Deep-web Search Engine Ranking Algorithms

by

Brian Wai Fung Wong

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

Masters of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2010

Author ..[illegible]..
Department of Electrical Engineering and Computer Science
February 8, 2010

Certified by ..[illegible]..
Michael Stonebraker
Adjunct Professor
Thesis Supervisor

Accepted by ..[illegible]..
Christopher J. Terman
Chairman, Department Committee on Graduate Theses

Deep-web Search Engine Ranking Algorithms

by

Brian Wai Fung Wong

Submitted to the Department of Electrical Engineering and Computer Science
on February 8, 2010, in partial fulfillment of the
requirements for the degree of
Masters of Engineering in Computer Science and Engineering

Abstract

The deep web refers to content that is hidden behind HTML forms. The deep web contains a large collection of data that are unreachable by link-based search engines. A study conducted at University of California, Berkeley estimated that the deep web consists of around 91,000 terabytes of data, whereas the surface web is only about 167 terabytes. To access this content, one must submit valid input values to the HTML form. Several researchers have studied methods for crawling deep web content. One of the most promising methods uses unique wrappers for HTML forms. User inputs are first filtered through the wrappers before being submitted to the forms. However, this method requires a new algorithm for ranking search results generated by the wrappers. In this paper, I explore methods for ranking search results returned from a wrapped-based deep web search engine.

Thesis Supervisor: Michael Stonebraker
Title: Adjunct Professor

Acknowledgments

I would like to thank Professor Stonebraker for his kind guidance and input into this project. He has provided many instrumental ideas in guiding my research process. In addition, I would like to thank Mujde Pamuk for her patience in helping me debug code, and the whole Goby team for kindly providing assistance in my research. The Goby team has provided many insights into the real-world implementation of a deep web search engine.

I would also like to thank my parents, family, friends, and role models for their guidance in life. Their advice enabled me to take each step in life with righteousness and awareness. I have gained a tremendous amount of wisdom by listening to their invaluable life lessons.

"A person who never made a mistake never tried anything new."

– Albert Einstein

Contents

List of Figures

List of Tables

Chapter 1

General Overview

1.1 The deep web

In this section, I will introduce the *deep web* part of the internet. The topic of searching content in the deep web will be the focus of this Masters of Engineering thesis.

1.1.1 Information in the deep web

The deep web refers to web content that is not part of the surface web, which can be indexed by standard search engines such as Google and Yahoo. A static web page belongs to the surface web, whereas a SQL database accessible through a HTML form belongs to the deep web [22]. These web forms are usually part of script-based web pages written in languages such as PHP, ASP, and Ruby on Rails. Examples of deep web resources include shopping websites, Mapquest, currency converters, and online banking sites. A typical link-based search engine cannot access the deep web because the content is hidden behind these web forms. In order to crawl the deep web content, a search engine has to submit valid input values for the form before the dynamically generated content is created as the result of a specific search [1].

The deep web contains an enormous amount of information. According to a research study conducted at University of California Berkeley in 2003, the surface web contains approximately 167 terabytes of data [11]. In contrast, the deep web is estimated to contain 91,000 terabytes of data, which is more than 500 times the size of the surface web. This means traditional link-based search engines are not reaching 99% of the content on the World Wide Web. There are tremendous opportunities in exploring methods for indexing deep web content. The deep web has become one of the most exciting topics in search engine technologies [16].

1.1.2 Difficulty in mining the deep web

The difficulty in indexing the deep web lies in the fact that search engines cannot understand the context of web forms. Without human assistance, search engines cannot submit meaningful search inputs to HTML forms. The challenge of understanding web form becomes a very difficult natural language processing (NLP) problem. Furthermore, to create a commercially-available deep web search engine, one needs a scalable method of mining and processing data. Currently, there are only a few NLP technologies that are scalable to large data sets [14]. Most recently, a paper published in 2008 outlines a method to use Google's parallel computing framework (MapReduce) for scaling up language processing algorithms [18].

Dynamically generated pages are usually of little value if information is not up-to-date. For example, the billions of possible web pages generated by simple search queries at a travel website means it is virtually impossible to index all of the search results. These dynamic data often change frequently. Thus, a deep web search engine needs to frequently update its index, which can be a very expensive exercise. The lack of hyperlinked data in the deep web also makes it difficult to discover new data and to evaluate the quality of data.[20].

1.2 Background

In this section, I will provide a brief overview of various efforts in searching the deep web. There has been a significant amount of research on this topic. As this portion of the web represents a large percentage of information on the internet, it has long been acknowledged as a crucial gap in search engine coverage. A large number of research projects have tackled the problem of crawling the deep web in a scalable and efficient manner. However, not many have adapted the wrapper approach in mining the deep web content.

1.2.1 Google's deep web efforts

Google's Alon Halevy is one of the leading figures in deep web search. In his paper entitled *Google's Deep Web Crawl* [13], he outlined a methodology that crawls deep web content by generating appropiate input fields for HTML forms. If the engine encounters a page with a form related to restaurants, it would start guessing possible search terms - "Italian", "Chinese", "Japanese" etc - until one of these terms returns a meaningful search result. The Google search engine would then index the result data for future queries by the user [24]. The result can be converted into structured data later. In fact, Google Labs released a technology called *Google Squared*, which adds structures and relationships to raw data, and presents the result in a more organized manner.

In addition to developing an active deep web crawler, Google also published the Sitemap Protocol. The protocol enables websites to advertise accessible URLs that are normally not linked to the shallow web. These URLs would be crawled by Google through an XML file, and are added to their universal search index. In additional to Sitemap protocol, Apache's mod_oai also achieves search engine content discovery.

1.2.2 Commercial deep web engines

In addition to Google, there are several start-up companies that are developing deep web search engines. Kosmix, for example, is a hybrid deep web crawler and aggregator portal. In addition to indexing data, Kosmix's engine builds a taxonomy of five million categories on various topics.

Pipl is another well-known deep web crawler. Unlike other typical search engines, Pipl focuses on searching personal information. A user can search based on a person's name, username, or even phone number. Pipl performs most of its search in real-time, i.e. the engine actually queries its sources on the fly. Pipl crawls websites such as MySpace, Hi5, Flickr, and Amazon. The use of a *live* query ensures the timeliness of its data, however, search performance is sacrificed as a result. Search queries on Pipl generally takes at least 20 seconds to complete, which is orders of magnitude slower than searching for pre-indexed data on Google.

DeepDyve is a deep web crawler that specializes in indexing academic journals. It indexes websites such as ACM.org and Psychological Reviews. Unlike other deep web search engine, DeepDyve gains access to premium articles that are normally not accessible to a shallow web crawler, and makes these content available to the user on a pay-per-view basis. In addition, DeepDyve is a subscription based business, which greatly limits the size of its audience.

1.3 Morpheus approach for mining the deep web

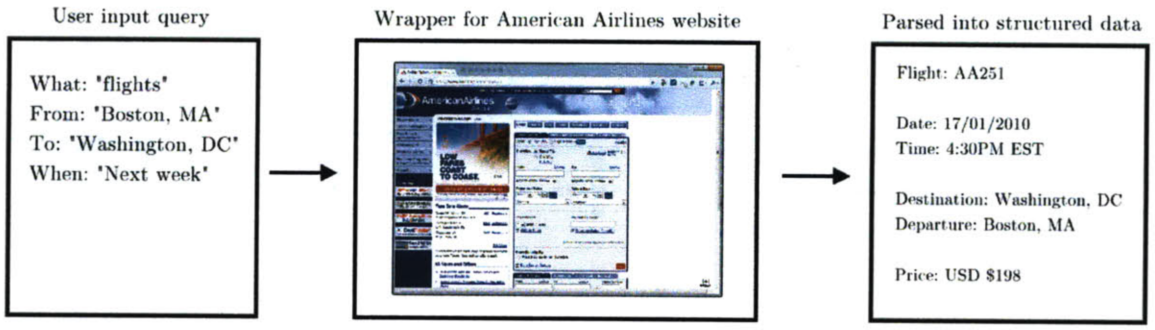
In this section, I will provide an introduction to the MIT Morpheus project, which eventually transformed into a commercial internet deep web search engine called Goby. The Morpheus project was started in CSAIL by Professor Michael Stonebraker. Professor Stonebraker was also the founder of many successful database companies, including Ingres and Postgres systems.

1.3.1 CSAIL Morpheus project

There have been numerous research projects dedicated to creating a deep web search engines. One of the most promising solutions involves the use of wrappers. Morpheus is a research collaboration between MIT and University of Florida. The engine uses website-specific wrappers to crawl the deep web and to transform search result into structured data. The Morpheus search engine initially queries the web on the fly - very much similar to the Pipl search engine. However, due to the performance and reliability limitations of live searches, the commercial version of Morpheus intelligently fetches information in the deep web and indexes the result.

1.3.2 Using wrappers to query the deep web

A wrapper transforms search terms into the range of possible input fields on any given web form. Each web form would have a unique wrapper written specifically for it. For example, a wrapper would describe how to query a plane ticket website that takes destination and travel dates as input arguments, and returns flight schedules and prices as a result. Figure 1-1 illustrates the concept of wrappers behind the Morpheus search engine.


User input query
Wrapper for American Airlines website
Parsed into structured data

What: 'flights'
From: 'Boston, MA'
To: 'Washington, DC'
When: 'Next week'

Flight: AA251

Date: 17/01/2010
Time: 4:30PM EST

Destination: Washington, DC
Departure: Boston, MA

Price: USD $198

Figure 1-1: This figure illustrates the concept of wrappers behind the Morpheus project.

The returned result is parsed into a structured data format. By containing subject, predicate and object triples, this format enables the Morpheus engine to understand meaning and relationship between data. The use of structured data is one of the major advantages of Morpheus over a shallow web search engine. Morpheus preserves the semantics of the returned data, and enables interactive result exploration such as user filtering. The structured data format also allows developers to easily develop website mash-up that take advantage of the Morpheus search engine [8].

Wrappers are usually generated by manual process. With the help of semi-automatic wrapper generation tool, a human being would identify each input for a web form and assign standardized categories to the inputs and outputs. The assignment of categories enables search results to be presented in a structured and organized manner to the user. This human-assisted wrapper generation process, however, is not yet scalable. There have been some research studies that attempt to automate the wrapper generation process, but they have not reached commercial applications.

Each wrapper is placed in a category hierarchy. A category hierary describes the "specificity" of a wrapper relative to other wrappers. A wrapper for a Boston Italian restaurant website is more specific than a wrapper for a generic restaurant search site. When a user searches for something in Morpheus, his search terms are first mapped to known categories in the category hierarchy. This matching allows the search engine to select the best wrappers to use. Figure 1-2 shows an example category hierarchy.

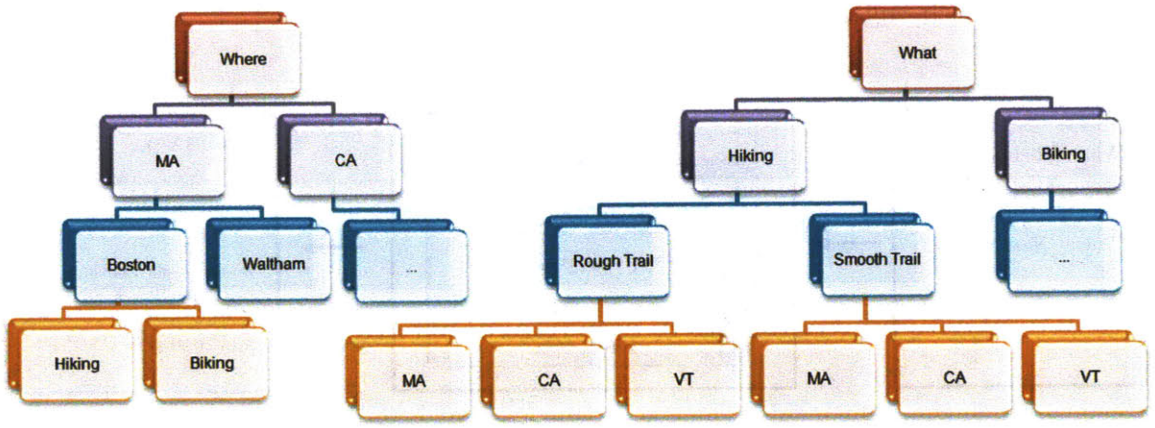


Figure 1-2: This particular data hierarchy demonstrates a deep web search engine designed specifically for travel-related queries.

1.3.3 Goby - commercial version of Morpheus project

The core ideas behind the Morpheus project have been implemented in a commercial deep web search engine called Goby. Figure 1-3 shows a the web interface of Goby.com in December 2009. Similar to the Morpheus project, Goby also utilizes website-specific wrappers to crawl the deep web. However, unlike the original project, Goby does not query on the fly. This means the Goby search engine actually indexes deep web content before the user executes the query. This process enables a much higher throughput and a more consistent search performance. A search for restaurants in Boston on Goby.com is generally completed within five seconds.

To achieve the goal of pre-indexing deep web content, the Goby engine structures search term combinations intelligently for each wrapper. For example, a wrapper for a chain restaurant website may take location, time and price as inputs for the HTML form. The Goby engine essentially forms all possible combinations of these three input fields, and queries the wrapped website. The returned result is parsed into structured data and indexed by the search engine for fast retrieval. The Goby index is updated periodically to reflect that latest information available from its sources.

While pre-indexing deep web resources is crucial for search performance, this method does have its draw-backs. First, the method requires forming all possible combinations of input terms for each wrapper. This imposes a high pre-processing cost for both the Goby crawler and the wrapped website. Unless the website queries are spaced out properly, they can be mis-interpreted as a denial of service (DDoS) attack. Second, at the point of query by the end user, the information is no longer real-time. While this may not be a problem for most information, some information such as weather data or flight information makes sense only if it is real-time.

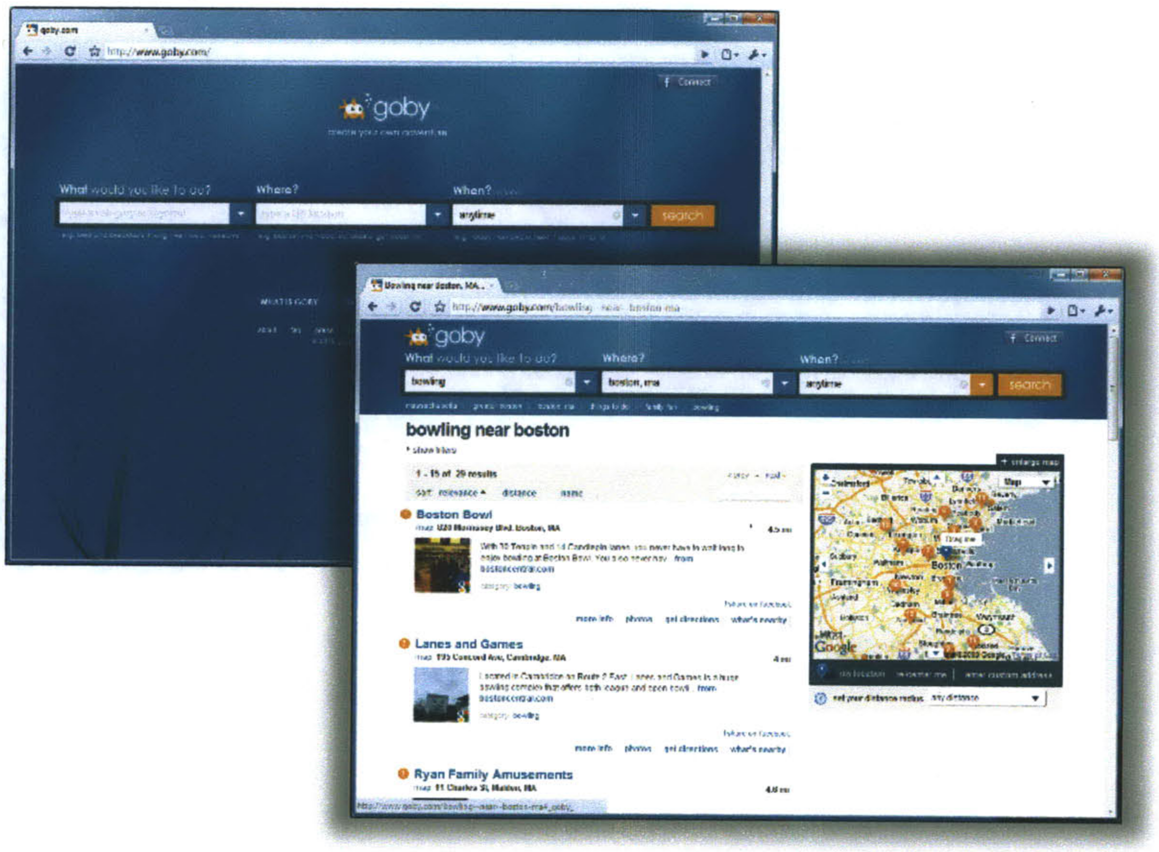


Figure 1-3: Interface of Goby.com as of December 20th, 2009

Chapter 2

The Ranking Problem

2.1 Research Overview

My research with Professor Stonebraker focused primarily on the ranking of search results generated by a wrapper-based deep web search engine. A search query is essentially a collection of search terms specified by the user. For example, a user query could be "Italian retaurants in Boston, MA" or "Sightseeing and tours in Las Vegas, NV". When a query is submitted to the search engine, the engine would find a list of results that match the user's criteria. However, it is important to present the list in a manner that minimizes user's effort in finding the result he needs. In other words, the engine needs to rank the results by relevance accurately.

Over the course of 6 months, I have worked with the Goby team to devise a new methodology to accurately rank deep web results. Our improved ranking algorithm utilizes (1) a best-fit scoring function using ten quality factors, and (2) a dynamic weighting algorithm that changes the factor weighting based on user behavior. This algorithm is scalable and requires minimal pre-processing to generate the factor weightings.

2.2 Motivation

Unlike the surface web, deep web data are not linked together by hyperlinks. Since the data are hidden behind web forms, it is also difficult to infer relationships between data. Typical search ranking algorithms such as the Google PageRank would not work for a deep web search engine (Figure 2-1). The lack of relationship creates a difficult challenge in determining which search result should be ranked first. While one may infer result quality based on knowledge about the origin wrapper, a result content-driven method should yield a more accurate ranking. The goal of my research is to develop an accurate and efficient ranking algorithm for the commercial implementation of the Morpheus project.

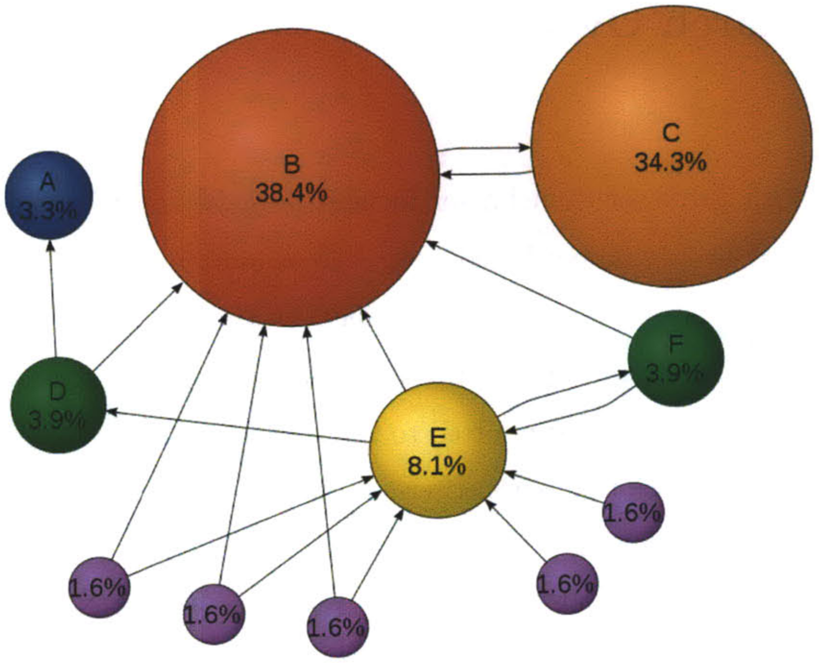


Figure 2-1: A graphical illustration of how PageRank algorithm works. PageRank essentially *flows* amongst linked nodes until an equillibrium state is achieved. Websites with a high number of incoming links (site B and C above) are considered popular and will receive a high PageRank. Unlike the shallow web, deep web content do not carry relationships such as hyperlinks. Consequently, a deep web search engine cannot utilize link-based techniques to rank results. *Image Credit: Wikipedia*

2.3 Deep web result scoring functions

In this section, I will present several scoring functions that can be used for ranking search results of a deep-web search engine. These are basic scoring functions that form the basis for the ten-factor scoring functions introduced in chapter 3.

2.3.1 Goby implementation

Currently, the Goby search engine utilizes a simple two-factor scoring fuction to rank results - a combination of distance score d and referral score r. The distance score is inversely proportional to the physical distance between a search result and the location of interest. The referral score represents the popularity of a result amongst wrapped websites. These two factors are each given an appropiate weight to compute the final score used for ranking results.

$$\text{Score} = \alpha \cdot d + \beta \cdot r \tag{2.1}$$

When a user searches for results (e.g. an event or a restaurant) at a specific location, he is generally interested in results that are physically close to his location of interest. In addition, result that appears numerous times in the raw search result is an indicator for its popularity.

Two factor model

The *distance* score is a metric indicating the physical proximity between the location of interest, e.g. Boston, and the location of the target result. In the Goby engine, a physical location always exists for every search result. The lower the distance, the higher the distance scoring metric. While the distance score is generally an accurate metric, however, some of the search results do not contain the exact address of the target location. This means the search engine has to approximate the location using the limited information it has regarding the target. It is possible that this approximation pushes *less* relevant results to the top of the result ranking.

On the other hand, the *referral* factor is an internal scoring metric generated during the crawl process. Since different wrappers may generate an overlapping set of search results, duplicates may appear in the raw result generated by Goby engine. In general, Goby performs a de-duplication process on the server side before sending back a clean search result back to the client side. When searching for restaurants in Boston, the result "Legal Sea Foods" is very likely appear in multiple wrappers. The referral score calculates the total number of times a specific result appears in the raw search result. The higher the number of duplicates, the higher the referral scoring metric. To a certain extent, this metric is similar to Google's PageRank - it attempts to infer popularity by determining the number of appearance of a particular search result.

Currently, the Goby search algorithm does not perform any dynamic weighting process. In other words, the pre-assigned weights are static attributes determined by the developers of the algorithm. The lack of a dynamic weighting process that responds to user behavior is an area of focus for my research work. My work adopts machine learning algorithms commonly used for rebalancing weights. Google, for example, utilizes similar methods for optimizing their search result rankings based on user behavior.

2.3.2 Other scoring functions

Due to the lack of relationship between data, the search engine must determine the quality of a search result based on the limited amount of information regarding it. In this section, I will present two additional methods that can be used for ranking deep web content.

Natural language processing techniques

If a search engine can apply NLP techniques to understand the content of the search result, then it can also use this understanding to infer content quality. For example, a NLP crawler can intelligently parse the returned result and place content into a structured database accordingly. The quality of a particular search result can be determined by the amount of structured data it contains.

It is possible to apply common natural language processing techniques such as statistical learning to analyze the contents of the result. Some recent research projects have taken advantage of cheap processing power provided by cloud computing technologies [18]. Nevertheless, NLP requires a huge amount of pre-processing that cannot be parallelized easily. Google has been engaged in projects that explore the possibility of NLP for its search engine. However, most of these projects have stayed at the research level [12]. Furthermore, if the NLP processing occurs in real-time, search performance may be hampered significantly.

Nevertheless, there are a few companies that have successfully incorporated NLP into their search engine technologies. In particular, a start-up called Powerset (acquired by Microsoft in 2008) was one of the first commercial search engines that supports basic natural language queries. The Powerset indexing engine performs context categorization during its crawl process, while the query engine performs real-time processing to understand natural language queries.

Shallow-deep web integrated methods

A deep web search engine can utilize a search result's URL as a unique identifier to check its Google PageRank. While PageRank describes the popularity of a website in the surface web, a more popular deep web result is likely to have a higher PageRank. However, this method has its limitations as well. Data mined from the deep web may not necessarily have a shallow web counterpart. Furthermore, even if there exists a shallow web counterpart, the two may have very different data quality. For example, if a travel search engine such as Goby or Kayak discovers a ticket website that contains details about a JFK→HKG flight on United Airlines, this result may get ranked higher than an equivalent Cathay Pacific Airlines flight simply because United Airlines has a high Google PageRank. As a result, using Google PageRank can skew the ranking towards "more popular" sites with high PageRank.

Chapter 3

Ten factor model

In this section, we introduce the ten-factor model for ranking deep-web search results. The model weights are calculated using a regression model developed in Visual Basic, Data Analysis Toolpak and AMPL. Data for the regression model are obtained through hundreds of experimental search query evaluations outlined in Chapter 4. Out goal is to apply experimental data to the regression model, which generates best-fit factor weights.

3.1 Motivation

Motivated by previous factor-based models, we want to develop a more sophisticated model that can capture users preference for evaluating the quality of search results. Consequently, we can utilize this factor-based model as a scoring function for ranking deep web search results. The factor weights are obtained by linear regression using experimental data. This method can be easily implemented in the current infrastructure of search engines and is highly scalable with little pre-processing overhead. There has been numerous research on the topic of distributed regression models [2].

In chapter 4, we explore in more details the experimental process we used for determining the factor weights of the scoring function. This chapter will be more focused on the theoretical underpinnings of the model.

3.2 Model definition

In this section, we will define the ten factors of the regression model. The ten factors represent possible metrics a typical user would look for when they evaluate the quality of search result. Once the factor weights are determined in the regression analysis, the model is used by the engine for ranking search results.

3.2.1 General linear regression model

The ten-factor linear regression model has the following general form:

$$R = \alpha_0 + \sum_{i=1}^{10} \alpha_i \cdot F_i + \epsilon \tag{3.1}$$

The entity R, which ranges from 0 to 10, is defined as the *quality score* of a particular search result.

In the regression model, this quality score is provided by the user during the experiment. This score represents quantitatively the usefulness of a particular search result to the user. The factors in the model, F_i, are defined in table 3.2.2. These factors represent features accessible to the user on the Goby search engine. The weight α_i for each factor is obtained through regression analysis.

The linear regression model we adopt is an error minimization regression model, i.e. we are finding "best-fit" factor weights to minimize the error. The error, ϵ, is defined as the difference between the actual experimental value, R, and the calculated value, $\bar{R}$, using regressed factor weights. Essentially, the regression model minimizes the sum of error squared [19]. The error minimization problem can be rewritten as a convex optimization problem as follows:

$$\begin{aligned} min \quad & \sum_{i=1}^{N} (R_i - \bar{R}_i)^2 && (3.2) \\ s.t. \quad & \bar{R}_i = \alpha_0 + \sum_{j=1}^{n} \alpha_j \cdot F_i && (3.3) \\ & \alpha_j \geq 0 \quad \forall j \subseteq (0...n) && (3.4) \end{aligned}$$

We utilize a convex optimization technique called Karush-Kuhn-Tucker conditions (KKT) combined with Newton's Method for solving the optimization problem. This is a convex problem because the objective function and the constraints satisfy convexity conditions. The factor weights α_i generated from this optimization problem represent the best-fit line for original ten-dimensional regression model [10].

3.2.2 Regression factors

The following table defines the 10 factors used in the above linear regression model. These factors are chosen carefully as they represent qualities users look for in the search result when they evaluate how useful it is. Our goal is to closely model user behavior using a best-fit linear regression model. While it is possible to add additional factors to the model, we want to keep our factor limited to features and data that are available to the user in the Goby interface.

The definitions for factor F_4 and F_5 are discussed in more details below. These two factors play an important role in determining the relevancy of the search result. During the experiment, it was observed that users exhibit preferences for proximity in location and time when they evaluate the quality of search result.

Definition of 10 factors		
Visually related	F_1	Binary indicator for presence of business supplied image
	F_2	Binary indicator for use of Google images
	F_3	Continuous variable $\subseteq (0...1)$ provided by user for indicating the quality of "More Photos" tab
Distance/Time related	F_4	*Distance metric* for indicating proximity to desired search location
	F_5	*Time metric* for indicating closeness to search date
Data related	F_6	Binary indicator for presence of "Description"
	F_7	Binary indicator for including a "Detailed location"
	F_8	Binary indicator for including a "Phone number"
	F_9	Binary indicator if price is available
	F_{10}	Continuous variable $\subseteq (0...1)$ provided by user for indicating the usefulness of "More Info" tab

Table 3.1: The ten factors used in the regression model

Distance metric

We define the *distance metric* as a normalized score that is inversely proportional to the distance between the target result location, and the location of interest. The original location of interest is indicated by the user when the search query is constructed.

The distance score, $\bar{D}$, is normalized before being used as a variable in the regression model. This score is computed in Excel before the data is fed into the regression model. The justification for normalization is that we ask users to rank only the top 20 search results. Therefore, we would like to construct a distance metric that compares only the relative proximity to desired location amongst these 20 results. The normalized metric is defined as follows:

$$\bar{D} = \frac{D_{max} - D}{D_{max} - D_{min}} \tag{3.5}$$

$\bar{D} \subseteq (0...1)$ is the normalized distance metric, while D_{max} and D_{min} are the maximum and minimum distance within the ranked 20 search results, and D is the actual physical distance indicated by the Goby engine.

This definition of the distance metric captures user's preference for physical proximity. For example, when the user searches for restaurants near 77 Massachusetts Avenue Cambridge, a result that is closest to the requested point of interest should have a distance score of 1. On the other hand, results that are farther away should have lower distance scores. Given our observations during the experimental process, we believe users have measurable preference for the physical proximity of the target result. In chapter 5, the relatively large regression weight in the regression model indicates such a preference. While the experimental queries use cities as search locations, this fact does not change our conclusion that users value physical proximity of the result. If the user has a more specific location in mind, it would not be surprising a even higher emphasis on physical proximity.

Time metric

We define the *time metric* as another normalized score within the $\{0...1\}$ range. The time metric is defined as 1 for results that do not have a specified date, i.e. businesses/attractions that are available daily, or results that occur within one day range of the query date. The query date is usually defined by the user in the search query. However, if the date is not defined, then it is assumed to be the day at which the query is executed.

Similar to the distance metric, the time metric is defined as follows:

$$\bar{T} = \frac{T_{max} - T}{T_{max} - T_{min}} \tag{3.6}$$

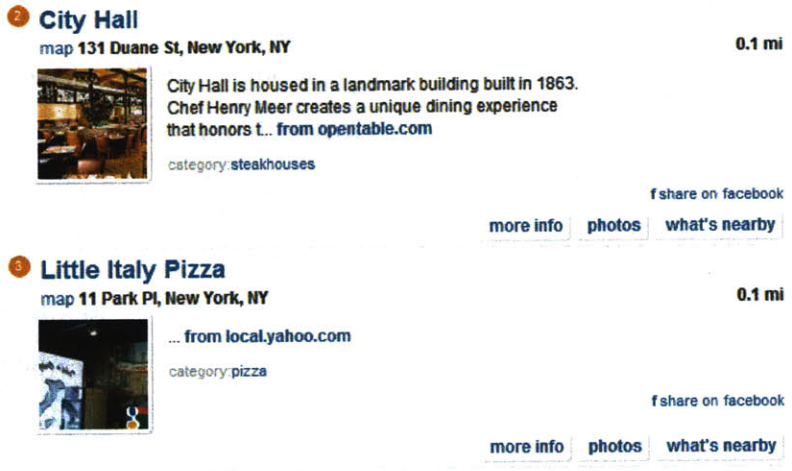
$\bar{T} \subseteq (0...1)$ is the normalized time metric, while T_{max} and T_{min} are the maximum and minimum number of days from the query date amongst the 20 ranked results, and T is the actual number of days away for a particular search result.

The time metric models user preference for closeness in the time dimension. In chapter 5, we discuss in details the importance of time proximity to the user. In the experimental search queries, no specific time is provided to the search engine. The reason is that is that users generally do not have a specific date in mind. It is more helpful to the user to see all the results and to allow them to to discover the useful results by using filters/sorting. In addition, by not specifying a time, the engine returns a more diversed list of search results to the user. This provides a greater varierty in features for the regression model to identify the important factors. It is observed that users demonstrates a consistent behavior that shows a very strong preference for time proximity. In other words, experimental users appear to place significant value on search results (e.g. an event) that are taking place close to the search date.

Example search query and result

Figure 3-1 illustrates two example results obtained when a user searches for "Retaurants in New York City, NY". The first result would obtain 1 for the first factor F_1, as it contains an image supplied by the business itself. The distance metric factor, F_4, is 1 because this restaurant is physically closest to the location of interest (New York City). The time metric factor, F_5, is also 1 because the retaurant is open daily. This result also contains a description, a detailed location, and a phone number (inside "More Info" tab). Thus, F_6, F_7, and F_8 are 1 as well. The remaining factors are zero.

Unlike the first result, the second result does not contain a description of the business. In addition, instead of having a business supplied image, this result contains an image pulled from Google Images. It also does not have a phone number. Consequently, its factors F_1, F_2, and F_8 are 0, 1, 0 respectively. It is interesting to note that the F_3 and F_{10} factors are 0 for both results, as the "More photos" and "More info" tabs do not contain additional useful information.


2 City Hall
map 131 Duane St, New York, NY
0.1 mi
City Hall is housed in a landmark building built in 1863. Chef Henry Meer creates a unique dining experience that honors t... from opentable.com
category: steakhouses
f share on facebook
more info | photos | what's nearby

3 Little Italy Pizza
map 11 Park Pl, New York, NY
0.1 mi
... from local.yahoo.com
category: pizza
f share on facebook
more info | photos | what's nearby

Figure 3-1: Figure illustrating two search result on Goby. The first result contains additional information not contained in the second result. It includes a business supplied image and a phone number.

3.2.3 Expansion to the ten factor model

While we have limited our model to feature-oriented factors, we can expand it to include factors based on data that are not accessible to the user. These include factors such as Google PageRank, the internal referral score defined in section 2.3.1, category hierchy metrics, and other statistical scores. In particular, the referral score and the category hierchy metrics are Goby-specific factors.

Google PageRank and internal referral score

The internal referral score counts the total number of appearance of a particular result amongst all the deep web wrappers. Its goal is very much the same as Google PageRank, as both metrics attempt to measure the popularity of an entity. Thus, it is possible to utilize these to metrics together in a way that captures the shallow web popularity through PageRank, and the deep web popularity through referral score.

In an iteration of the ten factor model, I will add in the referral score as the 11^{th} factor. As seen in chapter 6, this metric is ranked 3^{rd} highest by factor weight. However, in terms of weight value, its weight is actually not that large compared to the two most important factors.

Category hierarchy metric

Goby utilizes a category hierarchy to place its wrappers and cached crawl results (Figure 1-2). Thus, it is possible that the search engine can guess the quality of a result based on where it is located on the hierarchy. For example, suppose the user is searching for restaurants in Boston. The result "Legal Seafoods" is most likely placed in the category called "Retaurants", whereas the result "Faneuil Hall" is likely to be placed in categories such as "Things to do", "Events", and "Restaurants". In other words, "Faneuil Hall" is a less specific result than "Legal Seafoods", and should be ranked lower.

In addition to using the number of matches in the category hierarchy, the search engine can also use the location of a wrapper to infer its specificity. A wrapper that is deep in the hierarchy tree is more specific than a wrapper that is higher up. However, it is possible that one branch of the hierarchy tree is simply much deeper than another branch of tree, allowing a wrapper to be placed lower down. The use of the category hierarchy metric appears to be a promising research direction in future work.

Chapter 4

Experiments

4.1 Overview

600 experimental queries were conducted with 30 test subjects. Each person was asked to rank the top 20 search results of 20 different queries. The goal of our experiment is to provide enough data points to execute a reliable linear regression and obtain the 10 factor weights, α_i. These factor weights would form a 10-factor scoring function that can be used for ranking results more accurately than the current 2-factor scoring function.

4.2 Experiment design

We have prepared 20 unique queries covering four broad categories. These queries represent a variety of different travel-related topics and locations that may be of interest to a typical user. The 20 queries are summarized in table 4.2. These queries are structured based on the commercially available Goby.com search engine, and include categories such as "Food and Drinks", "Places to Stay", and "Attractions".

30 experimental users of different ethnicity, gender, and age were asked to execute these 20 queries. By seeking user's opinion on individual search result, we hope to determine quantitatively how users evaluate the quality of search results in general. For each individual query, we ask users to provide a *quality score* from 0 to 10 for

the top 20 results returned by the Goby engine. The quality score is then used as the independent variable in the linear regression model outlined in chapter 3.

For every test user, we generate 20 scoring functions after running the regression model. These 20 scoring functions are then checked for self-consistency to ensure the user is exbiting consistent behavior in evaluating the quality of search results. The process for checking self-consistency score is outlined in section 4.2.2, where we have defined a self-consistency score. Users with low self-consistency scores are rejected, and are not used for generating the universal scoring function.

Summary of experimental queries	
Category	*Search term*
Food and Drink	Bars and Pubs [Boston, MA]
	Restaurants [Miami, FL]
	Food and Drink events [San Francisco, CA]
	Wineries [Los Angeles, CA]
	Restaurants [Seattle, WA]
	Coffee and Cafe [New York City, NY]
Things to do	Entertainment and Night life [Boston, MA]
	Adventure and Extreme [Miami, FL]
	Nature [Washington DC, MD]
	Sight seeing and Tours [Las Vegas, NV]
	Places of interest [San Francisco, CA]
	Festivals [Miami, FL]
	Nature [Chicago, IL]
Places to Stay	Hotels [Boston, MA]
	Hostels [San Francisco, CA]
	Hotels [New York City, NY]
Events	Sports events [Boston, MA]
	Comedy events [New York City, NY]
	Arts events [Chicago, IL]
	Museum and Gallery events [Philadelphia, PA]

Table 4.1: This table summarizes the different categories of the twenty queries used in the experiment.

We have chosen to run relatively general queries because they tend to yield a large variety of results. This means there is usually a significant variation in the quality of results, as the search engine draws results from many different sources. This diversity allows the regression model to accurately single out important factors to the users when they evaluate the quality of search results. In future work, we would like to run queries with a variety of specificity and observe how regression results would differ.

4.2.1 Experimental process

The goal of the experimental process is to simulate a regular deep web query on Goby, i.e. the user is performing the search himself. We provide the query terms verbally to the user. The user then visits Goby.com and keys in the query terms manually. Next, we ask the user to provide a quality score within (0...10) for the top 20 search results. Figure 4-1 illustrates the experimental search process on Goby.com.

In addition to registering the quality score, user behavior and comments during the experiment are carefully observed to help us understand how site visitors would evaluate the search quality on Goby's site. User's behavior helps us identify factors that are important to them. This in turn enables us to define meaningful factors in the regression model.

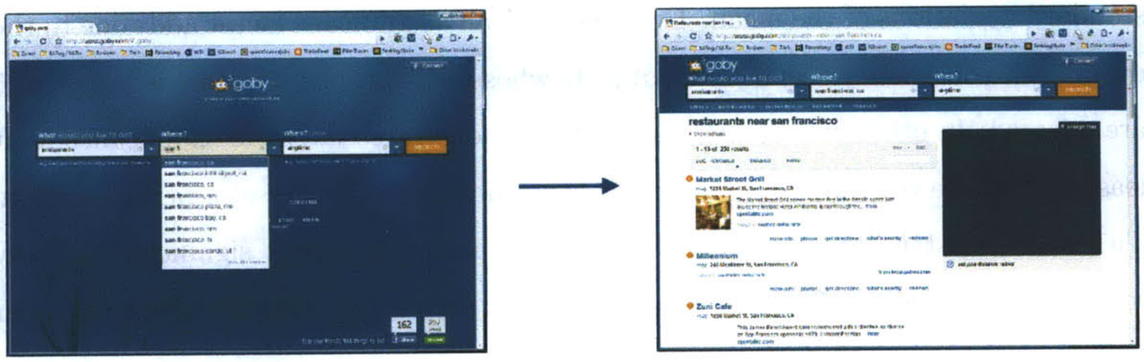


Figure 4-1: Users are given query via verbal instructions. (1) The user then visits Goby.com and inputs query terms, and (2) ranks the top 20 search result by providing a quality score $\subseteq$ (0...10) for each individual result. Each user generates 20 scores/query $\cdot$ 20 queries/user = 400 data points/user.

4.2.2 Self-consistency score

The self-consistency score, σ, ensures each human test subject is exhibiting consistent behavior when ranking search results. In general, a user's method for evaluating the quality of search result is an inherent psychological behavior that does not deviate within a short period of time. While most test subjects appear to demonstrate a high level of self-consistency, a few (around 23%) are not statistically self-consistent. Problem arises when we factor inconsistent individual scoring functions into our universal scoring function. The factor weights are contaminated by statistically insignificant values.

Consequently, we need a method to determine the self-consistency of users. We define the self-consistency score of a user as follows:

$$\sigma = \sum_{i=1}^{10} \beta_i \tag{4.1}$$

β_i is defined over the 20 queries executed by each user. It is defined as:

$$\beta_i = \begin{cases} 1 & \text{avg}(F_i)/\text{std}(F_i) \geq 2 \\ 0 & \textit{otherwise} \end{cases} \tag{4.2}$$

A particular factor is statistically significant if zero is not contained within two standard deviations of the mean. Every statistically significant factor adds one to the overall self-consistency score. Test subjects whose self-consistency score is less than 5 are rejected. In other words, only users who demonstrate statistical significance in at least half of the ten factors are used for constructing the universal scoring function. This self-consistency test is crucial for ensuring the statistical significance of any resulting universal scoring functions. Table 4.2 shows the scenario where test user #17 passes the test, while test user #27 fails the test.

The self-consistency score ranges anywhere from 1 to 8 in the experimental data. Amongst the 30 test users, 7 of them are statistically inconsistent (23%). While this is a relatively small percentage of the total size, most scores are clustered in the 4 - 6 range. Consequently, if the cut-off score is changed to 6, then 15 users would fail the self-inconsistency test (50%). If the cut-off score is changed to 7, then 19 users would fail (63%). The error tolerance greatly affects the number of available data points. The 50% cut-off was chosen because we believe that user's behavior is essentially predictable and can be captured by a scoring function.

σ	Test #		F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
🟩		Mean	0.70384	0.00000	0.00000	0.59534	1.27219	2.84019	0.45990	0.00000	0.25233	0.00000
6	17	Std	0.21851	0.00000	0.00000	0.50035	0.59290	0.97249	0.40394	0.00000	0.09682	0.00000
		Mean/Std	3.22109	N/A	N/A	1.18985	2.14573	2.92052	1.13854	N/A	2.60620	N/A
🟥		Mean	0.53329	0.24260	0.04736	0.37812	0.74156	1.07418	0.45089	0.00000	0.12637	0.00000
2	27	Std	0.38913	0.13049	0.02529	0.42094	0.35105	0.33895	0.79334	0.00000	0.19944	0.00000
		Mean/Std	1.37046	1.85925	1.87218	0.89827	2.11241	3.16917	0.56834	N/A	0.63364	N/A

Table 4.2: Table demonstrating two users where #17 passes and #27 fails the self-consistency test. A factor is statistically significant if its Mean\Std ratio is ≥ 2. The self-consistency test rejects test users if their score is less than 50%, i.e. at least 5 of the 10 factors are statistically significant. This test is crucial for ensuring the statistical significance of the analysis. For the first user, his factors F_1, F_5, F_6, and F_9 are statistically significant (mean\std ratio ≥ 2). This represents a self-consistency score of 67%. On the other hand, the second user passes has only two statistically significant factors - F_5 and F_6. This means he has a self-consistency score of 25%. N/A factors are ignored as they do not play a role in the regression model.

Chapter 5

Experiment Results

5.1 Results overview

The result of the linear regression model are summarized in table 5.1. In the experiments, we have conducted 600 search queries on Goby, involving 30 test users in evaluating the quality of search results.

Linear regression factor weights											
Test #	Self-consistency Score	Main Image (real)	Main Image (Google)	More Photos	Distance from Point of Interest	Closeness to search date	Description	Detailed Location	Price	Phone number	More info
1	7	0.253673	0.085322	0.000000	0.039033	0.360451	1.382905	0.539099	0.000000	0.000000	0.000000
2	8	0.501098	0.194857	0.020494	0.287910	0.694083	1.100584	0.470918	0.000000	0.104740	0.000000
3	3	nullified	nullified	nullified	nullified	nullified	nullified	nullified	nullified	nullified	nullified
4	5	0.133045	0.090493	0.001950	0.130538	1.703716	2.405339	0.230548	0.000000	0.021043	0.000000
5	6	0.204731	0.140483	0.000000	0.490638	0.936110	1.406903	0.390829	0.000000	0.009438	0.000000
6	5	0.843204	0.250871	0.049671	0.098059	1.104960	1.269129	0.194038	0.000000	0.000000	0.000000
7	5	0.130741	0.000000	0.000000	0.351795	1.242584	2.604837	0.410974	0.000000	0.328403	0.000000
8	7	0.348592	0.181322	0.002439	0.409778	0.429525	1.332934	0.702083	0.000000	0.000000	0.000000
9	4	nullified	nullified	nullified	nullified	nullified	nullified	nullified	nullified	nullified	nullified
10	5	1.204222	0.119521	0.000000	0.402797	1.319087	0.849049	0.330194	0.000000	0.209473	0.000000
11	5	0.087142	0.003731	0.000000	0.556333	1.242811	2.594719	0.415663	0.000000	0.301894	0.000000
12	6	0.265768	0.000000	0.000000	0.011867	0.360801	1.448985	0.498576	0.000000	0.000000	0.000000
13	5	0.243606	0.000000	0.000000	0.391487	0.892004	1.433523	0.330761	0.000000	0.000000	0.000000
14	7	0.704871	0.204974	0.000000	0.375926	1.830419	1.029488	0.194724	0.000000	0.309843	0.000000
15	8	0.188894	0.000000	0.000000	0.300739	1.257218	2.834414	0.341951	0.000000	0.300723	0.000000
16	5	1.394367	0.000000	0.000000	0.403302	1.087304	1.224526	0.288525	0.000000	0.009738	0.000000
17	6	0.703840	0.000000	0.000000	0.595340	1.272193	2.840190	0.459897	0.000000	0.252333	0.000000
18	7	0.198717	0.000000	0.000000	0.256876	1.698727	2.369723	0.337132	0.000000	0.034689	0.000000
19	2	nullified	nullified	nullified	nullified	nullified	nullified	nullified	nullified	nullified	nullified
20	6	0.472106	0.007381	0.000000	0.384964	0.767801	1.890218	0.766467	0.000000	0.034380	0.000000
21	1	nullified	nullified	nullified	nullified	nullified	nullified	nullified	nullified	nullified	nullified
22	5	2.165331	0.805174	0.000000	0.454191	1.132868	0.982293	0.358357	0.000000	0.000000	0.000000
23	8	0.495279	0.249084	0.000000	0.476300	0.717950	1.937735	0.729356	0.000000	0.007026	0.000000
24	7	0.897220	0.000000	0.000000	0.192823	1.646129	0.867449	0.317413	0.000000	0.173568	0.000000
25	6	1.163549	0.039362	0.000000	0.435010	1.312789	0.691535	0.598955	0.000000	0.248980	0.000000
26	7	0.040560	0.000000	0.000000	0.236249	1.689980	2.209435	0.371240	0.000000	0.009430	0.000000
27	2	nullified	nullified	nullified	nullified	nullified	nullified	nullified	nullified	nullified	nullified
28	7	0.185460	0.000000	0.000000	0.535035	1.212330	2.277610	0.417355	0.000000	0.290793	0.000000
29	2	nullified	nullified	nullified	nullified	nullified	nullified	nullified	nullified	nullified	nullified
30	4	nullified	nullified	nullified	nullified	nullified	nullified	nullified	nullified	nullified	nullified

Table 5.1: This table summarizes the regression results. Each person has an averaged scoring function if his behavior passes the self-consistency test outlined in section 4.2.2

In this section, we present the process for analyzing the results of the experiments. In particular, the discussion focuses on how to transform the raw data into universal scoring functions used in Goby. This section provides the groundwork for the next section, which presents the algorithm for dynamically updating weights based on user behavior. The focus of this section is on the following topics:

- Generating individual scoring functions
- Categorizing users and queries
- Generating universal scoring functions
- Change in quality of search result ranking

5.2 Individual scoring functions

Each test user executed 20 search queries on Goby's website. For each query, users were asked to provide the quality score, R (Section 3.2.1), for every individual result ranked in the top 20 by Goby.

There are two approaches in analyzing these data - (1) group all 400 data points into one single bucket and perform linear regression as if all data points are generated in a single session; (2) maintain data separation and run linear regression only on the 20 data points of each query, i.e. generating 20 scoring functions for each individual. Figure 5-1 illustrates the two approaches. We have chosen the latter approach in the analysis, as we believe it is crucial to ensure the test user is exhibiting consistent behavior across queries. In other words, if a user has more than five factors that are statistically insignificant (high variability), this user's behavior is considered inconsistent. Consequently, it does not make sense to generate a scoring function for an inconsistent user. His results are not included in the categorization of users and creation of universal scoring functions. In addition to maintaining statistical significance, putting all users into one bucket prevents us from analyzing individual behavior, which could yield interesting categorization of users as seen in section 5.3.

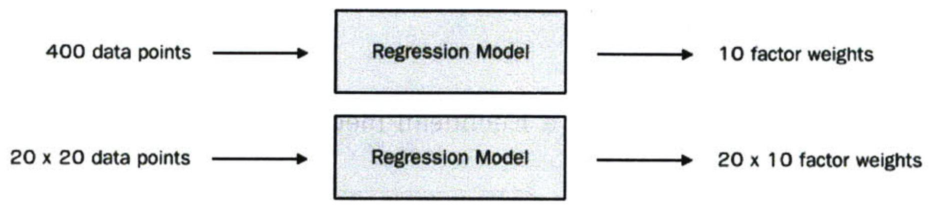


Figure 5-1: Figure illustrating two approaches in analyzing individual data. Top: bucket all 400 data points and generate one scoring function. Bottom: maintain separation of data points and generate 20 scoring functions. The latter approach is taken to ensure self-consistency of data.

Table 5.1 previously shows all 30 individual scoring functions obtained through the linear regression model and self-consistency check. The factor weights for each individual are obtained by taking the average across his 20 scoring functions. It is important to note that only statistically significant factor weights are used in generating the individual scoring function. Otherwise, the weight is simply considered zero.

5.3 Categorizing users and queries

Having generated the 30 individual scoring functions, it is interesting to see if users cluster into distinguishable categories based on their scoring functions. For example, there may be users who exhibit a strong preference for visual elements. Consequently, for this type of users, we expect relatively large factor weights placed on visually-related factors, and relatively small weights placed on data-related factors.

This categorization of user behavior enables Goby to use k-neighbor classification to identify the type of a new user [4]. When users interact with the website, the search engine can see what kind of features are included in each result that he clicked on. For example, if the user clicks on only results that contain a description, than it would make sense to categorize the user as someone who values data. Once the user type has been identified, the Goby search engine can apply the best-fit scoring function to improve search result quality.

5.3.1 Grouping by users

In order to cluster users, we utilize a Euclidean metric defined as follows [5]:

$$D = \sqrt{\sum_{k=1}^{10}(\alpha_{k,i} - \alpha_{k,j})^2} \qquad (5.1)$$

This metric enables us to calculate the Euclidean distance, D, between scoring function i and scoring function j. The metric simply calculates the square root of the sum of squares of the difference in factor weights. The Euclidean matrix for every (i, j) pair is included in appendix B due to its large size.

By observing the Euclidean distance between test user pairs, we can gain insights into the proximity of users in the 10-dimensional space spanned by the factor weights. We expect test users who fall into the same category of behavior to be relatively "close together" as defined by equation 5.1. On the other hand, users who exhibit different preferences should have their weight vector pointing in a different directions. By analyzing the Euclidean distance, users generally fall into four distinct buckets characterized by their distribution of factor weights: data-weighted users, distance/time proximity weighted users, visually weighted users, and uncategorized users. The first three categories of users place extra emphasis on certain factors when they evaluate search quality.

The following figure summarizes the break-down of user categories:

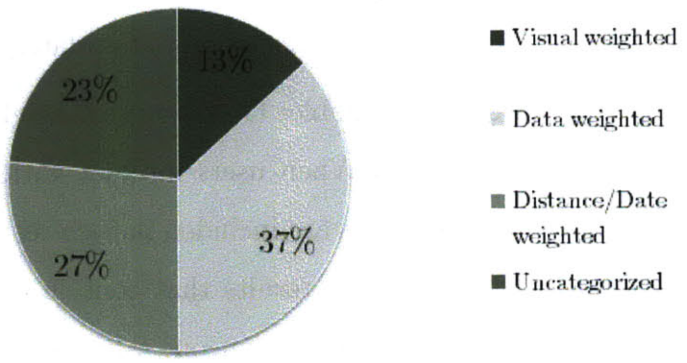


Figure 5-2: Figure showing the breakdown of user categories. These categories are defined based on characteristics of their regression factor weights.

Three main categories of users emerge by observing the Euclidean distance matrix in appendix B:

Data weighted users

This category of users demonstrates a very strong preference for information related to the search result. This fact is evidenced by the large factor weights on data-related factors such as *description*, *detailed location*, and *phone number*. Test users numbered 1, 5, 7, 8, 11, 12, 15, 17, 20, 23, and 28 fall into this category.

While this category of user are characterized by their large regression weights on data-related factors, we also observe a universal preference for the *description* factor across all category of users. It appears users find the simple one-line description in some of the search results to be extremely valuable. This interesting fact is further supported by user comments during the experimental process. A few users actually noted frustration when they find it difficult to tell what some of the search results are without a simple description.

Distance/time proximity weighted users

This category of users demonstrates a strong preference for proximity in the distance and time dimension. In other words, these users prefer search results that are (1) physically close to his location of interest, and/or (2) occuring close to the query date. This fact is evidenced by the large factor weights on distance/time related factors such as the *distance metric* and the *time metric*. Test users numbered 4, 10, 13, 14, 18, 24, 25, and 26 fall into this category.

In general, the weight on the *time metric* appears to be much larger than the weight on the *distance metric*. An explanation for this behavior is that when a user search for things to do in a city, the user does not have a specific address in mind. As a result, search results that are within the near vicinity of the city center are generally considered equivalent. It is also interesting to note that while no specific time is specified in the search query, users in this category still prefer getting results that take place as close to the query date as possible. This surprising result may indicate a general user preference for time proximity regardless of the query date input.

Visually weighted users

This category of users demonstrates a strong preference for visual elements in the search results. This fact is evidenced by the relatively large factor weights on visually-related factors such as *business supplied image*, *Google images*, and *use of more photos*. Test users numbered 2, 6, 16, and 22 fall into this category.

It is interest to note that the factor weight on the *business supplied image* factor is much larger than other two weights. This is not a surprising result as Google images are generally less helpful to the user than business supplied images. In fact, images extracted from Google often appear to be irrelevant to the business itself. As a result, this feature can potentially cause confusion to the user.

5.3.2 Grouping by queries

In the previous section, we constructed categories based on user preferences. However, it is also possible to see if patterns emerge based on query categories. In section 4.2, we defined four broad categories of queries: (1) *Food & Drink*, (2) *Things to do*, (3) *Places to stay*, and (4) *Events*. These categories are defined based on the category hierarchy of the Goby engine. The following table summarizes the results if we break down user behavior by query category.

	Factor weight α_i									
Query Category	α_1	α_2	α_3	α_4	α_5	α_6	α_7	α_8	α_9	α_{10}
1	0.3321	0.1326	0.0108	0.2962	1.0834	1.8038	0.3857	0.0000	0.0718	0.0000
2	0.5471	0.0410	0.0000	0.4134	1.2678	1.8724	0.3232	0.0000	0.1745	0.0000
3	0.4727	0.0887	0.0775	0.4396	1.2580	1.9280	0.4316	0.0000	0.1595	0.0000
4	0.6311	0.0247	0.0000	0.3939	1.0564	1.7915	0.3986	0.0000	0.1547	0.0000

Table 5.2: Table summarizing the results of breaking down user behavior by query category. It is interesting to note that patterns are much less obvious than previously. This result suggests user preferences do not vary much with respect to query category. It is important to note that preference for the distance and time factors, α_5 and α_6, are strong and consistent across query category. This indicates a general user preference for time proximity and the presence of a description in the search result.

It is interesting to observe that there does not appear to be a obvious pattern in the results, i.e. user behavior seems fairly similar across query categories. The weights on visually-related factors appear to be slightly higher for the *Things to do* and *Events* categories. Nevertheless, this lack of differentiation suggests that user behavior may not actually change that much with respect to query category. Users value essentially the same factors when they evaluate the quality of a search result.

5.4 Generating universal scoring functions

The three categories of users - *visual*, *data*, and *distance/time-weighted* - enable us to define three universal scoring functions. Users with sufficient activity on the Goby website can be placed into the appropiate bucket using simple k-neighbor classification algorithm. Based on the search results he clicked on, each user has a vector describing his preference for features (described by the 10 factors). k-neighbor classification simply uses Euclidean distance to see which category a user falls into. The goal of using three different scoring functions is to generate more relevant search results based on predicted user preferences.

Each scoring function is obtained by averaging the factor weights, α_i, of users who fall into the category. Users who are assigned a category would have to have passed the self-consistency test. This fact ensures the universal scoring function generated is statistically significant.

Result scoring function for each category

Data-weighted category

$$\begin{aligned} \bar{R}_d &= \sum_{i=1}^{10} \bar{\alpha}_i \cdot F_i \\ &= 0.303 \cdot F_1 + 0.06 \cdot F_2 + 0.00 \cdot F_3 + 0.38 \cdot F_4 + 0.891 \cdot F_5 + \\ &\quad 2.05 \cdot F_6 + 0.52 \cdot F_7 + 0.00 \cdot F_8 + 0.14 \cdot F_9 + 0.00 \cdot F_{10} \end{aligned} \tag{5.2}$$

Distance/Time proximity-weighted category

$$\begin{aligned}\bar{R}_p &= 0.573 \cdot F_1 + 0.06 \cdot F_2 + 0.00 \cdot F_3 + 0.22 \cdot F_4 + 1.51 \cdot F_5 + \\ &\quad 1.48 \cdot F_6 + 0.34 \cdot F_7 + 0.00 \cdot F_8 + 0.13 \cdot F_9 + 0.00 \cdot F_{10}\end{aligned} \tag{5.3}$$

Visually-weighted category

$$\begin{aligned}\bar{R}_v &= 1.23 \cdot F_1 + 0.31 \cdot F_2 + 0.02 \cdot F_3 + 0.31 \cdot F_4 + 1.00 \cdot F_5 + \\ &\quad 1.14 \cdot F_6 + 0.33 \cdot F_7 + 0.00 \cdot F_8 + 0.03 \cdot F_9 + 0.00 \cdot F_{10}\end{aligned} \tag{5.4}$$

5.5 Evaluating the new result scoring function

To evaluate the quality of a search result, we want to define a metric that captures the usefulness of the ranking to the user. The higher the metric score, the higher the usefulness to the end user. This metric should reward useful websites being ranked high in the search result. Conversely, if useful websites are ranked low, the metric should reflect that as well. As a result, we define a position-weighted quality score as follows:

$$S = \sum_{i=1}^{20} \frac{R_i}{i} \tag{5.5}$$

S is the position-weighted quality score, and R_i is the user's quality score for search result item at position i. Position 1 refers to the 1st search result. This metric essentially calculates the sum of quality score divided by its position in the search result.

In this section, we will analyze the change in search performance based on the three categories of user defined earlier in section 5.3.1.

5.5.1 Data-weighted user category

There are 11 test users who fall into the *data-weighted* category. Each user executed 20 search queries, i.e. 20 position-weighted quality score S. We are interested in seeing how the average quality score would change if the new data-weighted scoring function (equation 5.2) is used to rank the search results instead of the basic two-factor scoring function.

Of these 11 experimental users, 6 of them participated in the re-evaluation of the search result using the new result scoring function. During the experiment, we show these users the modified ranking of results using the same search queries. We then ask them to provide search result quality scores again. The average position-weighted quality score is then calculated based on their response. There appears to be an improvement of approximately 6.9% in search ranking quality. Table 5.3 summarizes the differences.

While a 6.9% improvement does not appear to be a discernible difference, a careful look at the new ranking shows that most high quality results are now ranked near the top. The small increase in percentage is simply a result of the definition of the position-weighted quality score.

5.5.2 Distance/time proximity-weighted user category

A similar analysis is performed for users in the *distance/time proximity-weighted* category. Of the 8 users in this category, 5 of them participated in the search result re-evaluation using the new ten-factor scoring function. Table 5.4 summarizes the result for these users. There is a 2.4% improvement in search relevancy based on improvements in the position-weighted quality score.

Compared to the data-weighted user category, this category of users have a lower improvement in the quality of search result. This can be a result of higher variability in user preferences, as evidenced by the relatively even distribution of weights across the 10 factors (equation 5.3). In other words, while these users do prefer distance/time proximity, their preference is not as strong as the data-oriented users' preference for information. Consequently, it is more difficult for the new scoring function to capture their preferences accurately. Furthermore, Goby already uses the highest ranked factor to rank search results. Hence, any quality improvement will be incremental.

5.5.3 Visually-weighted user category

All 4 users in the visually-weighted category participated in the query re-evaluation experiment. It is interesting to note that the position-weighted quality score improved by 7.0%. This significant improvement in ranking quality may be due to the fact that there are only 4 users in this category. The small number of users allows the regression model to generate an excellent fit to the data points. However, this improvement can be misleading as they are not enough data points to draw meaningful conclusion on the behavior of visually-oriented users. In future iterations of the experiments, it would be helpful to obtain more experimental users who belong to the visually-weighted category.

	Change in quality score for test user #					
	1	8	12	15	17	23
2-factor model score	1.3044	1.4391	1.1484	1.3783	1.2151	1.1862
10-factor model score	1.5397	1.4964	1.2820	1.2473	1.4390	1.1706
% change	18.0%	4.0%	11.6%	-9.5%	18.4%	-1.3%

Table 5.3: The table illustrates the position-weighted quality score for (1) using the basic two-factor scoring function in Goby, and (2) using the *data-weighted* ten-factor model in equation 5.2. There appears to be an improvement in search relevancy by 6.9%. The quality score decreases for user #15 and #23. An explanation for this result could be mis-categorization or bad regression fit for these users.

	Change in quality score for test user #				
	4	13	14	24	26
2-factor model score	1.0726	0.9701	0.9554	1.2275	1.2157
10-factor model score	1.0167	1.0438	0.9214	1.3498	1.2606
% change	-5.2%	7.6%	-3.6%	9.9%	3.7%

Table 5.4: The table shows the position-weighted quality score for the basic two-factor scoring function in Goby and the ten-factor scoring function for *distance/time proximity-weighted users* in equation 5.3. There appears to be an improvement in search relevancy by 2.4%.

	Change in quality score for test user #			
	2	6	16	22
Org score	0.8018	0.9371	0.7241	0.7983
New score	0.6705	1.0255	0.8875	0.8986
% change	-16.4%	9.4%	22.6%	12.6%

Table 5.5: The table shows the position-weighted quality score for the basic two-factor scoring function in Goby and the ten-factor scoring function for *visually-weighted users* in equation 5.4. Search relevancy improves by 7.0%.

5.6 Discussion of results

In the three categories of user, we have observed an improvement in ranking quality by 6.9%, 2.4%, and 7.0% respectively. Table 5.3, 5.4, and 5.5 summarize the results. There are possible explanations for the relatively mild improvement in search quality. First, these numbers could simply be artifacts of the way the position-weighted quality score is defined. If the score is defined differently, it would be not surprising to observe very different quality score improvements. Secondly, it is possible that users are incorrectly categorized using the Euclidean distance method. For example, the quality score for user #15 decreases by 9.5% after using the 10-factor scoring function. If scoring function for distance/time proximity user is used instead, his quality score actually *increases* by 8.1%. In addition, it is also possible that there are simply more than 3 categories of users. By placing dissimilar users into the same bucket, the scoring function generated may be not produce good results. If more experimental users are included, we should be able to observe more distinct user categories emerging. Thirdly, the 20 experimental queries do not have a specified date and utilize only city-level locations. Hence, the two important factors are not very well specified in the queries. In future work, we should consider a bigger set of search queries with different specificity in date and location.

Chapter 6

Regression model with 11 factors

6.1 Adding the 11^{th} factor

In this section, we explore a 11-factor model. We add the additional internal referral factor of Goby as the 11^{th} factor to the original 10-factor model. This metric measures the popularity of a deep web resource and could potentially serve as an indicator of content quality.

6.2 Internal Referral Factor

The *referral* factor is an internal scoring metric used by the Goby search engine. This metric is generated during the deep web crawl process. Since different wrappers may generate an overlapping set of search results, duplicates may appear in the raw result generated by Goby engine. The internal referral score simply measures the number of times a particular deep web resources is mentioned. To a certain extent, this metric is similar to Google's PageRank - it attempts to infer popularity by determining the number of appearance of a particular search result.

The 11-factor model is defined as follows:

$$R = \alpha_0 + \sum_{i=1}^{10} \alpha_i \cdot F_i + \alpha_{11} \cdot F_{11} \tag{6.1}$$

where $F_{11} \subseteq (0...1)$ is the normalized version of the referral factor. This normalization process is crucial for generating meaningful regression weights, as other factors are also upper-bounded by 1.

The goal is to determine the significance of the new factor weight α_{11}. We perform a similar linear regression analysis on 2 queries - "Things to do $\rightarrow$ Nature" [Austin, TX] and "Things to do $\rightarrow$ Places of interest" [San Francisco, CA]. The reason for performing regression on just two queries is because the internal referral score data are available only for these two queries. Nevertheless, as observed in section 5.3, user behavior does not vary greatly across different types of queries. As a result, it is possible to infer some information about the importance of the 11^{th} factor just by analyzing these two queries.

6.3 Regression result

The results of the 11-factor model regression are summarized in table 6.3:

	Factor weights α_i generated by regression model										
Query #	Main Image (real)	Main Image (Google)	More Photos	Distance from Point of Interest	Closeness to search date	Description	Detailed Location	Price	Phone number	More info	Internal referral score
1	0.2983	0.0524	0.0525	0.3805	0.7815	1.1494	0.3660	0.0000	0.2426	0.0000	0.4716
2	0.3294	0.0792	0.0802	0.4351	0.9123	0.9942	0.6318	0.0000	0.1183	0.0000	0.3429

Table 6.1: This table summarizes the factor weights generated after performing 11-factor linear regression. Note the new factor weight α_{11}.

The α_{11} values for the two queries are 0.47 and 0.34 respectively. For query #1, this factor is the 3^{rd} highest ranking factor by factor weight. Nevertheless, its value is not particularly large compared to the two most important factors. This suggests while there is some correlation between the value of the referral factor and the perceived quality of the search result, it is not a particularly good indicator of result content quality. In this project, we obtained referral score data only for two of the 20 queries. In future iterations of multi-factor model, it would make sense to analyze the importance of the 11^{th} factor with more data points.

Chapter 7

Updating Factor Weights

7.1 Motivation

While the universal scoring function would provide an improved result ranking performance over the simple two-factor model, these scoring functions are static and do not adapt to individual user's behavior. Consequently, if a visually-oriented user is initially (perhaps incorrectly) placed in the data-weighted user category, a static scoring function would continue to weight his preferences inaccurately. On the other hand, a dynamically-weighted scoring function allows each user to have an individualized scoring function that adapts intelligently to his personal preference.

In this section, we discuss an algorithm for achieving dynamical factor weighting in the scoring function. In addition, we analyze experimental simulation derived using a Visual Basic model. The dynamic factor weighting combined with the optimized result scoring function should improve search result relevancy for deep web search engines such as Goby.

7.2 Algorithm for updating weights

The algorithm for updating factor weights should achieve two functions - (1) it updates factor weights α_i based on user click activities on the website, and (2) responds to gradual changes in user behavior and preferences. These two requirements suggest the use of a time-decay process that places more emphasis on recent user behavior.

It is possible to apply k-neighbor classification algorithm again to determine the behavior of a user based on his website activity. However, this method simply places the user in one of the three pre-defined categories. It is very possible that a user simply does not fall into any of these categories. If that is the case, a k-neighbor algorithm cannot change the result scoring function to adapt to this user's preference.

7.2.1 Basic update model

A user may click on some of the result links when he is going to a list of search results. For every result item the user clicked on, the update engine would analyze the set of features for this item. For example, the engine sees if the result item contains images, description, detailed location or prices etc. By analyzing the set of features, the engine can guess what the user preferences are. These features are simply defined by the 10 factors, F_i, from the model earlier.

User behavior is analyzed in batches of website activity (e.g. clicks on search result links). Each batch should have at least 10 data points for meaningful calculations. Analysis is done in batches because it is computational expensive to update factor weights every time an user action is registered.

For each batch of user activity on the site, the algorithm computes the percentage of items that carries the feature outlined by factor i. If a user exhibits behavior of a visually-oriented person, it should not be surprising to see that a large percentage of result items he clicks on contains visual elements such as images or videos.

To update factor weights in the result scoring function, the algorithm compares the percentage of user activity that contains feature i against the statistical average percentage that feature. For example, if 90% of items a user clicks on contains a "Description" (compared to an site-wide average of 65%), it is very likely that this user values the "Description" feature more than other users. Consequently, the weight for the "Description" factor should be increased accordingly to adapt to this behavior. In addition, the total change in the result scoring function should be gradual. As a result, the total cumulative percentage change across all factor weights should be upper-bounded by a threshold value T.

We define the fractional change in factor weight $\Delta\alpha_i$ as follows:

$$\Delta\alpha_i = T \cdot \frac{P_i - \bar{P}_i}{\sum_{i=1}^{10} |P_i - \bar{P}_i|} \tag{7.1}$$

where T is the threshold value, P_i is the percentage of feature i contained in user activity, and $\bar{P}_i$ is the historical average click-rate of feature i.

This dynamic weight update algorithm is computationally simple and easy to implement. In addition, it adapts to user behavior on the fly. One potential problem is that a user can be biased towards clicking on links that are ranked high by the default algorithm. Consequently, the engine is not capturing the true user preference, but rather user preference that is skewed by the order at which results are presented. It is possible to remove some of this bias with techniques such as stimulated annealing [15].

7.3 Experimentation

A Visual Basic simulation model is designed to observe how factor weights evolve under certain conditions in the long-run. To fully evaluate the performance of the factor weight update algorithm, it requires running the algorithm for an extended period of time. This is particularly difficult on the Goby website, since there is a relatively small number of registered users. Furthermore, most users perform searches without ever logging-in. Without the ability to relate a user with a result scoring function, it is difficult to run long-term experiments on the website.

The experiment tests the following two scenario to see how the factor weights evolve over time:

- **Incorrect categorization**: A user is initially categorized as data-oriented user, but his behavior switches to that of a visually-oriented user.

- **Gradual change in user behavior**: A user is initially categorized as a data-oriented user and behave similar to one for a period time. The user then switches to behave as as visually-oriented user.

The Visual Basic code essentially simulates the behavior of a data-weighted and a visually-weighted user behavior by using their result scoring functions to evaluate search quality. The simulation would choose to click on a result if it scores in the top 50% of the top 20 search results. The historical click rate of feature i (required $\bar{P_i}$ terms in equation 7.1) are estimates using experimental data from chapter 4. During some of the latter experiments, we record which links the users click on during the search process. The click data is then linked to see what features are contained in these result items. These numbers are summarized in table 7.1.

Historical average percentages $\bar{P_i}$									
$\bar{P_1}$	$\bar{P_2}$	$\bar{P_3}$	$\bar{P_4}$	$\bar{P_5}$	$\bar{P_6}$	$\bar{P_7}$	$\bar{P_8}$	$\bar{P_9}$	$\bar{P_{10}}$
5.0%	25%	1.0%	7.5%	10%	25%	5%	5%	5%	5%

Table 7.1: This table summarizes the historical click-rate percentages of the 10 factors. These are *estimated* values based on experimental data.

For the *incorrect categorization* scenario, the code starts the simulation with factor weights of the data-weighted scoring function. It utilizes a threshold value of T=10%, i.e. total cumulative change in factor weights is less than 10%. Furthermore, factor re-weighting is triggered every time the user has clicked on 10 search result items. User click actions are simulated by three test users from the previous experiments.

Figure 7.3 illustrates the change in result quality using this weight re-balancing algorithm. The result suggests that there is an initial ramp-up period when the weights are slowly being re-balanced. However, once past the initial ramp-up, search result quality improves as more relevant results are being pushed to the top. Consequently, the user behaves more and more like a visually-oriented user. The factor weights get re-balanced quickly to become a visually-weighted scoring function. Figure 7.3 shows the gradual change in search result quality.

Similar to the first scenario, the *gradual change* scenario also has an initial ramp-up period during which factor weights are slowly being re-balanced. This may be a good characteristic of the algorithm as changes are relatively gradual. However, if

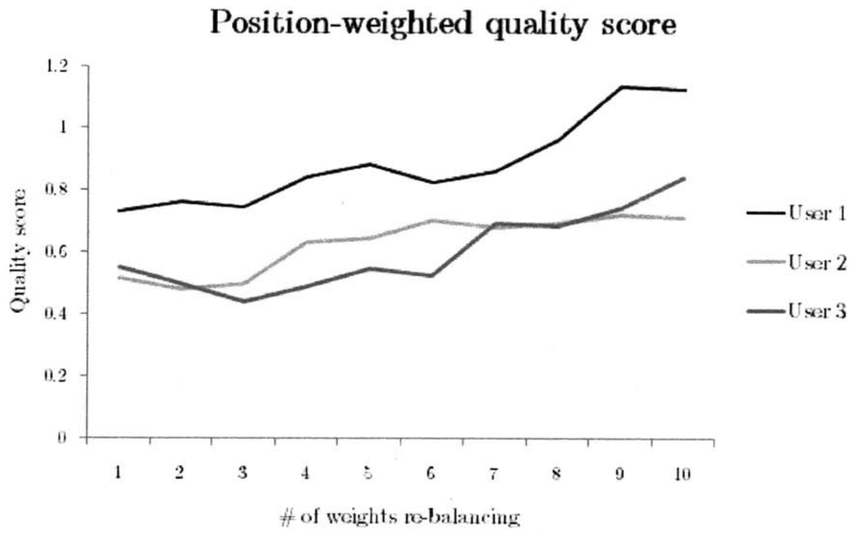


Figure 7-1: Figure showing the change in position-weighted quality score after several re-balancing of weights for scenario one. There is a slow initial ramp-up when weights are being re-balanced slowly. Once the ranking quality has improved, it becomes a positive cycle as users click on more useful items.

user behavior changes frequently and rapidly, then this algorithm may have difficulty keeping up with the pace of change to provide good result ranking. Nevertheless, the simulation indicates a strong possibility for improvements in search quality by applying the dynamic weighting algorithm. In future work, we would like to implement this algorithm in the Goby system and to observe how search quality changes in the long-run.

Chapter 8

Conclusion

The task of ranking deep web search result is an area of significant importance as it represents a crucial challenge in ensuring the quality of search result. In this research project, we have analyzed user preferences for evaluating the quality of search result. 600 experimental search queries are conducted on the Goby deep web search engine with 30 test subjects. Our goal is to develop a ranking algorithm that can be used for ranking deep web search results.

In this project, we model user behavior using a 10-factor regression model, where we have defined these factors based on observations during the experiments. We have noticed strong preferences for information (*description* and *business supplied image* factor) and proximity in time (*time metric*). We have also observed the categorization of user behavior. In particular, we defined 3 categories of users - data-weighted, distance/time proximity-weighted, and visually-weigthed users. In addition, we have also noticed that users behavior do not vary greatly across the different categories of search queries. Users appear to demonstrate strong and consistent preferences for the description and time metric factors.

Using this categorization of users, we created 3 universal scoring functions. These scoring functions yielded a measurable improvement in search quality. Furthermore, we discussed a dynamic factor weighting algorithm that changes factor weights based on user actions on the website. This algorithm suggests a promising solution for adapting the scoring function to individual user preferences.

While our results are encouraging, a more detailed study should paint a more comprehensive picture of user behavior. In particular, it would be interesting to expand the set of experimental queries to include queries with different degrees of specificity in time and location. Furthermore, we hope to include a more extensive study on using the category hierarchy metric and the referral metric for scoring search results.

Appendix A

Source code

```
public class LRC {
    public static void main(String[] args) {
        int MAXN = 1000;
        int n = 0;
        double[] x = new double[MAXN];
        double[] y = new double[MAXN];

        // first pass: read in data, compute xbar and ybar
        double sumx = 0.0, sumy = 0.0, sumx2 = 0.0;
        while(!StdIn.isEmpty()) {
            x[n] = StdIn.readDouble();
            y[n] = StdIn.readDouble();
            sumx  += x[n];
            sumx2 += x[n] * x[n];
            sumy  += y[n];
            n++;
        }
        double xbar = sumx / n;
        double ybar = sumy / n;

        // second pass: compute summary statistics
        double xxbar = 0.0, yybar = 0.0, xybar = 0.0;
        for (int i = 0; i < n; i++) {
            xxbar += (x[i] - xbar) * (x[i] - xbar);
            yybar += (y[i] - ybar) * (y[i] - ybar);
            xybar += (x[i] - xbar) * (y[i] - ybar);
        }
        double beta1 = xybar / xxbar;
        double beta0 = ybar - beta1 * xbar;
        // print results
        System.out.println("y   = " + beta1 + " * x + " + beta0);

        // analyze results
        int df = n - 2;
        double rss = 0.0;      // residual sum of squares
        double ssr = 0.0;      // regression sum of squares
        for (int i = 0; i < n; i++) {
            double fit = beta1*x[i] + beta0;
            rss += (fit - y[i]) * (fit - y[i]);
            ssr += (fit - ybar) * (fit - ybar);
        }
        double R2    = ssr / yybar;
        double svar  = rss / df;
        double svar1 = svar / xxbar;
        double svar0 = svar/n + xbar*xbar*svar1;
        System.out.println("R^2                 = " + R2);
        System.out.println("std error of beta_1 = " + Math.sqrt(svar1));
        System.out.println("std error of beta_0 = " + Math.sqrt(svar0));
        svar0 = svar * sumx2 / (n * xxbar);
        System.out.println("std error of beta_0 = " + Math.sqrt(svar0));

        System.out.println("SSTO = " + yybar);
        System.out.println("SSE  = " + rss);
        System.out.println("SSR  = " + ssr);
    }
}



'****************************************************************************
'* Subroutine executes a multiple linear regression analysis.              *
'*                                                                          *
'* The code has been adopted from Chad A. Rankin 2007 original project     *
'* http://www.freevbcode.com/ShowCode.asp?ID=9070
'****************************************************************************

Private Sub OK_Btn_Click()
Dim Missing As Variant, temp As Variant, Varnames As Variant, ref As String
Dim ShtName As String, FinalCol As Double, Time As Double, Data As Variant
Dim cntsheets, newsheet As Worksheet, p As Double, wholestring As Variant
Dim partstring As Variant, Model As String, n As Variant, wks As Variant
Dim i As Double, j As Double, ret_err As Boolean, Intercept As Variant

    On Error GoTo EndProc
    Application.Calculation = xlCalculationManual

    'Start the timer
    Time = Timer

'****************************
' Import data from listboxes
'****************************

    If Yvar.ListCount = 0 Or Xvars.ListCount = 0 Then
        Me.Hide
        MsgBox Prompt:="Please enter data to analyze.", _
                      Buttons:=48, Title:="Input Error!"
        VariableList.Clear  'Starting over-clear contents
        Me.Show
        Exit Sub
    End If

    If Xvars.ListCount > 50 Then
        Me.Hide
        MsgBox Prompt:="Procedure accepts no more than 50 predictors.", _
                      Buttons:=48, Title:="Input Error!"
        Me.Show
    End If

    'Find the last column with first row non empty
    FinalCol = Application.Cells(1, 255).End(xlToLeft).Column

    'Find the response variable and assign the reference
    For j = 1 To FinalCol
        If Cells(1, j) = Yvar.List(i) Then
            wholestring = Range(Cells(1, j), Cells(1, j)).AddressLocal
            partstring = Split(wholestring, "$")
            ref = "$" & partstring(1) & ":$" & partstring(1)
            Exit For
        End If
    Next j

    'Find the x variables and assemble a complete reference
    For i = 0 To Xvars.ListCount - 1
        For j = 1 To FinalCol
            If Cells(1, j) = Xvars.List(i) Then
                wholestring = Range(Cells(1, j), Cells(1, j)).AddressLocal
                partstring = Split(wholestring, "$")
                ref = ref & ", $" & partstring(1) & ":$" & partstring(1)
                Exit For
            End If
        Next j
    Next i

    Call Progress(0.1)    'update procedure's progress

    'Prepare Data: submit data matrix to be prepared
    Call Range_Prep(ref, temp, Varnames, ret_err)
    If ret_err = True Then GoTo EndProc             'error with range reference

    Call Progress(0.35)    'update procedure's progress

    'Remove any observations with missing values-response included
    'Variable "missing" is returned as an array containing the _
     observation numbers removed (if any).  The count of the array _
     is the number of rows removed.

    Call Remove_Missing(temp, Missing)
    n = UBound(temp, 1)
    p = UBound(temp, 2)

    Call Progress(0.55)    'update procedure's progress

    'Set value of intercept variable
    If ckb_INT.Value = True Then
        Intercept = 1#
    Else
        Intercept = 0#
    End If
    p = p - 1 + Intercept   'p = count of predictors including intercept

    'If intercept is specified, add to data matrix
    ReDim y(1 To n, 1 To 1), Data(1 To n, 1 To p)
    For i = 1 To n
        y(i, 1) = temp(i, 1)
        If Intercept = 1 Then Data(i, 1) = 1#
        For j = 1 + Intercept To p
            Data(i, j) = temp(i, j + 1 - Intercept)
        Next j
    Next i

    Call Progress(0.65)    'update procedure's progress

'****************************
```

```
' Regression Model Statement
'****************************

    Model = "= "
    Model = Model & """Y"""
    Model = Model & " & "
    Model = Model & """ = """

    'The information for the model statement is taken from the _
     worksheet and not hard coded.
    For i = 1 To p
        If Intercept = i Then
            temp = " & " & "round(B19,2)"
        Else
            temp = " & if(sign(B" & 18 + i & ")=-1, "" "","" + "")" & _
                   " & " & "Round(B" & 18 + i & ", 2)" & " & " & _
                   """ """ & " & " & "A" & 18 + i
        End If
        Model = Model & temp
    Next i

    Call Progress(0.75)   'update procedure's progress


'********************************************************************************
'******************************  OUTPUT  ****************************************
'********************************************************************************

    'Output in new worksheet
    'Check workbook for a worksheet named "Regression"
    For Each wks In Application.Worksheets
        If wks.name = "Regression" Then wks.Delete
    Next

    'Place new worksheet after the last worksheet in the workbook
    cntsheets = Application.Sheets.Count
    Set newsheet = Application.Worksheets.add(after:=Worksheets(cntsheets))
    newsheet.name = "Regression"
    FinalCol = 0

    Call Progress(0.8)    'update procedure's progress

    'Get the sheet name-either new or existing
    ShtName = Application.ActiveSheet.name
```

```
With Application
    'Place the data in the worksheet along with variable names
    .Cells(1, 13 + p).Value = Varnames(1)
    .Range(Cells(2, 13 + p), Cells(n + 1, 13 + p)).Value = y
    For i = 1 To p
        If Intercept = i Then
            .Cells(1, 13 + p + i).Value = "Intercept"
        Else
            .Cells(1, 13 + p + i).Value = Varnames(i + 1 - Intercept)
        End If
    Next i
    .Range(Cells(2, 14 + p), Cells(n + 1, 13 + p + p)) = Data

    'Insert the range formula for the X'Xinv
    .Cells(1, 8).Value = "X'X inverse"
    .Range(Cells(2, 8), Cells(1 + p, 7 + p)).FormulaArray = _
            "=MINVERSE(MMULT(TRANSPOSE(RC[" & 6 + p & _
            "]:R[" & n - 1 & "]C[" & 5 + p + p & "]),RC[" & 6 + p & _
            "]:R[" & n - 1 & "]C[" & 5 + p + p & "]))"

    'Insert the fomulae for the variance-covariance matrix
    .Cells(2 + p, 8).Value = "Variance-covariance matrix"
    .Range(Cells(3 + p, 8), Cells(p * 2 + 2, 7 + p)).FormulaR1C1 = _
                                "=R8C4*R[-" & 1 + p & "]C"

    'Build the correlation matrix using the 'Correl' function
    'Must apply the function to all combinations to get lower _
     triangular of correlation matrix--get other half by symmetry
    .Cells(3 + 2 * p, 8).Value = "Correlation matrix"
    For i = 1 To p - Intercept
        .Cells(3 + 2 * p + i, 7 + i).Value = 1#
        For j = i + 1 To p - Intercept
            .Cells(3 + 2 * p + j, 7 + i).FormulaR1C1 = _
                "=Correl(R2C" & 13 + p + Intercept + i & _
                ":R" & n + 1 & "C" & 13 + p + Intercept + i & "," & _
                "R2C" & 13 + p + Intercept + j & ":R" & n + 1 & "C" & _
                13 + p + Intercept + j & ")"
            .Cells(3 + 2 * p + i, 7 + j).FormulaR1C1 = _
                "=R[" & j - i & "]C[-" & j - i & "]"
        Next j
    Next i

    'Calculate the inverse of the correlation matrix
    Cells(4 + 2 * p + p - Intercept, 8).Value = "Inverse Correlation Matrix"
    .Range(Cells(5 + 2 * p + p - Intercept, 8), _
    Cells(4 + 2 * p + 2 * (p - Intercept), 7 + p - Intercept)).FormulaArray = _
            "=MINVERSE(R" & 4 + 2 * p & "C8:R" & 3 + 2 * p + p - Intercept & _
            "C" & 7 + p - Intercept & ")"

Call Progress(0.85)    'update procedure's progress

'Output ANOVA table
.Cells(1, 1).Value = "Regression Analysis of " & Varnames(1)
.Cells(1, 1).Font.Bold = True
.Cells(3, 1).Value = "Regression equation:"
On Error Resume Next
.Cells(3, 2).Value = Model
On Error GoTo 0
.Cells(5, 2).Value = "Sum of"
.Cells(5, 3).Value = "Degrees of"
.Cells(5, 4).Value = "Mean"
.Cells(6, 1).Value = "Source of Variation"
.Cells(6, 2).Value = "Squares"
.Cells(6, 3).Value = "Freedom"
.Cells(6, 4).Value = "Square"
.Cells(6, 5).Value = "F"
.Cells(6, 6).Value = "P-value"

'Output fitted values
.Cells(1, 9 + p).Value = "Fits"
.Range(Cells(2, 9 + p), Cells(n + 1, 9 + p)).FormulaR1C1 = _
        "=MMULT(RC[5]:RC[" & 4 + p & "],R19C2:R" & 18 + p & "C2)"

'Output residuals
.Cells(1, 10 + p).Value = "Resids"
.Range(Cells(2, 10 + p), Cells(n + 1, 10 + p)).FormulaR1C1 = _
        "=RC[3]-RC[-1]"

'Output regression sum of squares
.Cells(7, 1).Value = "Regression"
.Cells(7, 2).FormulaR1C1 = "=R[2]C-R[1]C"
.Cells(7, 3).Value = p - Intercept
.Cells(7, 4).FormulaR1C1 = "=RC[-2]/RC[-1]"
.Cells(7, 5).FormulaR1C1 = "=RC[-1]/R[1]C[-1]"

'Output error sum of squares
.Cells(8, 1).Value = "Error"
.Cells(8, 2).FormulaR1C1 = _
    "=SUMSQ(R2C" & 10 + p & ":R" & n + 1 & "C" & 10 + p & ")"
.Cells(8, 3).Value = n - p
```

```
.Cells(8, 4).FormulaR1C1 = "=RC[-2]/RC[-1]"

'Output total sum of squares
.Cells(9, 1).Value = "Total"
If Intercept = 1 Then
    .Cells(9, 2).FormulaR1C1 = _
        "=DEVSQ(R2C" & 13 + p & ":R" & n + 1 & "C" & 13 + p & ")"
Else
    .Cells(9, 2).FormulaR1C1 = _
        "=SUMSQ(R2C" & 13 + p & ":R" & n + 1 & "C" & 13 + p & ")"
End If

'Output error degrees of freedom
.Cells(9, 3).Value = n - Intercept

'Output RMSE
.Cells(11, 2).Value = "s"
.Cells(11, 3).FormulaR1C1 = "=SQRT(R[-3]C[1])"
.Cells(11, 3).NumberFormat = "0.0000"

'Output Rsq only with intercept model
If Intercept = 1 Then
    .Cells(12, 2).Value = "R-sq"
    .Cells(12, 3).FormulaR1C1 = "=R[-5]C[-1]/R[-3]C[-1]"
    .Cells(12, 3).NumberFormat = "0.00%"
    .Cells(13, 2).Value = "R-Sq(adj)"
    .Cells(13, 3).FormulaR1C1 = "=1-R8C4/(R9C2/R8C3)"
    .Cells(13, 3).NumberFormat = "0.00%"
End If

'Output table of coefficient estimates, etc.
.Cells(16, 1).Value = "Parameter Estimates"
.Cells(18, 1).Value = "Predictor"
.Cells(18, 2).Value = "Coef Est"
.Cells(18, 3).Value = "Std Error"
.Cells(18, 4).Value = "t value"
.Cells(18, 5).Value = "P-value"

'General formatting
'Draw lines on ANOVA table
Range(.Cells(4, 1), Cells(4, 6)) _
        .Borders(xlEdgeBottom).LineStyle = xlContinuous
Range(.Cells(6, 1), Cells(6, 6)) _
        .Borders(xlEdgeBottom).LineStyle = xlContinuous
Range(.Cells(9, 1), Cells(9, 6)) _
```

```
        .Borders(xlEdgeBottom).LineStyle = xlContinuous
'Draw line for table of coefs, se, VIFs, t & p statistics
Range(.Cells(18, 1), Cells(18, 5)) _
        .Borders(xlEdgeBottom).LineStyle = xlContinuous

.Columns(1).ColumnWidth = 18
.Columns(3).ColumnWidth = 11
.Columns(4).ColumnWidth = 9.75
.Range(Cells(7, 5), Cells(7, 6)).NumberFormat = "0.0000"

'Output the coefficient estimates
.Range(Cells(19, 2), Cells(18 + p, 2)).FormulaArray = _
    "=MMULT(R2C8:R" & 1 + p & "C" & 7 + p & _
    ",MMULT(TRANSPOSE(R2C" & 14 + p & ":R" & _
    n + 1 & "C" & 13 + 2 * p & ")," & _
    "R2C" & 13 + p & ":R" & n + 1 & "C" & 13 + p & "))"


'Ouput SEs, t values, pvalues, and VIFs
For i = 1 To p
    'Output variable names
    If i = 1 Then
        If Intercept = 1 Then
            .Cells(i + 18, 1).Value = "Constant"
        Else
            .Cells(i + 18, 1).Value = Varnames(i + 1)
        End If
    Else
        .Cells(i + 18, 1).Value = Varnames(i)
    End If
    .Cells(i + 18, 2).NumberFormat = "0.0000"

    'Output standard errors
    .Cells(i + 18, 3).FormulaR1C1 = _
        "=SQRT(R" & 2 + p + i & "C[" & 4 + i & "])"
    .Cells(i + 18, 3).NumberFormat = "0.0000"
    .Cells(i + 18, 4).NumberFormat = "0.0000"
    .Cells(i + 18, 5).NumberFormat = "0.0000"

    'Output VIFs
    If i > 1 And Intercept = 1 And p > 2 Then
        .Cells(18, 6) = "VIFs"
        .Cells(i + 18, 6).FormulaR1C1 = _
                "=R" & 3 + 3 * p - Intercept + i & "C[" & i & "]"
        .Cells(i + 18, 6).NumberFormat = "0.0000"
```

```
            .Cells(18, 6).Borders(xlEdgeBottom) _
                                    .LineStyle = xlContinuous
        End If
    Next i

    Call Progress(0.9)    'update procedure's progress

    'Write note detailing the use of observations
    If IsEmpty(Missing) Then
        .Cells(i + 19, 1) = n & _
            " observations were used in the analysis."
    Else
        .Cells(i + 19, 1) = n & _
            " observations were used in the analysis."
        'Two statements to get the verb tense correct
        If UBound(Missing, 1) = 1 Then
            .Cells(i + 20, 1) = UBound(Missing, 1) & _
                " observation was excluded due to missing values."
        Else
            .Cells(i + 20, 1) = UBound(Missing, 1) & _
                " observations were excluded due to missing values."
        End If
    End If

'****************************
' Diagnostic Calculations
'****************************

    'Output the value of the determinant of the correlation matrix
    .Cells(11, 4) = "Determinant"
    .Range(Cells(11, 5), Cells(11, 5)).FormulaArray = _
                        "=MDETERM(R" & 4 + 2 * p & _
                        "C8:R" & 3 + 2 * p + p - Intercept & _
                        "C" & 7 + p - Intercept & ")"


    Call Progress(0.95)    'update procedure's progress

    'Durbin-Watson statistic
    .Cells(1, 11 + p).Value = "Durbin-Watson"
    .Range(Cells(3, 11 + p), Cells(n + 1, 11 + p)).FormulaR1C1 = _
            "=(RC[-1]-R[-1]C[-1])^2"
    .Cells(12, 4) = "DW"
    .Cells(12, 5).FormulaR1C1 = _
                "=SUM(R3C" & 11 + p & ":R" & n + 1 & "C" & 11 + p & ")/R8C2"
        .Cells(12, 5).NumberFormat = "0.00"

        'Output processing time
        .Worksheets(ShtName).Cells(22 + p, 1) = _
            "Computational time: " & .Round(Timer - Time, 2) & " seconds."

        Call Progress(1)    'update procedure's progress

        'resume worksheet calculations
        .Calculation = xlCalculationAutomatic

        'Calculate probabilities after worksheet calculations _
         have been set to automatic
        't values
        .Range(Cells(19, 4), Cells(18 + p, 4)). _
                        FormulaR1C1 = "=RC[-2]/RC[-1]"
        'p values
        .Range(Cells(19, 5), Cells(18 + p, 5)). _
                        FormulaR1C1 = "=(1-TCDF(abs(RC[-1]),R8C3))*2"
        'F value
        .Cells(7, 6).FormulaR1C1 = "=1-FCDF(RC[-1],RC[-3],R[1]C[-3])"

    End With


Unload Me
End

EndProc:
Application.Calculation = xlCalculationAutomatic    'resume worksheet calculations
MsgBox ("Procedure has encountered a fatal error and will terminate.  " & _
         "Error code: " & Err)

Unload Me
End Sub

Sub Progress(Pct)
'This sub updates the width of the bar moving across the _
 progress indicator frame and the % complete caption
   With Me
      .Progress_Frame.Caption = FormatPercent(Pct, 0)
      .ProgressBar.Width = Pct * .Progress_Frame.Width
      .Repaint
   End With
End Sub
```

Appendix B

Euclidean distance matrix

Euclidean distance matrix for all (i,j) pairs

	Test #																													
Test #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1	0.00	3.48	N/A	3.98	1.50	3.83	1.99	1.43	N/A	4.56	1.27	1.47	4.87	4.65	1.89	3.54	1.80	3.80	N/A	1.81	N/A	3.12	1.34	4.89	4.57	4.54	N/A	1.81	N/A	N/A
2		0.00	N/A	3.01	4.57	1.29	4.51	4.33	N/A	3.14	4.61	4.39	4.70	3.31	3.47	1.02	3.77	4.51	N/A	3.33	N/A	1.55	3.96	4.10	3.88	4.07	N/A	4.67	N/A	N/A
3			0.00	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
4				0.00	3.01	3.50	4.22	3.71	N/A	1.85	4.77	3.75	1.32	1.50	3.91	4.91	3.23	1.75	N/A	3.25	N/A	4.35	4.18	1.61	1.60	1.17	N/A	3.05	N/A	N/A
5					0.00	3.41	1.67	1.53	N/A	4.96	1.03	1.64	4.90	3.65	1.14	3.47	1.91	4.34	N/A	1.56	N/A	4.09	1.01	4.49	3.45	3.74	N/A	1.13	N/A	N/A
6						0.00	4.80	3.83	N/A	3.84	3.26	4.72	4.73	3.98	4.00	1.95	3.18	4.40	N/A	3.28	N/A	1.41	4.00	4.05	3.23	4.00	N/A	4.80	N/A	N/A
7							0.00	1.37	N/A	3.55	1.06	1.41	4.53	4.65	1.70	3.02	1.81	3.12	N/A	1.93	N/A	3.95	1.75	3.55	3.52	3.17	N/A	1.54	N/A	N/A
8								0.00	N/A	3.21	1.05	1.83	3.08	4.10	1.51	4.99	1.35	4.94	N/A	1.09	N/A	3.48	1.33	3.10	3.24	4.34	N/A	1.06	N/A	N/A
9									0.00	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
10										0.00	3.11	4.71	1.61	1.64	4.27	4.60	3.67	1.12	N/A	3.63	N/A	3.00	3.67	1.82	1.62	1.52	N/A	3.11	N/A	N/A
11											0.00	1.31	4.75	4.27	1.18	4.66	1.68	3.64	N/A	1.54	N/A	4.08	1.06	4.66	4.59	4.57	N/A	1.28	N/A	N/A
12												0.00	3.19	3.90	1.38	4.70	1.74	3.57	N/A	1.53	N/A	3.40	1.41	4.27	4.44	4.14	N/A	1.25	N/A	N/A
13													0.00	1.27	4.42	3.37	4.06	1.65	N/A	4.79	N/A	3.86	3.59	1.46	1.55	1.87	N/A	4.07	N/A	N/A
14														0.00	3.13	4.07	4.21	1.58	N/A	3.95	N/A	3.61	3.66	1.77	1.17	1.01	N/A	3.02	N/A	N/A
15															0.00	3.75	1.35	3.96	N/A	1.01	N/A	3.37	1.99	4.71	4.62	3.10	N/A	1.80	N/A	N/A
16																0.00	3.85	3.35	N/A	3.92	N/A	1.88	3.15	4.28	4.40	4.93	N/A	3.12	N/A	N/A
17																	0.00	3.28	N/A	1.09	N/A	3.02	1.90	3.52	3.47	3.67	N/A	1.19	N/A	N/A
18																		0.00	N/A	3.61	N/A	3.14	4.73	1.88	1.56	1.11	N/A	4.19	N/A	N/A
19																			0.00	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
20																				0.00	N/A	3.80	1.70	3.85	3.46	3.34	N/A	1.21	N/A	N/A
21																					0.00	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
22																						0.00	3.12	3.08	3.45	4.35	N/A	3.96	N/A	N/A
23																							0.00	3.09	4.03	4.03	N/A	1.23	N/A	N/A
24																								0.00	1.80	1.02	N/A	4.96	N/A	N/A
25																									0.00	1.32	N/A	3.78	N/A	N/A
26																										0.00	N/A	4.09	N/A	N/A
27																											0.00	N/A	N/A	N/A
28																												0.00	N/A	N/A
29																													0.00	N/A
30																														0.00

Bibliography

[1] Sergey Brin and Lawrence Page. The anatomy of a largescale hypertextual web search engine. *ACM*, 1998.

[2] Romain Thibaux Carlos Guestrin, Peter Bodik. Distributed regression: an efficient framework for modeling sensor network data. *ACM/IEEE International Conference on Information Processing in Sensor Networks*, 2004.

[3] S. et al. Chang. Contextaware wrapping: Synchronized data extraction. *VLDB*, 2007.

[4] Thomas M. Cover and Peter E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 1967.

[5] E. Davies. *Machine Vision: Theory, Algorithms and Practicalities.* Four volumes. Academic Press, 1990–90. Reference for k-Neighbor.

[6] et al. Dhamankar, R. Imap: Discovering complex mappings between database schemas. *SIGMOD*, 2004.

[7] P. Domingos Doan, A. and A. Halevy. Reconciling schemas of disparate data sources: A machine learning approach. *SIGMOD*, 2001.

[8] et al. Dobbins, Pete. Morpheus 2.0: A data transformation management system. *ACM VLDB*, 2007.

[9] et al. Gannon, T. Semantic information integration in the large: Adaptability, extensibiility, and scalability of the context mediation approach. *MIT DSpace*, 2006.

[10] Francois Glineur. Convex optimization. Lecture Notes 1, eVita School, 2009.

[11] et al. He, Bin. Accessing the deep web. *Communications of the ACM 50.5*, 2007.

[12] Matthew Hurst. Back to the future: Nlp, search, google and powerset. *Microsoft Research Scientists*, 2007.

[13] Alon Halevy Jayant Madhavan, Alex Rasmussen. Google's deep web crawl. *Proceedings of the VLDB Endowment*, 2008.

[14] Adeline Nazarenko Julien Deriviere, Thierry Hamon. A scalable and distributed nlp architecture for web document annotation. *Advances in Natural Language Processing*, 2006.

[15] M. P. Vecchi Kirkpatrick, S.; C. D. Gelatt. Optimization by simulated annealing. *Science*, 1983.

[16] Donald E. Knuth. *The Deep Web: Surfacing Hidden Value.*, volume 1 of *Four volumes*, section 7.1. Addison-Wesley, second edition, 2001. Deep web search research.

[17] Nikos Vlassis Likas, Aristidis and Jakob Verbeek. The global k]means clustering algorithm. *Pattern Recognition*, 2003.

[18] Jimmy Lin. Scalable language processing algorithms for the masses: A case study in computingword co-occurrence matrices with mapreduce. *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, 2008.

[19] R.J. Oosterbaan. Regression analysis. Book chapter 2, ILRI Publication, 1994.

[20] Denis Shestakov. Search interfaces on the web: Querying and characterizing. *Communications of the ACM*, 2008.

[21] A. Sheth and J. A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM COMputing Surveys*, 1990.

[22] Michael Stonebraker. Integrating the deep web with the shallow web. *Cambridge, MA*, 2007.

[23] Yitong Wang and Masaru Kitsuregawa. Link based clustering of web search results. *Lecture Notes in Computer Science*, 2001.

[24] Alex Wright. Exploring a 'deep web' that google can't grasp. *Wall Street Journal Report*, 2009.