can return information such as a node's parent, children, and the set of nodes that correspond to a certain string.

4.2.1 Design Rationale

The rationale behind the design is to delegate pieces of the process each into separate components. Each module represents a unique task. The interaction between them is handled by the *Controller* module, whose main objective is to minimize the interaction between module. Using a Bayesian network analogy, the controller acts as a hidden node that reduces the complexity of the system, while keeping the same dependency structure.

This setup allows for better flow. Future work involves a programmer taking out a module without disrupting too much the rest of the system. Each of the components can act as a stand alone, hence enforcing modularity in the system. This logic also presents less convoluted design.

4.2.2 Interfacing

The front end interfaces with the back end using an XML DTD. The schema consists of information needed to process the query, such as user name, user id, and each of the link. Each link must contain information such as its URL, associated ODP node, title, description, rank, and whether the user rated it as positive, negative, or neutral. Figure 4.2.3 describes the DTD schema.

We chose XML as an interfacing language because it is an industry standard and parsers for XML are widely available.

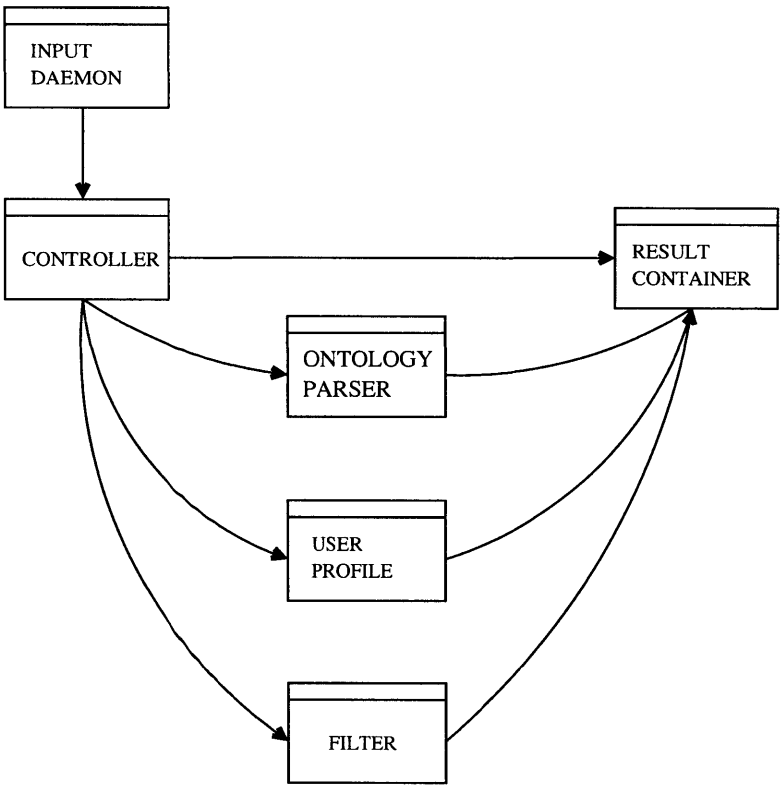


Figure 4-2: High level Module Dependency Diagram

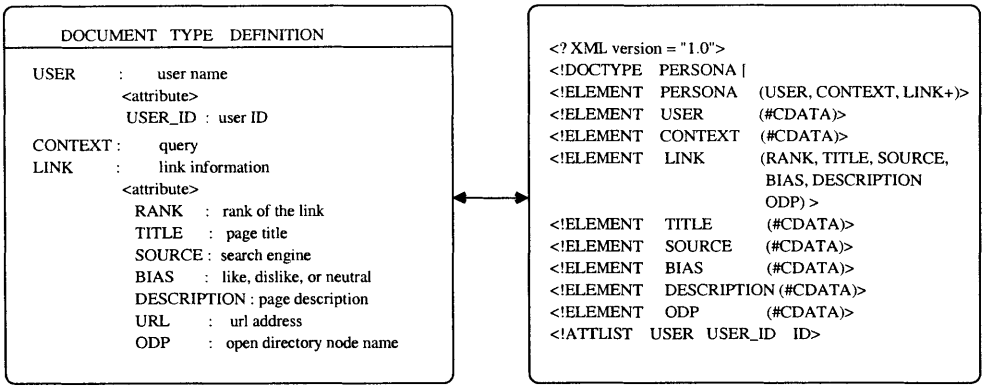


Figure 4-3: XML DTD (Document Type Definition)

4.2.3 Implementation notes

- For the *Persona* prototype, we use the *Health* and *Computers* section of the ODP taxonomy. *DMOZ* provides the file in RDF format. The whole file is 200MB in size, while the *Health* and *Computers* section has 6MB total.
- Our prototype currently lacks a login manager. The back end handles multiple users, but a possible immediate improvement is to add a login page at the front end.
- We use Apache's *Xerces* parser to parse incoming XML.

Chapter 5

Validation and Results

To validate the theory, we run sets of controlled experiment to complement our prototype. There are several issues with regards to validation. Since *Persona* is implementation heavy, it is of first and foremost importance to test whether the system works in accordance to the prescribed algorithms. Both the mathematics and the calculations implementation must behave exactly as the algorithm dictates. In addition, we wish to evaluate how useful each algorithm performs in predicting user preference. Thus we separate the tests into two categories: *behavior correctness* and *algorithm effectiveness*. The former set is a requisite of the latter.

To test behavior correctness, we perform tests at the lower level. Since our main interest is to analyze the performance of the gradient ascent variant of HITS, we will do the following. First we check the implementation for normal HITS. Next we check the implementation of gradient ascent HITS. Then we compare the performance of gradient ascent HITS against the single node lifting variant.

Next we test the effectiveness of each algorithm. The normal HITS algorithm acts as a benchmark. We lift a document and test how much of an improvement does each algorithm offer. To make these tests clear cut, we first only compare the single node lifting to the gradient ascent method using one sample case. Then we compare the performance of all four in a set of simulations. The reason is that we first want to have a rough idea how the single node lifting variant performs against the gradient ascent variant. It does not give

more insight to compare each of the four algorithms in pairs. We only incorporate all four techniques when we have a better understanding of how the two behaves.

The following sections describes the tests in detail.

5.1 Matrix level validation

Tests at the matrix level deal directly with the nodes. The algorithm takes as its input the following parameters:

- *Adjacency matrix A.* A is a binary $N \times N$ matrix that encodes the relationship between all the nodes in a directed graph structure of size n.
- *Number of iteration.* The HITS algorithm and all its variants tries to find a steady state solution. In practice, we only concern ourselves with the relative values of the elements of the eigenvector. After three to five iterations, the relative values changes very little. Hence in *Persona* the number of iterations is set to four.
- *Authority or Hub.* A boolean parameter that determines whether the desired vector to be an authority vector or a hub vector.
- *Gamma(γ).* For the gradient-ascent HITS, the value γ determines the 'step' of the ascent, while for the single node lifting γ is the size of perturbation.

5.1.1 HITS algorithm

The pseudo code for the HITS algorithm is as follows:

```
   HITS (adjacency-matrix A, num-of-iteration i, boolean hub)
:  Matrix v = new Matrix(sizeOf(A) × 1)
:  Matrix product = new Matrix(sizeOf(A) × sizeOf(A))
:
:  if (hub) then product = A × A^T
:  else product = A^T × A
:
```

```
:   for j ← 0 up to i
:     v = product × v
:     v.normalize()
:   end loop
:
:   return v
```

All the nodes in the $N \times 1$ vector v are initially set to be one. To find hub values, we set the multiplicand matrix to be $A \times A^T$, whereas for the authority values, we set the multiplicand to be $A^T \times A$. After each iteration, the vector v should step closer to the principal eigenvector of the multiplicand matrix. In practice, we seek only the relative values of each of the nodes as opposed to their nominal values, for which a fixed number of iteration will suffice.

Simple Experiment

To check the correctness of behavior of the HITS implementation, we set up the following experiment: Create a structure with 10 nodes, numbered from 1, 2, ... , 10. Each node i points to all the nodes whose value are greater than i, i.e node 4 points to nodes 5, 6, ... 10; node 5 points to 6, 7, ... 10. Thus node 1 has in-degree zero and out-degree 10, node 2 has in-degree 1 and out-degree 9, ..., node 10 has in-degree 10 and out-degree 0. Figure 5-1 displays the node graph of the structure.

From this figure, we infer the following adjacency matrix A.

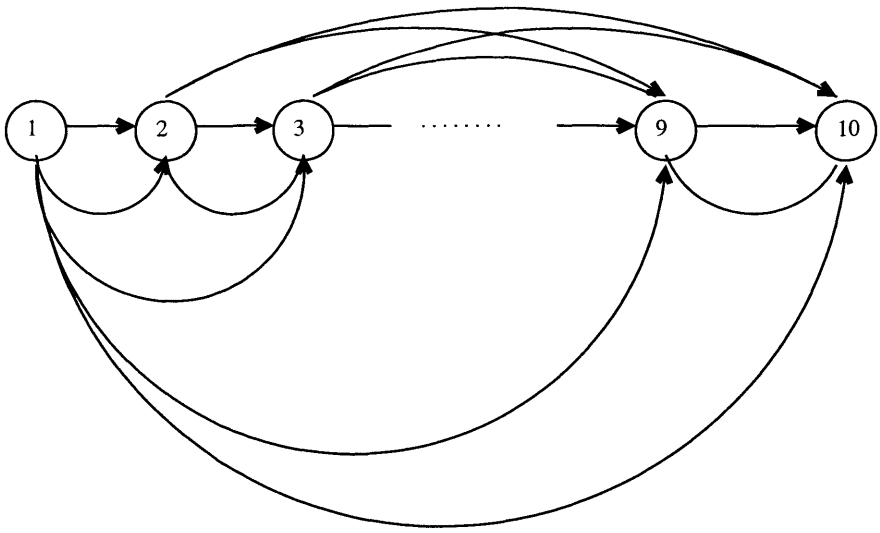


Figure 5-1: Each Node i points to all nodes i+1, i+2, ..., 10

$$\begin{bmatrix}
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

The HITS algorithm says that adjacency matrix A above has the most authoritative node to be 10, followed by 9, 8, 7, ... 1. The most hub node to be 1, followed by 2, 3, ... 10. We run the program *HITS* with input A, number of iteration 3, for both the hub and authority values. Table 5.1 summarizes the results.

Indeed the outcome closely follows the prediction of the HITS algorithm. Hence we claim

Node i	hubness		authoritativeness	
	Relative Rank	Nominal Value	Relative Rank	Nominal Value
1	1	0.457	10	0.0
2	2	0.445	9	0.076
3	3	0.420	8	0.149
4	4	0.384	7	0.219
5	5	0.338	6	0.282
6	6	0.282	5	0.338
7	7	0.219	4	0.384
8	8	0.149	3	0.420
9	9	0.076	2	0.445
10	10	0.0	1	0.457

Table 5.1: Nominal value and rank results for *hubness* and *authoritativeness*

the implementation behavior of normal HITS is as expected at the node level. Note also the symmetry between the hubs and the authority values. If we reverse all the arrows from Figure 5-1 to point the other way, the result will be the inverse of the current result.

Further experiment

To further test the program, we set up the following experiment: The HITS algorithm endorses higher authority for nodes which are pointed to by 'good' hubs. In other words, given two nodes with the same number of in-degrees, the one pointed to by a 'better' hub will be given more weight. A 'better' hub is defined to be a node with more number of out-degrees.

Figure 5-2 illustrates such a situation. In this case, nodes 2, 3, 4, and 6 has the same number of in-degree. However, nodes 2, 3, and 4 is pointed to by node 1, which is a 'better' hub than node 5. Thus, the HITS algorithm expects these nodes to have higher weighting than the latter node.

Table 5.2 summarizes the result from running the program with three iteration. Again, we see the the results indeed conforms with prediction. Nodes 2, 3, and 4 has significantly higher nominal values than node 6. As such, the ranking shifts accordingly. Given these results, we conclude the validation for simple HITS implementation.

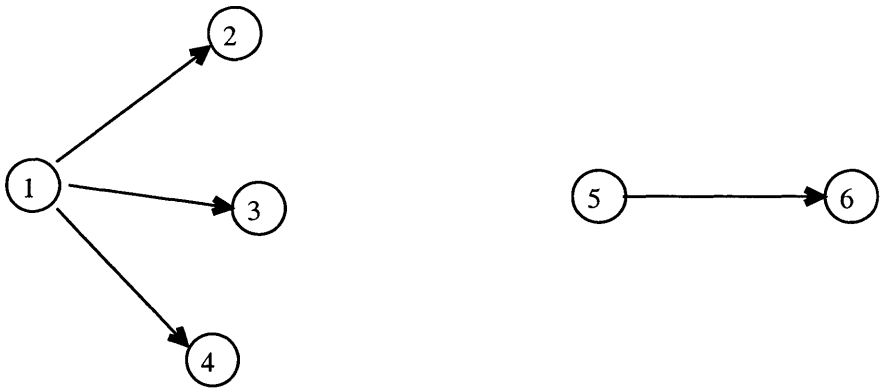


Figure 5-2: Nodes 2, 3, 4, 6 has the same number of in-degree

Node i	hubness		authoritativeness	
	Relative Rank	Nominal Value	Relative Rank	Nominal Value
1	1	0.999	3	0.0
2	3	0.0	1	0.577
3	3	0.0	1	0.577
4	3	0.0	1	0.577
5	2	0.004	3	0.0
6	3	0.0	2	0.022

Table 5.2: Nodes pointed to by 'better' hub are rated higher

5.1.2 Gradient ascent HITS algorithm

The pseudo code for the gradient-ascent HITS algorithm is as follows:

```
  gradientHITS(adjacency-matrix A, num-of-iteration i, boolean hub, gamma γ,
lift-index m)
:  Matrix v = new Matrix(sizeOf(A) × 1)
:  Matrix product = new Matrix(sizeOf(A) × sizeOf(A))
:
:  v = HITS(A, 1, hub)
:  i--
:
:  for j ← 0 up to i
:     if (hub) then gradient ∀k.l △Akl = Amk vl
:     else gradient ∀k.l △Akl = Akm vl
:
:     Akl = Akl + γ · △Akl / Σl △Akl
:
:     if (hub) then product = A × A^T
:     else product = A^T × A
:
:     v = product × v
:     v.normalize()
:  end loop
:
:  return v
```

As explained in chapter 2, gradient-ascent HITS is similar to normal HITS in many respects. The two additional input parameters are γ - which determines the impact of the lift node, and the lift index m - which is the index of the node to be lifted.

On the first iteration, assign the vector to be the normal HITS result. From there, we

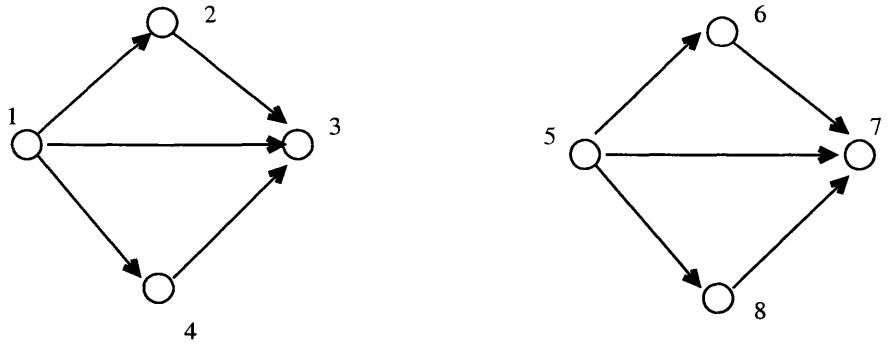


Figure 5-3: A set of two identical structures

Node i	original authoritativeness		new authoritativeness	
	Relative Rank	**Nominal Value**	**Relative Rank**	**Nominal Value**
1	3	0.0	3	0.0
2	2	0.291	4	0.216
3	1	0.575	3	0.395
4	2	0.291	4	0.216
5	3	0.0	5	0.0
6	2	0.291	2	0.425
7	1	0.575	1	0.624
8	2	0.291	2	0.425

Table 5.3: Effect of lifting node 6 towards authoritativeness.

modify the adjacency matrix at *each* succeeding iteration using the lift-node v_m. The impact of each lift nodes is affected by γ. If we set γ to be zero, the result will be the same as if the normal HITS is run.

To test whether our implementation works, we devise the following experiment: Given two sets of identical graph structure, specify a lift-node v_m. Taking v_m into account, recalculate the ranks for each of the elements in eigenvector v. Gradient ascent HITS predicts that nodes connected to v_m will rank higher in this setup.

Figure 5-3 illustrates such a graph. Nodes 1, 2, 3, 4 is identical in their relation to nodes 4, 5, 6, 7, 8. We determine node 6 to be the lift-node. Table 5.3 and Table 5.4 summarizes the result of running the gradient ascent HITS in three iterations.

Table 5.3 and Table 5.4 shows that lifting node 6 does dramatical change to the relative

Node i	original hubness		new hubness	
	Relative Rank	Nominal Value	Relative Rank	Nominal Value
1	1	0.575	2	0.545
2	2	0.291	5	0.259
3	3	0.0	6	0.0
4	2	0.291	5	0.259
5	1	0.575	1	0.548
6	2	0.291	4	0.263
7	3	0.0	3	0.359
8	2	0.291	4	0.263

Table 5.4: Affect of lifting node 6 towards hubness.

rankings of these nodes. In terms of authoritativeness, node 7 is now ranked first, and the rankings of node 6 and 8 are lifted. In terms of hubness, node 5 is now ranked first, and nodes 6, 7, and 8, all related to node 6 are lifted as well. With these results, we claim that our implementation of gradient ascent HITS behaves as expected.

5.1.3 Comparing gradient-ascent HITS with the single-node-lifting variant

Now that the effectiveness of gradient-ascent HITS over its plain version is verified, we want to compare the former to the single node lifting variant.

Single node lifting merely lifts one node of interest by increasing its relative value. This 'lifting' is done at each iteration of the HITS algorithm, hence by lifting nodes that are related to the 'lifter' node. The pseudo-code for single node lifting is as follows:

```
singleLiftHITS(adjacency-matrix A, num-of-iteration i, boolean hub, gamma
γ, lift-index m)
:  Matrix v = new Matrix(sizeOf(A) × 1)
:  Matrix product = new Matrix(sizeOf(A) × sizeOf(A))
:
:  if (hub) then product = A × A^T
:  else product = A^T × A
```

```
:
:   for j ← 0 up to i
:      v_lifter = (1 + γ) · v_lifter
:      v = product × v
:      v.normalize()
:   end loop
:
:   return v
```

Note that the pseudo-code is very similar to the plain HITS algorithm, the only difference being the single line modification of $v_{lifter} = (1+\gamma) \cdot v_{lifter}$ during the iteration loop. At each iteration, the value of the 'lifter' node is increased, thereby affecting the nodes that points to or are pointed by it, as captured in the *product* matrix.

We would like to compare this single-lifting method with the gradient ascent method. To give a fair comparison, we run the same tests above using two identical sets of nodes. As in the case in our gradient ascent HITS test, we lift one of the nodes in the set, and compare the performance of lifting toward the overall relative rankings of the nodes.

To contrast the two algorithms, we design two test cases, one in which the results are similar, and another in which the gradient ascent variant of HITS performs better. Discussion follows each test.

Special case that yields similar results for both algorithms

In this special case, we reuse the simple test done in our previous analysis of gradient ascent HITS. We have two identical structures as shown in Figure 5-3. Note that for each node i, if there is a path from node i to node j, for $i \neq j$, then node i can reach node j in one 'hop.' In other words, the longest minimum distance is one.

The results are shown in tables 5.6 and 5.7. For this particular setup, the results are comparable. The parameters of this experiment are: number of iteration is three, γ is 0.8. In both cases, the lifted node and its neighbors all have relatively higher ratings than before.

Node i	original authoritativeness		original hubness	
	Relative Rank	Nominal Value	Relative Rank	Nominal Value
1	3	0.0	1	0.575
2	2	0.291	2	0.291
3	1	0.575	3	0.0
4	2	0.291	2	0.291
5	3	0.0	1	0.575
6	2	0.291	2	0.291
7	1	0.575	3	0.0
8	2	0.291	2	0.291

Table 5.5: Effect of lifting node 6 towards authoritativeness.

Node i	single-lifting		gradient-ascent	
	Relative Rank	Nominal Value	Relative Rank	Nominal Value
1	2	0.431	2	0.545
2	4	0.218	5	0.259
3	5	0.0	6	0.0
4	4	0.218	5	0.259
5	1	0.661	1	0.548
6	3	0.371	4	0.263
7	5	0.0	3	0.360
8	3	0.371	4	0.263

Table 5.6: Comparisons of 'hubness' values from running the two methods

Node i	single-lifting		gradient-ascent	
	Relative Rank	Nominal Value	Relative Rank	Nominal Value
1	5	0.0	5	0.0
2	4	0.218	4	0.216
3	2	0.431	3	0.395
4	4	0.218	4	0.216
5	5	0.0	5	0.0
6	3	0.376	2	0.425
7	1	0.660	1	0.624
8	3	0.376	2	0.425

Table 5.7: Comparisons of 'authoritativeness' values from running the two methods

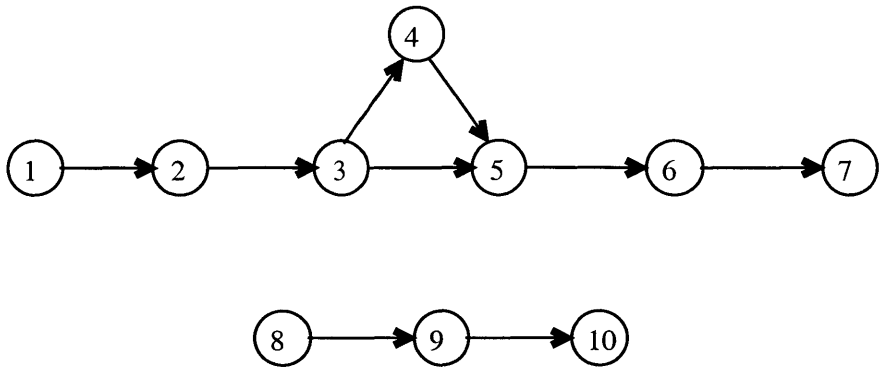


Figure 5-4: A more general case

Original values of these nodes are as shown in Table 5.5.

In the single node lifting variant of HITS, the lifter node mainly lift itself and nodes that are directly connected to it. For *authoritativeness* runs, nodes that points to the lifter node will be lifted, whereas for *hubness* runs, nodes that the lifter nodes points to will be given higher relative weighting.

The single node lifting puts itself in a much higher value nominally, and also the nodes that are connected to it. The gradient ascent variant, on the other hand, spreads the nominal relative value more evenly. This effect will be even more pronounced in the general case discussed in the following.

General case in which the gradient ascent variant of HITS yields better results

In the general case, there can be any number of 'hops' from one node to another. We will experiment with the setup shown in Figure 5-4. The nodes are divided into two cluster, one with nodes 1 through 7, and the other with nodes 8 through 10. Nodes 2, 3, 4, 6, 7, 9, 10 have the same number of in-degrees. Nodes 1, 2, 4, 5, 6, 8, 9 have the same number of out-degrees.

Here we analyze the effects of lifting node 3 towards authoritativeness and the effects of lifting node 5 towards hubness. Nodes pointed by better hubs are expected to have higher ratings, whereas nodes that points to good authorities are expected to have higher ratings.

Their relative measures ran in three iterations of the original HITS algorithm is shown

Node i	original hubness		original authoritativeness	
	Relative Rank	**Nominal Value**	**Relative Rank**	**Nominal Value**
1	3	0.0403	4	0.0
2	3	0.0403	3	0.0403
3	1	0.846	3	0.0403
4	2	0.523	2	0.523
5	3	0.0403	1	0.846
6	3	0.0403	3	0.0403
7	4	0.0	3	3.0403
8	3	0.0403	4	0.0
9	3	0.0403	3	0.0403
10	4	0.0	3	0.0403

Table 5.8: Running the general case against the *normal* HITS algorithm.

Node i	single lifting authoritativeness		gradient ascent authoritativeness	
	Relative Rank	**Nominal Value**	**Relative Rank**	**Nominal Value**
1	4	0.0	6	0.0
2	3	0.0108	3	0.11833446
3	3	0.0108	3	0.11833446
4	2	0.494	2	0.487
5	1	0.869	1	0.824
6	3	0.0108	5	0.11833443
7	3	0.0108	4	0.11833444
8	4	0.0	6	0.0
9	3	0.0108	5	0.11833443
10	4	0.0108	4	0.11833444

Table 5.9: Comparing authoritativeness given *lifter* node 5.

Node i	single lifting hubness		gradient ascent hubness	
	Relative Rank	Nominal Value	Relative Rank	Nominal Value
1	3	0.0108	5	0.0514
2	3	0.0108	4	0.149
3	1	0.869	3	0.398
4	2	0.494	1	0.702
5	3	0.0108	2	0.566
6	3	0.0108	7	0.00739
7	4	0.0	8	0.0
8	3	0.0108	6	0.0410
9	3	0.0108	7	0.00739
10	4	0.0	8	0.0

Table 5.10: Comparing hubness given *lifter* node 3.

in Table 5.8. As expected, the nodes that have similar number of in-degrees have the same rank, and nodes with the same numbers of out-degrees likewise.

Running the gradient ascent and single lifting variant of HITS, we would expect the lifter node to reposition these nodes. First, we lift node 3 - the most authoritative node - and run both variants against each other. The parameters for this test is: number of iteration is three, γ value of 0.8. Table 5.9 shows the results of this experiment.

Next, we lift node 5 - the most hub node - and run both variants. The parameters for the test is the same as before. Table 5.10 shows the results of this experiment.

The results shows that the gradient ascent HITS does yield better measures than the single lifting variant. For the authoritativeness comparison, the more relevant nodes - nodes 2, 3, 4 - are distanced away from nodes 8, 9, 10 from the other cluster. Relatively, the rank of these nodes are lifted in the gradient ascent case, as opposed to the single lifting variant in which the relative rankings of these nodes are not specifically better than before.

For the hubness comparison, the gradient ascent variant again yields better results. Nodes 4, 5, and even nodes 1, 2 are lifted in their relative rankings as compared to nodes 8, 9, 10 from the other cluster. On the other hand, the single lifting variant only lift itself and node 4, while ignoring the rest.

In addition, if we look at nodes 8 and 9 in the hubness comparison in Table 5.10, the

gradient ascent variant of HITS takes into account the fact that node 9 is a better hub than node 10, and accordingly gives node 8 a higher ranking. Single lifting, on the other hand, ignores this fact. Likewise, node 7 and 8 in the authoritativeness comparison in Table 5.9 exhibits the same phenomena.

Moreover, we observe that the single node lifting HITS suffers from 'egotism': it raises the relative nominal value of itself significantly higher than the rest. The scope of the single node variant is also limited, as evident from the results.

We conclude our discussion in comparison of the two variants of HITS.

5.2 Document level validation

After verifying the validity of the implementation, we move to the next level of tests: verifying effectiveness. Ideally, we would like to analyze the performance of the system from directly querying the web. However, it is not possible to give a robust quantitative analysis from direct querying, due to the fact that it takes an infinite amount of memory to map the state of the *world* - in this case, the Internet - into an enormous web graph. Yet on the other hand, we still want to analyze quantitatively how the system performs in practice, hence the set of tests below.

First we introduce a sample case. For a given cluster of documents, we know the number of nodes involved and how they interact with each other. All the parameters of the experiments are known. We test how the gradient ascent HITS run against the single node lifting variant.

Next we generate a set of simulations. Here we introduce the algorithms that we extended in chapter 3 and look at their performance. The justification for these set of simulations is that the experimenter has control over all these variables and can perform various analyses to produce a reliable data set, which is lacking in real time experiments.

The following sections describes each of these experiments in turn.

5.2.1 Controlled experiments

In the controlled experiment, we generate clusterings of documents. Each cluster consists of a fixed number of documents, and they are bind to a certain context. Each document has a fixed number of links. There two types of links, ones by which a document points to a document in a the same cluster, and ones by which a document points to another document in a different context. To generate a statistically reliable data set, the way each node interacts with one another is determined stochastically. To following summarizes the experiment parameters:

- *Number of clusterings or context.*

  Each generated set contains a fixed number of contexts or topics. Each node k in the set is attached to a context $c_k \in \{C_1, C_2, ..., C_M\}$ for M contexts.

- *Number of documents in a cluster: D.*

  To create a set of balanced clusters, the number of documents per cluster is fixed.

- *Number of links per document: N.*

  Again, to create a set of balanced clusters, the number of links per document is fixed. This parameter is upper bounded by the number of documents per cluster.

- *Probability that a link connects two documents in the same cluster: p*

  Define p to be Prob($l_{ij} \mid c_i = c_j$) and $1 - p$ to be Prob($l_{ij} \mid c_i \neq c_j$), where l_{ij} is a link from node i to node j. In other words, p denotes the probability of a document pointing to another document of the same context, and $1 - p$ denotes the probability of a document pointing to another document with a different context.

Figure 5-5 illustrates an example of a possible clusterings of documents - represented as nodes - as well as their interaction. Since links are probabilistic, and for each link l_{ij} pointing from node i, node j is chosen at random, each generated nodes-set has varying graph structures.

Now that we have this platform, we want to create test harnesses based on our algorithms. In particular, we would like to see the effects of lifting in both the gradient ascent variant

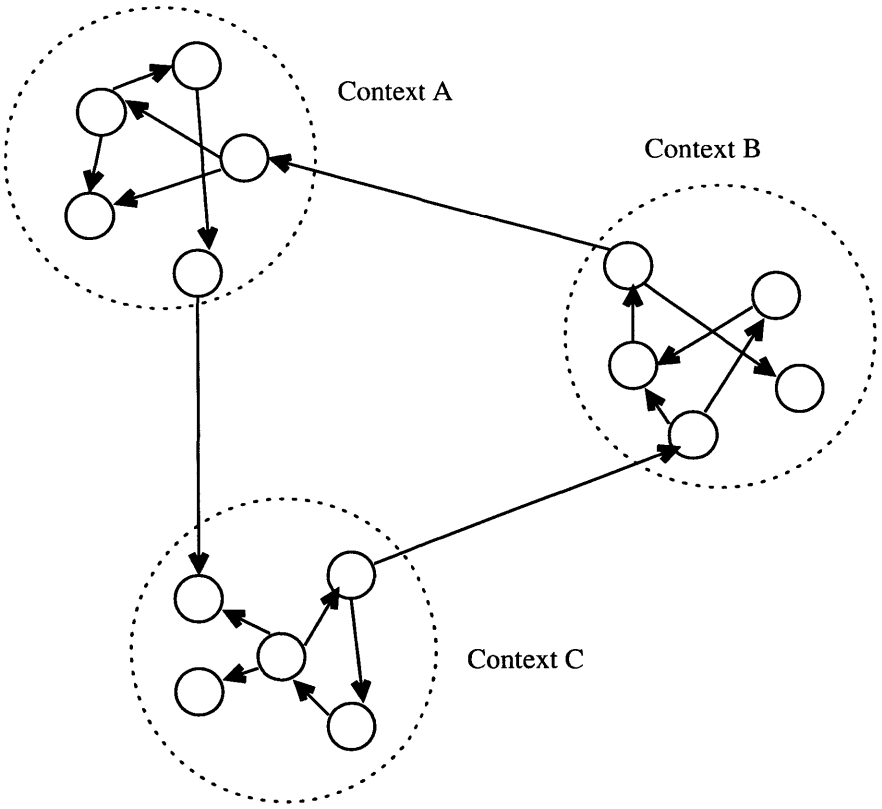


Figure 5-5: View of clusters of documents tagged to different context cross referencing each other

Parameter	Variable	Value
Number of categories	M	3
Number of documents per category	D	10
Number of links per document	N	8
Prob($l_{ij} \mid c_i = c_j$)	p	0.7
Prob($l_{ij} \mid c_i \neq c_j$)	$1-p$	0.3
Weighting factor	γ	0.5
Number of iterations	-	4
Hub or Authority	-	Authority

Table 5.11: Parameters for sample data experiment

and the single lifting variant of HITS towards the relative ranking of documents in such a structure.

Sample Data

As a sample data, we would like to analyze such effects upon a particular instantiation of a nodes-set. The experiment parameters are summarized in Table 5.11. The last three lines of Table 5.11 are for performing the gradient ascent and single lifting HITS. Preceding them are parameters for generating the nodes-set.

Running our randomized platform above, we have a graph structure similar to that shown in Figure 5-5. A simple program routine then computes the adjacency matrix for the graph structure and runs the *normal* HITS, gradient-ascend HITS, and single-node-lifting HITS on the same structure. The output of the computation is shown in Table 5.12 and Table 5.13

The results are promising, but we would like to have a performance metric to quantify these results. From a quick glance, both algorithms manage to lift the rankings of relevant nodes. Looking at nodes one through ten in Table 5.13, both variant of the HITS algorithm increase these nodes' relative rankings. Analyzing the data qualitatively, we see that the gradient ascent HITS performs better, because the relative rankings of relevant nodes increased more *on average.*

Essentially, a good metric should be able to capture this fact. We want to measure performance on two criteria: how much the average relative rankings of relevant nodes are

Node i	Nominal Value		
	Plain HITS	**Single node lifting**	**Gradient ascent**
first context: nodes numbered 1 - 10			
1	0.1767611	0.18256958	0.2756714
2	0.16509369	0.16886494	0.2672982
3	0.17553343	0.18224487	0.27733904
4	0.17491709	0.18153954	0.27657786
5	0.19628778	0.20176086	0.14729327
6	0.18203038	0.1970692	0.14112088
7	0.16459058	0.16878106	0.26965842
8	0.16997905	0.1770913	0.13567474
9	0.15326434	0.1577038	0.12627058
10	0.2105928	0.21509345	0.15392815
second context: nodes numbered 11 - 20			
11	0.16227429	0.16108964	0.124522574
12	0.19047816	0.18775927	0.13510692
13	0.17577828	0.17407782	0.13023962
14	0.19039173	0.1861986	0.27722722
15	0.19387662	0.1945795	0.13858229
16	0.18631834	0.1849758	0.13572805
17	0.16930962	0.16777395	0.12658183
18	0.17969474	0.17626317	0.13031766
19	0.19547182	0.1882115	0.2769008
20	0.19661158	0.19148134	0.13780752
third context: nodes numbered 21 - 30			
21	0.1958135	0.19939977	0.14224866
22	0.18640624	0.1850691	0.13586761
23	0.17655301	0.17519887	0.13064061
24	0.17332835	0.1686358	0.26581448
25	0.19042253	0.18253064	0.13417171
26	0.18613556	0.18068835	0.13214676
27	0.17442444	0.17257237	0.12835792
28	0.18888198	0.18372989	0.13488168
29	0.18630393	0.1810041	0.1317574
30	0.19672135	0.19101444	0.13778776

Table 5.12: Nominal values of the nodes, lifting node 6.

Node i	Relative Rankings		
	Plain HITS	**Single node lifting**	**Gradient ascent**
first context: nodes numbered 1 - 10			
1	18	14	5
2	27	25	7
3	21	16	1
4	22	17	4
5	4	2	10
6	16	4	12
7	28	26	6
8	25	20	18
9	30	30	29
10	1	1	9
second context: nodes numbered 11 - 20			
11	29	29	30
12	8	9	19
13	20	23	26
14	10	10	2
15	7	5	13
16	13	12	17
17	26	28	28
18	17	21	25
19	6	8	3
20	3	6	14
third context: nodes numbered 21 - 30			
21	5	3	11
22	12	11	16
23	19	22	24
24	24	27	8
25	9	15	21
26	15	19	22
27	23	24	27
28	11	13	20
29	14	18	23
30	2	7	15

Table 5.13: Relative rankings of the nodes, lifting node 6

Metric	Single node lifting	Gradient ascent
L	3.7	9.1
S	1.85	4.55

Table 5.14: Metric for this sample data

increased by the technique, and how much of the average relative rankings of irrelevant nodes are decreased. By relevant nodes, we mean nodes that has the same context as the lifted node. To quantify this fact, we use a performance metric described as follows:

- Given D number of documents per category, and node i with rank R_i and context $c_i = c_{lift-node}$, we measure the average 'lift' L to be:

$$L = \frac{1}{D} \cdot \sum_{\forall i \,|\, c_i = c_{(lift-node)}} (R_{i_{old}} - R_{i_{new}})$$

- The average 'suppress' S of all nodes j with context $c_j \neq c_{lift-node}$ is defined to be:

$$S = -\frac{1}{D \cdot (M-1)} \cdot \sum_{\forall j \,|\, c_j \neq c_{(lift-node)}} (R_{i_{old}} - R_{i_{new}})$$

Note that L and S are not independent metrics. For example, if all the rankings stay the same, that is, if $L = 0$ then there is very high probability that S is also equal to zero. If most of the rankings change - L is relatively high - then S is very likely to be relatively high as well.

Since in this example, node six is chosen, we are interested in the lift values of nodes one through ten, and the suppress value of nodes eleven to thirty. After some calculation, we find the values in Table 5.14. On *both* measures, the metric shows gradient ascent is superior to single lifting.

Using these performance metrics, we can perform several runs and accumulate data for statistical analysis. The results are discussed in the next section.

Simulation Results

We are interested in the relationship between probability of a node pointing to another node of the same context p to the performance metrics L and S. Using the parameters described in Table 5.11, we vary the value of p from 0 to 1, with increments of 0.1. At each interval, we calculate the L and S metrics for five different trials. With higher p, we expect the number of documents lifted or suppressed to be larger. The relationship should be somewhat linear. In the following, we will run two sets, one for authoritativeness measure, another for hubness measure.

At this point, we introduce the two variants of gradient ascent that we extend from chapter 3. We refer to them as gradient ascent combination one, or gradient ascent$^+$, and gradient ascent combination two, or gradient ascent^{++}.

As a quick review, the first combination takes the sum of contributions from all nodes i in the document set *only if* the set of positive contributions is larger than the set of average negative contribution. In the second combination, we loosen this restriction, with the justification that even though the total sum is negative, node of interest j should still have a *relative* higher value. Then we alternate between this calculation and the normal gradient ascent at each iteration.

The discussion is broken down into the two subsections. First we distinguish between *authority* and *hub* measures. Second we separate the L metric from the S metric.

Authority simulation results

Running the simulation as described above, we have the set of points as listed in Table A.1 through Table A.12 in appendix A.

First we analyze the results for the L metric. We graph the set of points to better visualize the results. Figure 5-6 plots this graph in its point representation and its least square estimate for each cluster of data. In this simulation, the line that represents single node lifting is very close to the line that represents gradient ascent$^+$. The line that represents gradient ascent is very close to the line that represents gradient ascent^{++}. The shape of all the lines, however, are linearly increasing, which is what we should expect.

In chapter 3, we discuss that both extension of the gradient ascent method should perform better. However, the graph merely shows that gradient ascent^{+} performs at least as well as single node lifting, and gradient ascent^{++} performs at least as well as the original gradient ascent. We had expected both variants of gradient ascent should converge faster to a solution that differentiates between lifted and suppressed nodes.

What we can infer from this graph is the difference between the performance of gradient ascent in comparison to the single node lifting variant of HITS. Single node lifting at *most* lifts as much documents than the gradient ascent variants. With these tentative results, we look at the next graph.

Figure 5-7 plots the data points and its least square estimation for the S metric. Here the results are more encouraging. We see that the gradient ascent^{++} does confirm our theoretical expectations. However, gradient ascent^{+} performs in between single node lifting and the original gradient ascent.

This substantial difference suggests that both extensions of gradient ascent works better in *suppressing* irrelevant documents than *lifting* relevant documents. With this fact in mind, we move on to the next analysis.

5.2.2 Hub simulation results

This section analyzes the results from running the simulation to calculate *hub* measure. As before, the raw data and least square estimate for the L metric is calculated. Figure 5-8 plots these values.

As in the previous L metric analysis, the original gradient ascent is very closely connected to gradient ascent^{++} in that they are practically collinear. The other gradient ascent variant, however, fare better than in the previous simulation, though it is still distinctly below the original gradient ascent.

We note that the metric values are negative in this simulation for p less than 0.4 The reason is that for low p, the lifted document will be linked to many irrelevant documents. The lifted hub will pull up these documents along with it.

The S metric analysis, however, shows interesting results. In this simulation, as depicted

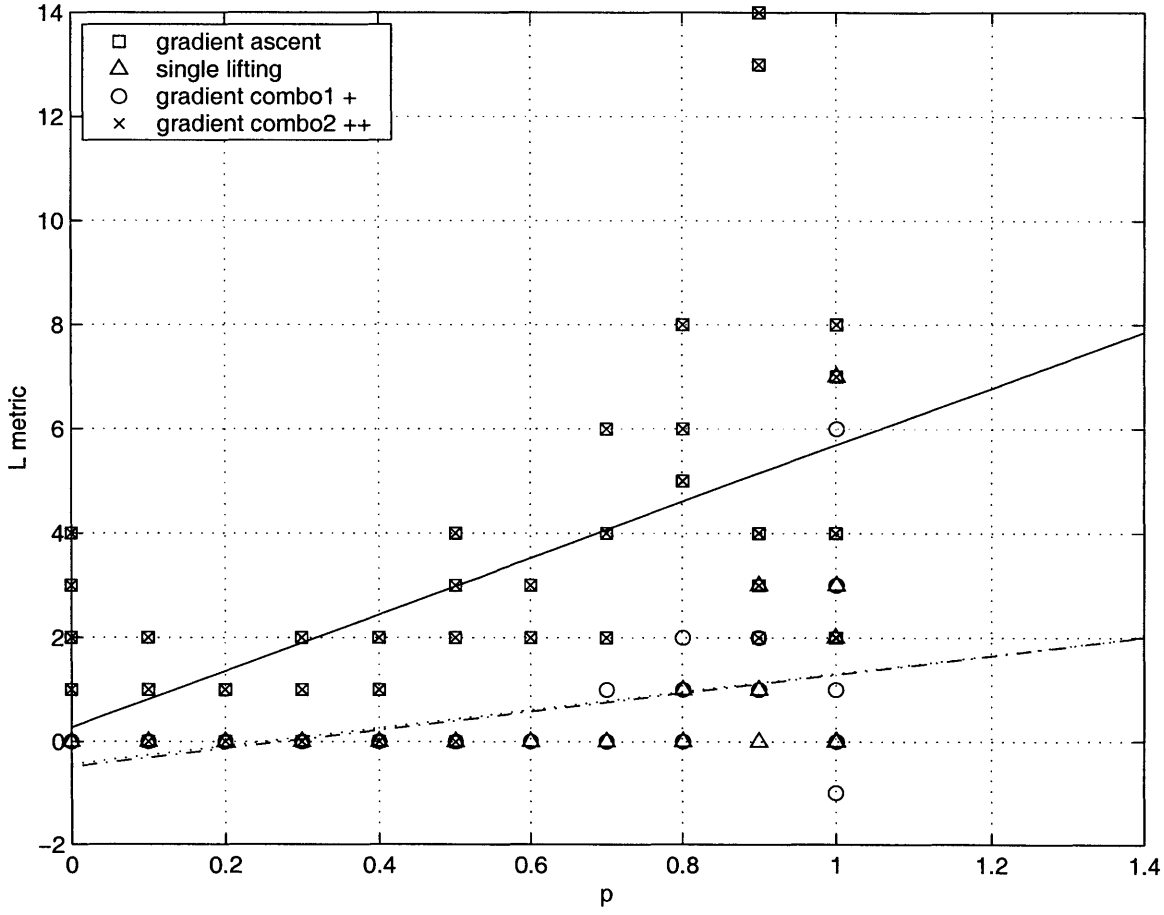


Figure 5-6: Data points for authoritativeness measure using metric L. The top line represents the least square estimate for gradient ascent and gradient ascent^{++}. The bottom line represents the estimate for gradient ascent^{+} and single node lifting, the other two merged into one.

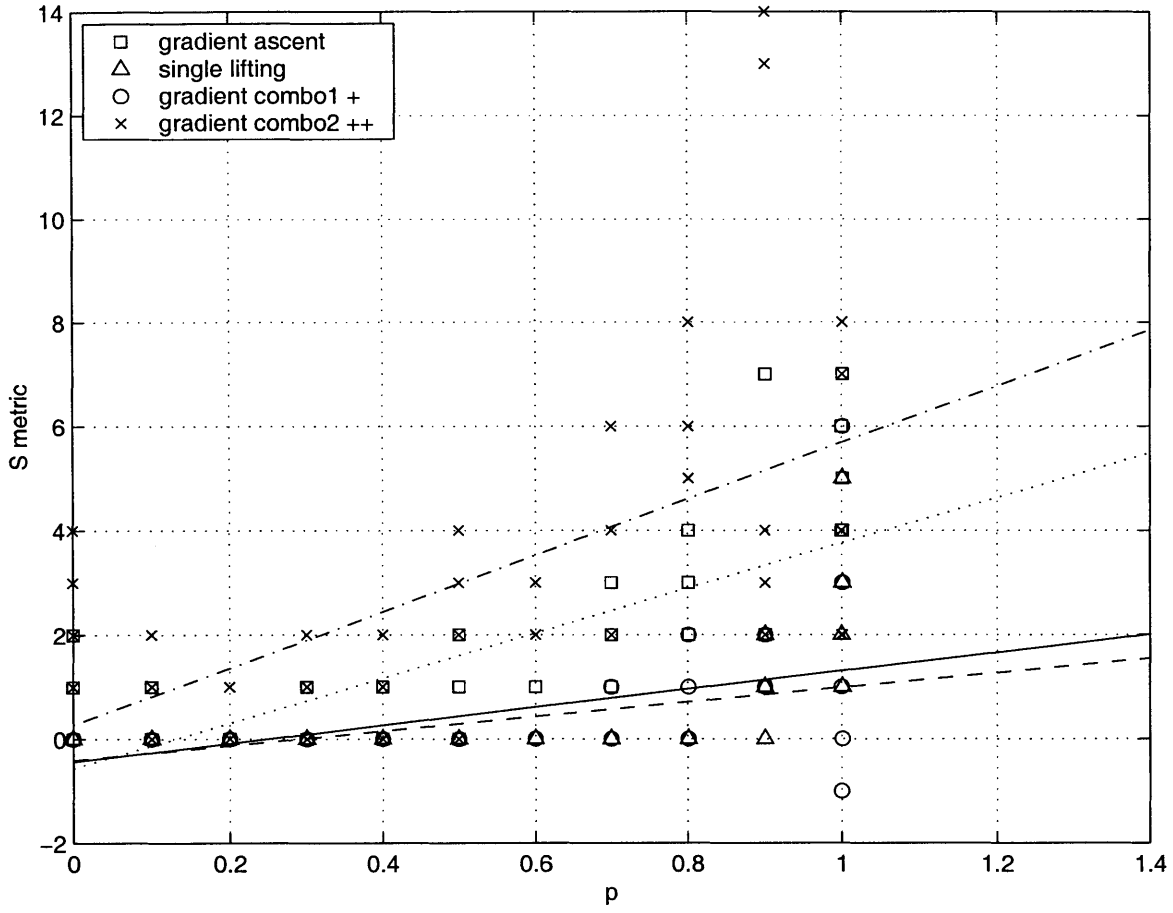


Figure 5-7: Data points for authoritativeness measure using metric S. The dashed line at the top is the least square estimate for gradient ascent^{++}. The dotted line represents normal gradient ascent. The solid line represents gradient ascent $^{+}$. Lastly, the bottom line represents single node lifting.

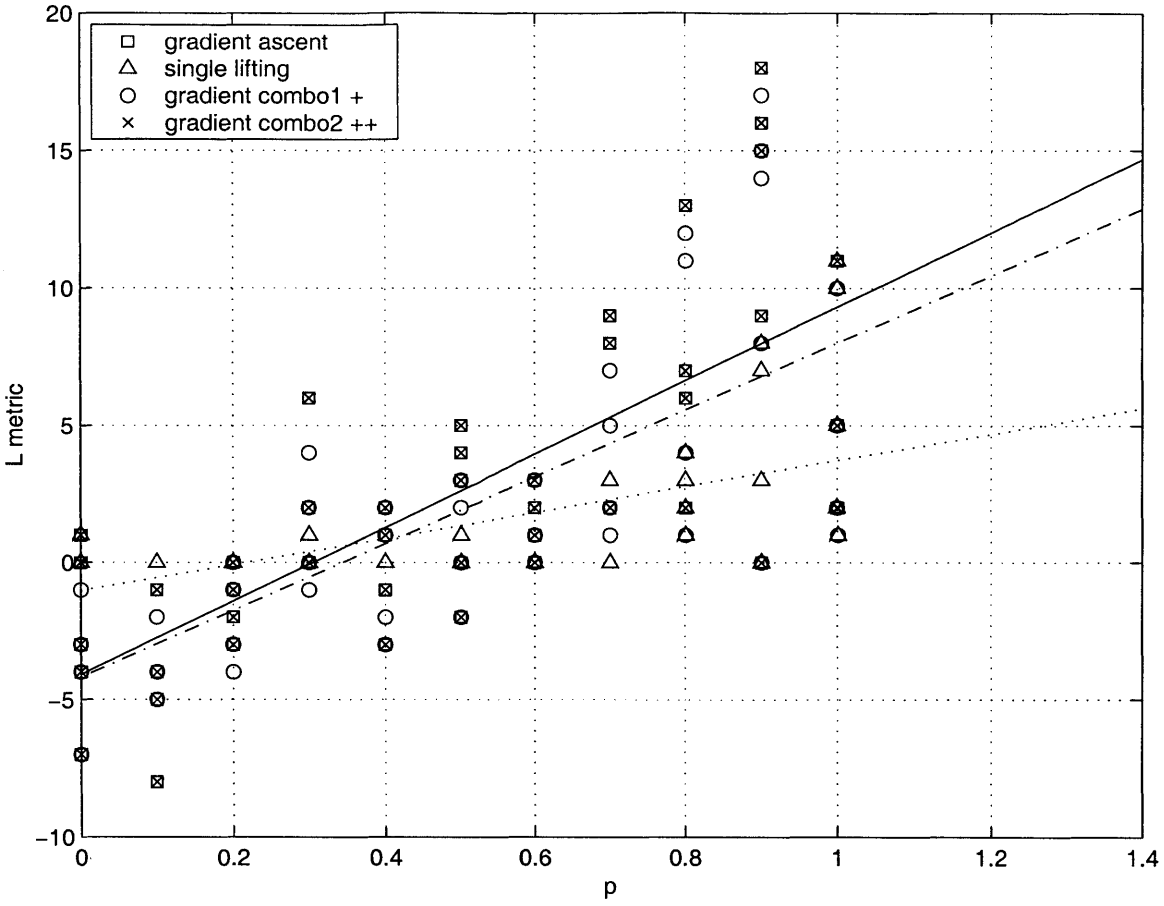


Figure 5-8: Data points for hubness measure using metric L. The dotted line is the least square estimate for single node lifting, the dashed line for gradient ascent combo 1, the other two merged into one.

in Figure 5-9, *both* variants of the gradient ascent perform better than the original gradient ascent and the single node lifting variant. Again, this follows the trend that we see in our previous S metric analysis: *both extensions of gradient ascent suppress irrelevant document better than they lift relevant documents.*

Figure 5-9 validates our expectation that combining both gradient ascent and descent results in a *faster rate of change.* The slope is steeper for both gradient ascent^{+} and gradient ascent^{++} than it is for the other two techniques.

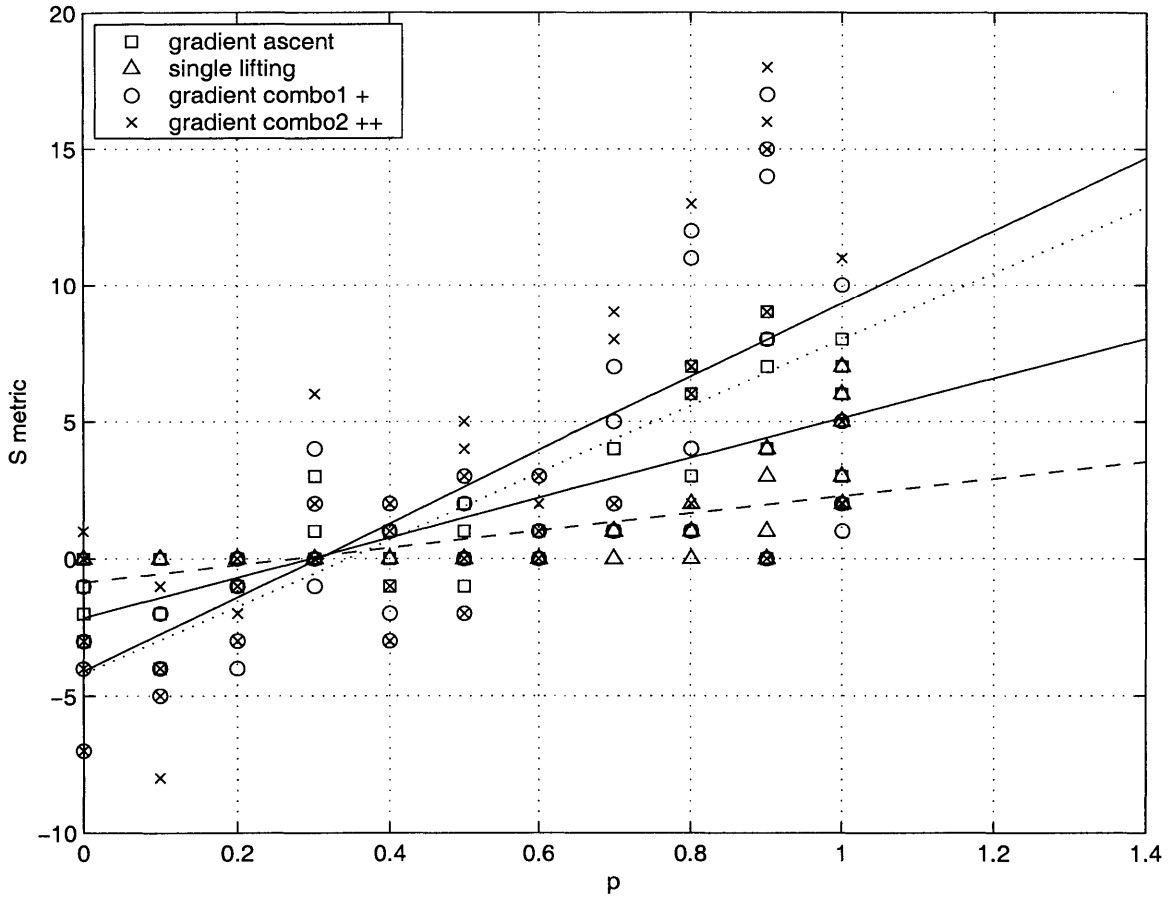


Figure 5-9: Data points for hubness measure using metric S. The top solid line is the least square estimate of gradient ascent^{++}. The dotted line is the estimate for gradient ascent^{+}. The second from the bottom solid line is for normal gradient ascent. The dashed line is for single node lifting.

5.2.3 Summary of analysis

Though the results are somewhat mixed, we can conclude the following results:

- The first combination of gradient ascent, or gradient ascent^{+} *at least* performs as well as single node lifting.
- The second combination of gradient ascent, or gradient ascent^{++} *at least* performs as well than the original gradient ascent.
- Both combinations does better at *suppressing* irrelevant documents rather than *lifting* relevant documents.

In both single node lifting and gradient ascent variants, we only lift the authority or hub measure for a certain node. In our extension, we do this lifting while supressing irrelevant nodes. Therefore, it makes sense that both combinations performs better under the S metric as compared to the L metric.

The reason why our first combination does not work as well may be that the approach is 'half hearted' in that instead of totally maximizing the contribution of preferred document j and minimizing the contribution of other documents $l \neq j$, we only do so when the *nominal* value of j at least increases. The result is a gradient step size that does neither. Our second approach is more effective because we only concern ourselves with the *relative* value of document j in comparison to other nodes. In the end, this does indeed produce better results.

5.2.4 Real-time analysis

The purpose of the set of validation and testing above is to complement the system implementation, since it is not possible to run controlled experiments in this real time system. However, we did test the *Persona* prototype to check if it produces reasonable results. We tried using several queries, namely: machine learning and virus. For the machine learning query, we indicate that we like a page that relates machine learning to games, and indicate all other unrelated pages as negative. After submitting our feedback, *Persona* returns seven

out of ten pages relating to machine learning, as opposed to one out of ten pages in the previous case. As for the virus query, we got back two sets of pages, one pointing to health related virus, and another to computer related virus. We indicate as positive all the pages that are health related, and indicate as negative all the pages that are computer related. *Persona* returns a set of ten pages all relating to health. We explained in chapter 4 how our method is different than the 'click to find similar pages' that we find in other search engines.

However, the prototype as it stands needs much improvement, particularly in terms of speed. Right now, for every page, it will open all the URL's in real time, causing the system response to be very slow. We need to add a caching and a multi threading model to hide latency. The filtering process itself is relatively fast; the system can process a few hundred nodes in less than one second.

A snapshot of the *Persona* user interface is shown in Figure 5-10. The next chapter concludes our discussion.

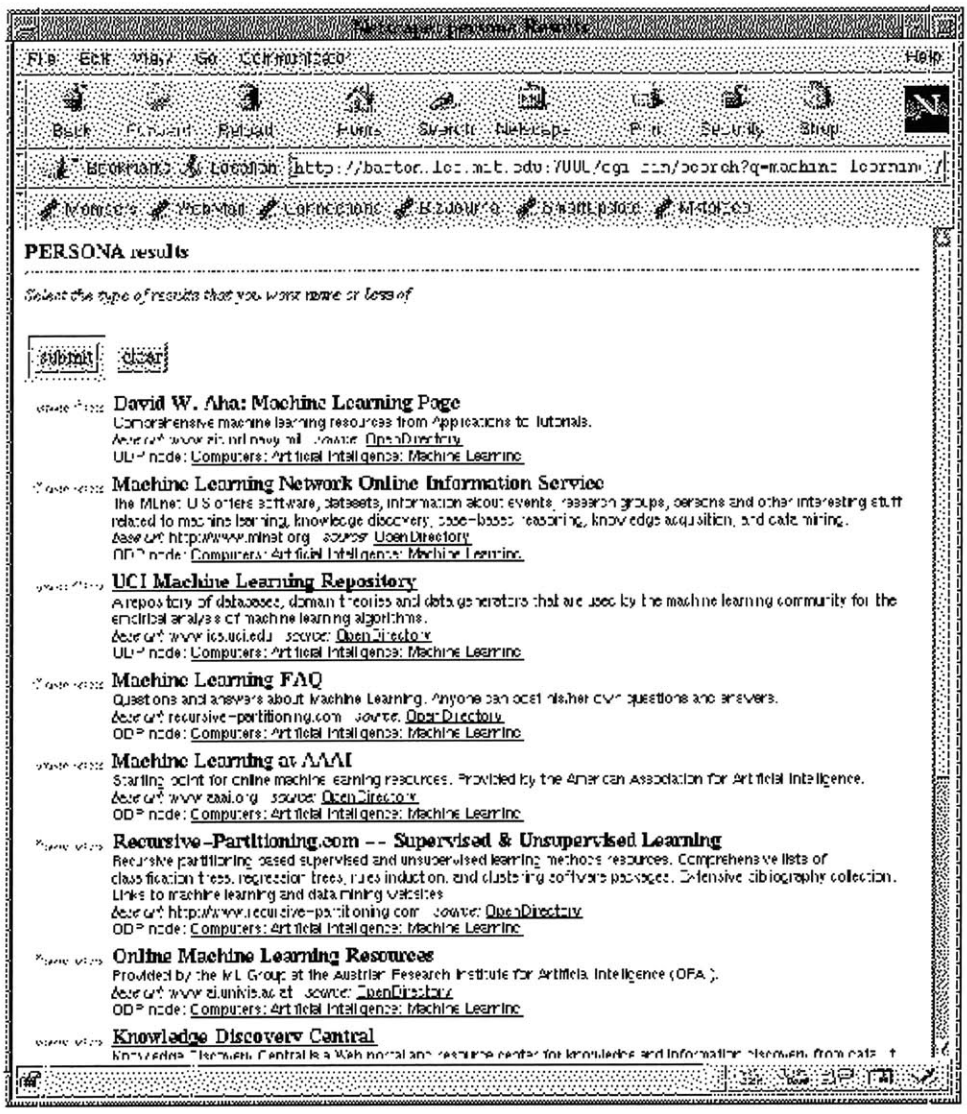


Figure 5-10: Snapshot of the *Persona* system

Chapter 6

Conclusion

6.1 Summary

In this thesis, we perform analyses of several methodologies in graph based search algorithms and extend existing theory. We explore the mathematical model to build our intuition regarding these theories. In addition, we develop a robust, scalable user model using a taxonomy structure as provided by the *Open Directory Project*. Our goal is to create a system that better personalizes users' web experience. In hoping to achieve this goal, we implemented *Persona*, a real time personalization module that wraps around an existing search engine.

Our extension of current theory is derived from the gradient ascent variant of HITS. We experiment with two possible combinations of gradient ascent. From running sets of simulations, we verified that gradient ascent is superior to single node lifting. One of our proposed combination managed to perform at least as well as the gradient ascent variant. We come up with robust quantitative results regarding the performance of these techniques.

We build a user model that improves upon current existing methods. Though it is not feasible to test the effectiveness of our user profiling technique, several tests with the *Persona* system has shown positive results. Moreover, recent models has been developed to perform static matching among user profiles to determine similarities between users [24]. We find this very encouraging, and may extend further work in this direction.

6.2 Future work

There is plenty of room for future work. Our current prototype needs much improvement in terms of speed and scalability. One immediate improvement would be to implement a threading model and a caching model. The *Persona* system is equipped to handle multiple users, but the current prototype lacks a login manager that can run an authentication mechanism. Fortunately, there are many implementations readily available.

In terms of theory, we can extend this work even further. Our aggregation method to incorporate multiple documents of interests are only satisfactory. We would like to create a more robust model to generalize over a set of documents. The user model itself is also open to improvements. Possible extension may be to do a dynamic update of trees, such that if a node is updated, its ancestors will be updated, the magnitude of which depends on how many edges lie between the node and the ancestor.

There are unlimited possibilities for further improvements. Our implementation uses open industry standards, and the code is fully documented in *Javadoc*. We design the system keeping in mind that it should be open ended and open to future upgrades. We leave it to the reader's imagination to proceed to the next stage.

Appendix A

Simulation Data

The following tables captures the raw data from our simulations. Each are labelled accordingly.

p	L value data									
	Single node lifting					Gradient ascent				
	#1	#2	#3	#4	#5	#1	#2	#3	#4	#5
0.0	0.0	0.0	0.0	0.0	0.0	3.0	1.0	4.0	2.0	2.0
0.1	0.0	0.0	0.0	0.0	0.0	1.0	1.0	2.0	0.0	1.0
0.2	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	1.0	1.0
0.3	0.0	0.0	0.0	0.0	0.0	2.0	2.0	1.0	1.0	0.0
0.4	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	1.0
0.5	0.0	0.0	0.0	0.0	0.0	3.0	0.0	2.0	4.0	2.0
0.6	0.0	0.0	0.0	0.0	0.0	3.0	2.0	2.0	3.0	3.0
0.7	0.0	0.0	0.0	0.0	0.0	6.0	6.0	2.0	2.0	4.0
0.8	1.0	0.0	0.0	1.0	0.0	8.0	6.0	6.0	6.0	5.0
0.9	0.0	3.0	1.0	1.0	3.0	14.0	13.0	3.0	2.0	4.0
1.0	3.0	7.0	0.0	2.0	0.0	7.0	8.0	2.0	4.0	2.0

Table A.1: L data over several trials using authoritativeness measure

p	L value data									
	Gradient ascent^{+}					Gradient ascent^{++}				
	#1	#2	#3	#4	#5	#1	#2	#3	#4	#5
0.0	3.0	1.0	4.0	2.0	2.0	0.0	0.0	0.0	0.0	0.0
0.1	1.0	1.0	2.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
0.2	1.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0
0.3	2.0	2.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
0.4	0.0	2.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
0.5	3.0	0.0	2.0	4.0	2.0	0.0	0.0	0.0	0.0	0.0
0.6	3.0	2.0	2.0	3.0	3.0	0.0	0.0	0.0	0.0	0.0
0.7	6.0	6.0	2.0	2.0	4.0	1.0	1.0	0.0	0.0	0.0
0.8	8.0	6.0	6.0	6.0	5.0	1.0	0.0	1.0	2.0	1.0
0.9	14.0	13.0	3.0	2.0	4.0	2.0	2.0	1.0	1.0	2.0
1.0	7.0	8.0	2.0	4.0	2.0	3.0	6.0	0.0	1.0	-1.0

Table A.2: L data over several trials using authoritativeness measure with two other variants of gradient ascent

p	S value data									
	Single node lifting					**Gradient ascent**				
	#1	#2	#3	#4	#5	#1	#2	#3	#4	#5
0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	2.0	1.0	1.0
0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.3	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0
0.4	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
0.5	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	2.0	1.0
0.6	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0
0.7	0.0	0.0	0.0	0.0	0.0	3.0	3.0	1.0	1.0	2.0
0.8	0.0	0.0	0.0	0.0	0.0	4.0	3.0	3.0	3.0	2.0
0.9	0.0	2.0	0.0	0.0	1.0	7.0	7.0	1.0	1.0	2.0
1.0	5.0	1.0	3.0	2.0	2.0	7.0	4.0	5.0	6.0	4.0

Table A.3: S data over several trials using authoritativeness measure

p	S value data									
	Gradient ascent^{+}					**Gradient ascent^{++}**				
	#1	#2	#3	#4	#5	#1	#2	#3	#4	#5
0.0	1.0	0.0	2.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0
0.1	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.3	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.4	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.5	1.0	0.0	1.0	2.0	1.0	0.0	0.0	0.0	0.0	0.0
0.6	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0
0.7	3.0	3.0	1.0	1.0	2.0	0.0	0.0	0.0	0.0	0.0
0.8	4.0	3.0	3.0	3.0	2.0	0.0	0.0	0.0	1.0	0.0
0.9	7.0	7.0	1.0	1.0	2.0	1.0	2.0	0.0	0.0	1.0
1.0	7.0	4.0	5.0	6.0	4.0	3.0	2.0	3.0	3.0	3.0

Table A.4: S data over several trials using authoritativeness measure with two variants of gradient ascent

p	L value data							
	Single node lifting		**Gradient ascent**		**Gradient ascent$^+$**		**Gradient ascent^{++}**	
	μ	σ^2	μ	σ^2	μ	σ^2	μ	σ^2
0.0	0.0	0.0	2.4	1.04	0.0	0.0	2.4	1.04
0.1	0.0	0.0	1.0	0.4	0.0	0.0	1.0	0.4
0.2	0.0	0.0	0.8	0.16	0.0	0.0	0.8	0.16
0.3	0.0	0.0	1.2	0.56	0.0	0.0	1.2	0.56
0.4	0.0	0.0	0.6	0.64	0.0	0.0	0.6	0.64
0.5	0.0	0.0	2.2	1.76	0.0	0.0	2.2	1.76
0.6	0.0	0.0	2.6	0.2401	0.0	0.0	2.6	0.24
0.7	0.0	0.0	4.0	3.2	0.4	0.2401	4.0	3.2
0.8	0.4	0.2401	6.2	0.96	1.0	0.4	6.2	0.96
0.9	1.6	1.44	7.2	26.96	1.6	0.24	7.2	26.91
1.0	2.4	6.64	4.6	6.24	1.8	6.16	4.6	6.24

Table A.5: Statistics for L metrics using authoritativeness measure

p	S value data							
	Single node lifting		**Gradient ascent**		**Gradient ascent$^+$**		**Gradient ascent^{++}**	
	μ	σ^2	μ	σ^2	μ	σ^2	μ	σ^2
0.0	0.0	0.0	1.0	0.4	0.0	0.0	1.0	0.4
0.1	0.0	0.0	0.2	0.1601	0.0	0.0	0.2	0.1601
0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.3	0.0	0.0	0.4	0.2401	0.0	0.0	0.4	0.2401
0.4	0.0	0.0	0.2	0.1603	0.0	0.0	0.2	0.1603
0.5	0.0	0.0	1.0	0.4	0.0	0.0	1.0	0.4
0.6	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0
0.7	0.0	0.0	2.0	0.8	0.0	0.0	2.0	0.8
0.8	0.0	0.0	3.0	0.4	0.2	0.1601	3.0	0.4
0.9	0.6	0.64	3.6	7.84	0.8	0.5606	3.6	7.84
1.0	2.6	1.84	5.2	1.36	2.8	0.16	5.2	1.36

Table A.6: Statistics for S metrics using authoritativeness measure

p	L value data									
	Single node lifting					Gradient ascent				
	#1	#2	#3	#4	#5	#1	#2	#3	#4	#5
0.0	0.0	1.0	1.0	1.0	1.0	1.0	-7.0	-4.0	-3.0	0.0
0.1	0.0	0.0	0.0	0.0	0.0	-8.0	-5.0	-1.0	-5.0	-4.0
0.2	0.0	0.0	0.0	0.0	0.0	0.0	-2.0	-3.0	-1.0	-1.0
0.3	0.0	0.0	1.0	0.0	0.0	2.0	6.0	0.0	0.0	0.0
0.4	0.0	0.0	0.0	0.0	0.0	2.0	-3.0	2.0	-1.0	1.0
0.5	0.0	1.0	0.0	0.0	0.0	3.0	4.0	0.0	5.0	-2.0
0.6	0.0	0.0	0.0	0.0	0.0	3.0	1.0	3.0	0.0	2.0
0.7	0.0	0.0	0.0	3.0	0.0	2.0	8.0	8.0	9.0	2.0
0.8	3.0	4.0	1.0	1.0	2.0	13.0	13.0	2.0	7.0	6.0
0.9	7.0	0.0	7.0	8.0	3.0	15.0	0.0	9.0	18.0	16.0
1.0	1.0	11.0	10.0	2.0	5.0	2.0	11.0	11.0	2.0	5.0

Table A.7: L data over several trials using hubness measure

p	L value data									
	Gradient ascent^{+}					Gradient ascent^{++}				
	#1	#2	#3	#4	#5	#1	#2	#3	#4	#5
0.0	1.0	-7.0	-4.0	-3.0	0.0	-1.0	-7.0	-4.0	-3.0	-1.0
0.1	-8.0	-5.0	-1.0	-5.0	-4.0	-5.0	-4.0	-2.0	-4.0	-4.0
0.2	0.0	-2.0	-3.0	-1.0	-1.0	0.0	-3.0	-4.0	0.0	-1.0
0.3	2.0	6.0	0.0	0.0	0.0	2.0	4.0	-1.0	0.0	-1.0
0.4	2.0	-3.0	2.0	-1.0	1.0	1.0	-3.0	2.0	-2.0	1.0
0.5	3.0	4.0	0.0	5.0	-2.0	2.0	2.0	0.0	3.0	-2.0
0.6	3.0	1.0	3.0	0.0	2.0	3.0	1.0	1.0	0.0	1.0
0.7	2.0	8.0	8.0	9.0	2.0	2.0	5.0	5.0	7.0	1.0
0.8	13.0	13.0	2.0	7.0	6.0	12.0	11.0	1.0	4.0	4.0
0.9	15.0	0.0	9.0	18.0	16.0	15.0	0.0	8.0	17.0	14.0
1.0	2.0	11.0	11.0	2.0	5.0	1.0	10.0	10.0	2.0	5.0

Table A.8: L data over several trials using hubness measure

p	S value data									
	Single node lifting					Gradient ascent				
	#1	#2	#3	#4	#5	#1	#2	#3	#4	#5
0.0	0.0	0.0	0.0	0.0	0.0	0.0	-3.0	-2.0	-1.0	0.0
0.1	0.0	0.0	0.0	0.0	0.0	-4.0	-2.0	0.0	-2.0	-2.0
0.2	0.0	0.0	0.0	0.0	0.0	0.0	-1.0	-1.0	0.0	0.0
0.3	0.0	0.0	0.0	0.0	0.0	1.0	3.0	0.0	0.0	0.0
0.4	0.0	0.0	0.0	0.0	0.0	1.0	-1.0	1.0	0.0	0.0
0.5	0.0	0.0	0.0	0.0	0.0	1.0	2.0	0.0	2.0	-1.0
0.6	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0
0.7	0.0	0.0	0.0	1.0	0.0	1.0	4.0	4.0	4.0	1.0
0.8	1.0	2.0	0.0	0.0	1.0	6.0	7.0	1.0	3.0	3.0
0.9	3.0	0.0	3.0	4.0	1.0	7.0	0.0	4.0	9.0	8.0
1.0	2.0	6.0	7.0	3.0	5.0	2.0	7.0	8.0	3.0	6.0

Table A.9: S data over several trials using hubness measure

p	S value data									
	Gradient ascent^{+}					Gradient ascent^{++}				
	#1	#2	#3	#4	#5	#1	#2	#3	#4	#5
0.0	0.0	-3.0	-2.0	-1.0	0.0	0.0	-3.0	-2.0	-1.0	0.0
0.1	-4.0	-2.0	0.0	-2.0	-2.0	-2.0	-2.0	-1.0	-2.0	-2.0
0.2	0.0	-1.0	-1.0	0.0	0.0	0.0	-1.0	-2.0	0.0	0.0
0.3	1.0	3.0	0.0	0.0	0.0	1.0	2.0	0.0	0.0	0.0
0.4	1.0	-1.0	1.0	0.0	0.0	0.0	-1.0	1.0	-1.0	0.0
0.5	1.0	2.0	0.0	2.0	-1.0	1.0	1.0	0.0	1.0	-1.0
0.6	1.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0
0.7	1.0	4.0	4.0	4.0	1.0	1.0	2.0	2.0	3.0	0.0
0.8	6.0	7.0	1.0	3.0	3.0	6.0	6.0	0.0	2.0	2.0
0.9	7.0	0.0	4.0	9.0	8.0	7.0	0.0	4.0	8.0	7.0
1.0	2.0	7.0	8.0	3.0	6.0	2.0	8.0	8.0	2.0	6.0

Table A.10: S data over several trials using hubness measure

p	L value data							
	Single node lifting		**Gradient ascent**		**Gradient ascent$^+$**		**Gradient ascent^{++}**	
	μ	σ^2	μ	σ^2	μ	σ^2	μ	σ^2
0.0	0.8	0.16	-2.6	8.24	-3.2	4.96	-2.6	8.24
0.1	0.0	0.0	-4.6	5.04	-3.8	0.96	-4.6	5.04
0.2	0.0	0.0	-1.4	1.04	-1.6	2.64	-1.4	1.04
0.3	0.2	0.16	1.6	5.44	0.8	3.76	1.6	5.44
0.4	0.0	0.0	0.2	3.76	-0.2	3.76	0.2	3.7
0.5	0.2	0.16	2.0	6.8	1.0	3.2	2.0	6.8
0.6	0.0	0.0	1.8	1.36	1.2	0.96	1.8	1.36
0.7	0.6	1.44	5.8	9.76	4.0	4.8	5.8	9.76
0.8	2.2	1.36	8.2	18.16	6.4	18.6	8.2	18.16
0.9	5.0	9.2	11.6	42.64	10.8	38.16	11.6	42.64
1.0	5.8	16.56	6.2	16.6	5.6	14.64	6.2	16.5

Table A.11: Statistics for L metrics using hubness measure

p	S value data							
	Single node lifting		**Gradient ascent**		**Gradient ascent$^+$**		**Gradient ascent^{++}**	
	μ	σ^2	μ	σ^2	μ	σ^2	μ	σ^2
0.0	0.0	0.0	-1.2	1.36	-1.2	1.36	-1.2	1.36
0.1	0.0	0.0	-2.0	1.6	-1.8	0.16	-2.0	1.6
0.2	0.0	0.0	-0.4	0.24	-0.6	0.64	-0.4	0.24
0.3	0.0	0.0	0.8	1.36	0.6	0.6405	0.8	1.36
0.4	0.0	0.0	0.2	0.56	-0.2	0.56	0.2	0.56
0.5	0.0	0.0	0.8	1.36	0.4	0.64	0.8	1.36
0.6	0.0	0.0	0.6	0.24	0.2	0.16	0.6	0.24
0.7	0.2	0.16	2.8	2.16	1.6	1.04	2.8	2.16
0.8	0.8	0.56	4.0	4.8	3.2	5.76	4.0	4.8
0.9	2.2	2.16	5.6	10.6	5.2	8.56	5.6	10.6
1.0	4.6	3.44	5.2	5.36	5.2	7.36	5.2	5.36

Table A.12: Statistics for S metrics using hubness measure

Appendix B

Module Dependency Diagram for the *Persona* back end

The elements of the system are as follows:

1. *Daemon.* Listens for input from a port, and outputs to another port. The format of 'legal' input is specified in an XML Document Type Definition (DTD), and thus uses XML as an interface.

2. *ResultTable.* The input stream from the daemon is transformed into a ResultTable object. A ResultTable encapsulates the higher level information such as user name, search context. A ResultTable consists of several ResultEntry objects.

3. *ResultEntry.* A ResultEntry encapsulates link level information, such as rank, source, bias, etc.

4. *XMLExtractor* A helper class whose function is to parse incoming XML.

5. *User Profile.* Captures the user's interest. Implemented as a table that maps keywords to context and its associated ODP nodes. Each node in the UserProfile is mapping between a context and a ProfileCore.

6. *ProfileCore.* ProfileCore is the basic building block of UserProfile. Contains information such as ODP node name, node ID, number of times the node has been visited,

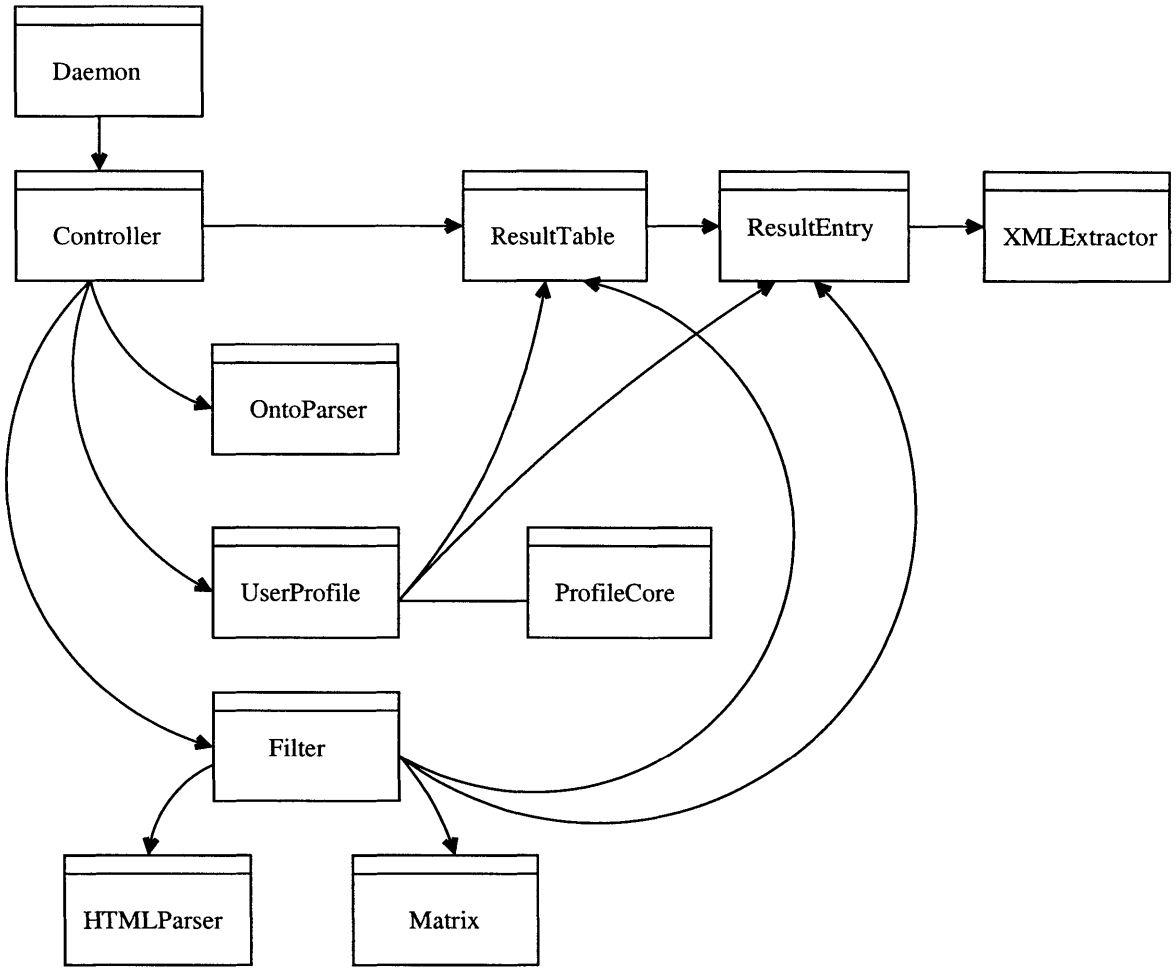


Figure B-1: Complete MDD for the *Persona* system

the number of times the node is rated positively and negatively.

7. *Filter.* Encapsulates all the graph based search techniques we discussed in chapter 2 and 3. The filter module does all the calculation and filtering to determine authoritativeness and hubness.

8. *HTMLParser* A class that 'crawls' through HTML pages and extracts information such as the links inside the page, page title, description, etc.

9. *Matrix* A helper data structure that Filter uses to do the calculations for the graph based search algorithms.

10. *Controller.* Controls the transaction between all modules. Manages the logic and control flow among each of the components. The controller interfaces with the input daemon to pass back results.

11. *Ontology Parser.* Parses in and traverses the ODP taxonomy. The parser identifies each node in the ODP by its ID number and name. The parser can return information such as a node's parent, children, and the set of nodes that may correspond to a certain string.

Figure B-1 is a schema of the full Module Dependency Diagram.

Bibliography

[1] Eytan Adar, David Karger, and Lynn Andrea Stein. Haystack: Per-user information environments. *Proceedings of the 1999 ACM CIKM International Conference on Information and Knowledge Management*, November 1999.

[2] Marco Balabanovic. An adaptive web page recommendation service. *Proceedings of the First International Conference on Autonomous Agents*, February 1997.

[3] Marco Balabanovic and Yoav Shoham. Fab: Content-based, collaborative reccomendation. *Communications of the ACM*, 40(3), March 1997.

[4] Daniel Billsus and Michael J. Pazzani. A hybrid user model for news story classification. *Proceedings of the Seventh International Conference on User Modelling*, June 1999.

[5] Phillip Bonacich and Paulette Lloyd. Eigenvector-like measures of centrality for assymetric relations. *Social Networks.*

[6] Phillip K. Chan. A non-invasive learning approach to building web user profiles. *Proceedings of ACM International Conference on Knowledge Discovery and Data Mining*, 1999.

[7] Huan Chang, David Cohn, and Andrew McCallum. Creating customized authority lists. *Proceedings of the Seventeenth International Conference of Machine Learning*, 2000.

[8] Boris Chidlovski, Natalie S. Glance, and M. Antonietta Grasso. Collaborative re-ranking of search results. *AAAI Workshop on AI for Web Search*, 2000.

[9] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Proceedings of International Workshop on Design Issues in Anonymity and Unobservability*, July 2000.

[10] Chrysanthos Dellarocas. Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior. *Proceedings of the Second ACM Conference on electronic Commerce*, October 2000.

[11] John Doyle and Michael P. Wellman. Representing preferences as *ceteris paribus* comparatives. *Working Notes of the AAAI Spring Symposium on Decision-Theoric Planning*, March 1994.

[12] Tom Fawcett and Foster Provost. Combining data mining and machine learning for effective user profiling. *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*, 1999.

[13] Christiane Fellbaum. *WordNet: An Electronic Lexical Database.* MIT Press, 1998.

[14] Tim Finin, Yannis Labrou, and James Mayfield. Kqml as an agent communication language. *IJCAI Workshop on Agent Theories, Architectures, and Languages*, 1995.

[15] Leonard Foner. Yenta: A multi-agent, referral-based matchmaking system. *Proceedings of the First International Conference on Autonomous Agents*, 1997.

[16] Yongjian Fu, Kanwalpreet Sandhu, and Ming-Yi Shih. Clustering of web users based on access patterns. *Proceedings of ACM International Conference on Knowledge Discovery and Data Mining*, 1999.

[17] Lisa Getoor and Mehran Sahami. Using probabilistic relational models for collaborative filtering. *Proceedings of ACM International Conference on Knowledge Discovery and Data Mining*, 1999.

[18] Alyssa Glass and Barbara Grosz. Socially conscious decision-making. *Proceedings of Agents2000 Conference*, June 2000.

[19] Qi He, Katia P. Sycara, and Timothy Finin. Personal security agent: Kqml-based pki. *Proceedings of the Second International Conference on Autonomous Agents*, 1998.

[20] Jinmook Kim and Douglas W. Oard. User modeling for information access based on implicit feedback. Technical Report HCIL-TR-2000-111/UMIACS-TR-2000-29/CS-TR-4136, University of Maryland, 2000.

[21] Jinmook Kim and Douglas W. Oard. Using implicit feedback for user modeling in internet and intranet searching. Technical Report CLIS-TR-00-01, College of Information Studies, University of Maryland, 2000.

[22] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Proceedings of the ACM-SIAM symposium on Discrete Algorithms*, 1997.

[23] Jon M. Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew S. Tomkins. The web as a graph: measurements, models, and methods. *Invited survey at the International Conference on Combinatorics and Computing*, 1999.

[24] Waikit Koh. An information theoric approach to interest matching. Master's thesis, Massachusetts Institute of Technology, 2001.

[25] Waikit Koh and Lik Mui. An information theoric approach to ontology-based interest matching. *Hawaii International Conference on System Sciences*, 2001.

[26] Peter Kollock. The production of trust in online markets. *Advances in Group Processes*, 16, 1999.

[27] Teppo Kurki, Sammi Jokela, Reijo Sulonen, and Marko Turpeinen. Agents in delivering personalized content based on semantic metadata. *Proceedings of AAAI Spring Symposium Workshop on Intelligent Agents in Cyberspace*, 1999.

[28] Tessa Lau and Eric Horvitz. Patterns of search: Analyzing and modeling web query refinement. *Proceedings of the Seventh International Conference in User Modeling*, 1999.

[29] Henry Lieberman. Letizia: An agent that assists web browsing. *Froceedings of International Joint Conference on Artificial Intelligence*, 1995.

[30] Nitin Manocha, Diane J. Cook, and Lawrence B. Holder. Structural web search using a graph-based discovery system. *Intelligence*, 12(1):20–29, 2001.

[31] Raymond J. Mooney and Loriene Roy. Context-based book reccomending using learning for text categorization. *Proceedings of the Fifth ACM Conference on Digital Libraries*, June 2000.

[32] Dan Murray and Kevin Durrell. Inferring demographic attributes of anonymous internet users. *Proceedings of ACM International Conference on Knowledge Discovery and Data Mining*, 1999.

[33] David M. Nichols. Implicit rating and filtering. *Fifth DELOS Workshop on Filtering and Collaborative Filtering*, November 1997.

[34] Douglas W. Oard and Jinmook Kim. Implicit feedback for reccomender systems. Technical Report WS-98-08, American Association for Artificial Intelligence, 1998.

[35] Michael Pazzani and Daniel Bilsuss. Learning and revising user profile: The identification of interesting web sites. *Machine Learning*, 27, 1994.

[36] Alexander Pretschner and Susan Gauch. Ontology based personalized search. *Proceedings of the Eleventh IEEE International Conference on Tools with Artificial Intelligence*, November 1999.

[37] Alexander Pretschner and Susan Gauch. Personalization on the web. Technical Report ITTC-FY2000-TR-13591-01, University of Kansas, 1999.

[38] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: An open architecture for collaborative filtering of netnews. *Proceedings of ACM Conference on Computer Supported Cooperative Network*, pages 175+, 1994.

[39] Yoav Shoham. Agent-oriented programming. *Journal of Artificial Intelligence*, 60(1):54+, 1993.

[40] Gilbert Strang. *Introduction to Linear Algebra.* Wellesley - Cambridge Press, 1993.

[41] Katia P. Sycara, Jianguo Lu, Matthias Klusch, and Seth Widoff. Matchmaking among heterogenous agents on the internet. *Proceedings of AAAI Spring Symposium on Intelligent Agents in Cyberspace*, 1999.

[42] Milind Tambe and Weixiong Zhang. Towards flexible teamwork in persistent teams: Extended report. *Journal of Autonomous and Multi-agent Systems*, 3:159+, 2000.

[43] Michael Woolridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 1994.