

Search Using Social Networks

by

Ammar Ammar

S.B., Massachusetts Institute of Technology (2009)

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

Master of Engineering in Electrical Engineering and Computer Science

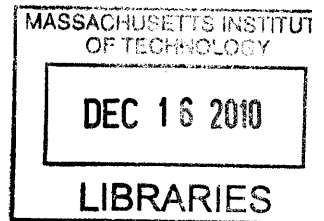
at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2010

© Massachusetts Institute of Technology 2010. All rights reserved.

ARCHIVES



Author
Department of Electrical Engineering and Computer Science
August 20, 2010

Certified by
Devavrat Shah
Jamieson Career Development Associate Professor of Electrical Engineering and
Computer Science
Thesis Supervisor

Accepted by
Dr. Christopher J. Terman
Chairman, Department Committee on Graduate Theses

Search Using Social Networks

by

Ammar Ammar

Submitted to the Department of Electrical Engineering and Computer Science
on August 20, 2010, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In this thesis, we present an approach to the problem of personalized web search which makes use of the searcher's social network, in addition to the hyper-link based score used in most search engines. This combination of social and absolute search scores aims to improve the visibility of information that is relevant to the searcher, while maintaining any absolute measures of document importance. In our approach we adopt a flexible framework for combining information from different sources using Rank Aggregation techniques. Our search system, implemented using Java and Python, covers all the events and web pages present on MIT owned websites. We discuss the theory, design, and implementation of this system in details.

Thesis Supervisor: Devavrat Shah

Title: Jamieson Career Development Associate Professor of Electrical Engineering and Computer Science

Acknowledgments

I gratefully acknowledge the contribution of my advisor, Devavrat Shah, my group-mates in the Inference and Stochastic Networks group, and members of the Laboratory of Information and Decision Systems community, especially professor Munther Dahleh, and Alireza Tahbaz-Salehi for their advice, patience, and thoughtful feedback.

This work was supported by a Xerox Fellowship.

Contents

1	Introduction	13
1.1	Motivating Example	15
1.2	System Components	16
1.3	Outline	17
2	System Details and Algorithms	19
2.1	Front-end Interface	19
2.2	Remote Procedure Calls	23
2.3	Back-end Server	23
2.3.1	Crawling and Basic Indexing	24
2.3.2	Social Tagging	25
2.3.3	The Enhanced Index	26
2.3.4	The Search Server	27
2.3.5	The Query Parser	28
2.3.6	The Ranking Box	28
2.3.7	The Rank Aggregator	33
2.3.8	Caching	35
2.3.9	Logging	35
2.4	Scalability	35
3	Performance and Evaluation	37
3.1	System Performance	37
3.2	Search Quality	40

4	Related and Prior Work	47
4.1	Absolute Search	47
4.2	Personalized Search	49
4.3	Social Search	49
4.4	Rank Aggregation	52
5	Future Directions	55
5.1	Coverage	55
5.2	Scalability	56
5.3	More Personalized Search	56
5.4	Privacy	56
5.5	Rank Aggregation	57
5.6	Learning	57

List of Figures

1-1	Architecture of our social network based search engine	17
2-1	Search Box and Adjustment Slider	20
2-2	Search Results	21
2-3	Front-end Components	22
2-4	The Basic Index	24
2-5	The Enhanced Index	27
2-6	Back-end Components	36
3-1	Query, index access, and scoring times, in seconds.	38
3-2	Query Time, in seconds.	39
3-3	Query Time for various indices, in seconds.	42
3-4	Search Results for “annealing” from LIDS	43
3-5	Search Results for “annealing” from MTL	44
3-6	Search Results for “cooperation” from LIDS	45

List of Tables

2.1 Query Syntax	28
----------------------------	----

Chapter 1

Introduction

Of all the challenges created by the World Wide Web (WWW), it can be confidently said that the problem of web search is one that has been successfully solved. A statement which should not be taken lightly given the enormous size, rate of growth, and heterogeneity of the web, and the highly subjective nature of the task at hand. The search engines built and maintained by Google, Yahoo!, and Microsoft provide some of the most successful examples of such solutions, and the core of a multi-billion dollar industry. For the most part, these search engines automatically crawl the web for accessible documents¹, build an index of these documents, and respond to user queries with lists of these documents selected using the query, and ranked using certain ranking functions, or algorithms. These lists can contain text documents, images, videos, and other types of files.

Since the influential work of Kleinberg [1] [2] [3] and Brin and Page [4] [5], a common theme running through most ranking algorithms is the focus on the hyper-link structure of the web to deduce information about the popularity of certain documents. For that purpose, the web is modeled as a directed graph, and each hyperlink between anyx documents is taken to be an endorsement of the second by the first. The graph is then given as an input to an algorithm that computes a score for every file based on the total number of direct and indirect endorsement it receives from other documents. In the PageRank algorithm, used by Google, the scores are provided by a

¹We use “document” to refer to any kind of search results.

stationary distribution over a graph derived from the web, and can be computed using a sequence of matrix multiplications with certain convergence guarantees. Other search engines use similar algorithms [1] [4].

In addition to this, several other heuristics, such as anchor text analysis, page structure analysis [6] [7], and URL keyword matching, have been used to enhance the quality of the ranking.

While ranking algorithms that make use of these information sources have been very successful in giving users access to the most important content on the web, it is important to note that by taking the aforementioned approach, they compute a ranking that reflects the view of the authors of the web, as opposed to that of the users. Moreover, the rank they compute is global to the whole web, and is not suited to tailor search results to the specific needs of each user. Given the great potential in personalization, most search engines have tried to address this shortcoming by updating the score using information about the user’s location [8] and search, or click through, history [9] [10] [11]. While there is no doubt about the usefulness of this information, the ways to incorporate them usually involve coming up with an ad-hoc modification of an elegant ranking algorithm, or a more complicated model.

To provide personalized search result, a search engine designer is usually interested in capturing the preferences of the user in the ranking function, and while some of these preferences are very idiosyncratic, many can be captured by looking at the “context” of the user such as the way she is connected to other users and authors, her geographic location, and so on. Furthermore, since the investigation of these preferences is an open area of inquiry, it is important that the way they get combined during the personalization process is flexible to allow for future extensions.

This thesis documents a personalized search system that utilizes the the user’s social network, her content, and other content producers and recommenders. We argue that the incorporation of this social information is of great help in achieving a satisfactory search resolution on the local scale, as opposed to the global. Our design emphasizes the utility of user-specific information in ranking search results, and the need for modularity in combining rankings from different sources to determine the

importance of each search result.

This thesis introduces some contributions to the field of web search. Our application social network analysis techniques to the problem of web-search in a production environment is one of the few attempts in the field. Our use of rank aggregation methods from social choice theory, and statistics, simplifies the problem of search personalization. Finally, our design and approach leaves space for future work on learning, and other related issues.

1.1 Motivating Example

To appreciate the utility of the search personalization approach mentioned before, consider a query that contains the name “Michael Jordan”. For a member of the MIT Basketball team, or even a general member of the MIT community, the query might be an attempt to learn the whereabouts of a professional basketball player with that name. In contrast, a member of the Lab of Information and Decision Systems at MIT is probably interested in talks and publications by a leading researcher in Machine Learning and Artificial Intelligence.

Similarly, a student at the department of material science might use the keyword “annealing” to obtain information about the metallurgical process with that name, but a computer scientist will probably use the same keyword to search for information about the method of “simulated annealing”; an algorithm used in optimization problems.

In each of the two aforementioned cases, what the user “means” by the query depends on a set of properties pertaining to what they do, or what they are interested in. While these queries can be disambiguated in different ways [?], but we believe that the use of the user’s social context is a very natural approach to the problem, due to its similarity to the way people obtain information in real life, namely by asking colleagues, friends, and family.

For a more general case, consider the scenario where a user has to choose between a number of events to attend, where each event has a time, a topic, and a geographical

location on campus, and might be attended by the user's friends or acquaintances. While it might be easy for the user to indicate her preferences for each of these factors separately, it is usually hard for her to combine the different criteria, and the ability to rank the available events according to each factor, and combine them to get an aggregate rank is of great value.

1.2 System Components

The main components of our search engine are:

- A Front-end search interface, which receives user queries and informations and transforms them into a format that can be used by the other parts of the system.
- A Search Server which bridges the user interface with the components that handle the actual search procedure.
- A Crawler which crawls the web for content.
- An Basic Index which indexes the crawled corpus and provides efficient term-based access to the raw documents.
- A Parser which parses the crawled documents, recognizes special types of documents (e.g. calendar events), and represents these in an appropriate manner.
- A Tagger, which tags parsed documents with their corresponding social entity.
- An Enhanced Index, which contains the parsed and tags documents, and provides keyword based access to the parsed and tagged documents.
- A Scoring Box, which scores search results in various ways and combines the scores.
- A Social Graph, which stores the social network, and provides search and various distance routines on the graph.

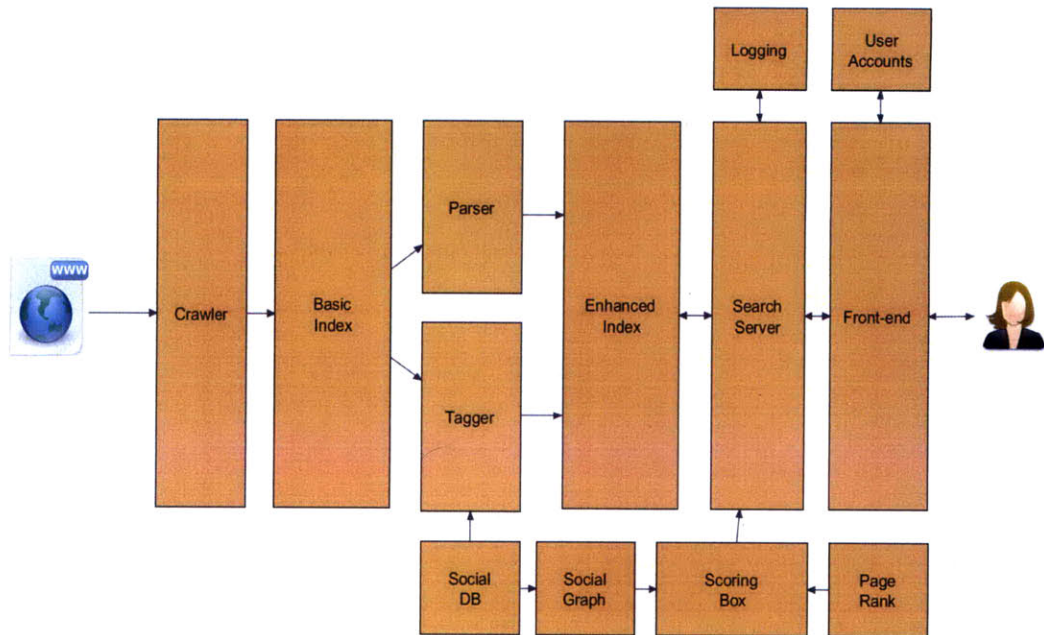


Figure 1-1: Architecture of our social network based search engine

- A PageRank scoring module, which provides an approximation of the PageRank score for the documents served by the system.

All the modules of the system are implemented using Java and Python.

1.3 Outline

The remainder of this thesis is structured as follows. In Chapter 2, we discuss the design and implementation of the system together with the algorithms implemented in each part. In Chapter 3, we provide an evaluation of the system, and some illustrative examples. In Chapter 4, we look at some of the previous work in the areas of web search, personalized web search, social search, and rank aggregation. In chapter 5, we consider some of the future directions for our system.

Chapter 2

System Details and Algorithms

In this chapter, we discuss the implementation details of the personalized social search system. Each section details a particular component of the system, discussing the design assumptions, choices, and trade-offs that were necessary.

This chapter, along with the code documentation, serves as a comprehensive documentation of this project for future users and developers of the system. While the exact implementation details are left out, a working knowledge of the technologies and programming languages used might help appreciate the different components of the design.

At the top level, our search engine is divided into two main components: the front end user interface, which handles user queries and settings, and a back end which handles user authentication, query processing, and the actual search functionality. The two components are designed and implemented separately, and communicate using a remote procedure call protocol. This separation, which is not necessary, is a common practice in the software industry, which simplifies the task of optimizing and scaling up the system.

2.1 Front-end Interface

The front-end of our system is mostly implemented using the Django framework; a Python web application framework that follows the Model-View-Controller design

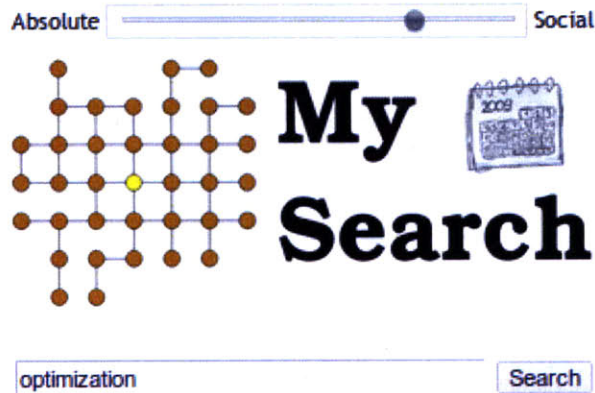


Figure 2-1: Search Box and Adjustment Slider

pattern. This part consists of a user interface for search, an automatic data collection module, and a user interface to allow users to modify their profile, and annotate data sets.

The web interface for our search engine receives the search query from the searcher, passes it to the Search Server, and displays the returned search results. In addition to the search box, the web interface includes a slider, which enables the searcher to adjust the contribution of their social network to the search score. The results are displayed as a list of items, where each item consists of a title, a link, a descriptive text blurb, the social entities associated with the result, and any available optional fields. For special types of search results, such as seminars, these optional fields include the time and location of the seminar, and the searcher is given the option to add the seminar to her calendar, or share it with friends.

The annotation module allows users to display the social networks they participate in, and modify these social networks by adding or removing social entities or links between them.

The automatic data collection module monitors the user behavior in response to

optimization

Search

Pricing Randomized Allocations

Location: **E51-376**
April 29th
4:15pm -



optimal pricing for single-parameter consumers does not require selling lotteries, but this ceases to be the case when one considers multi-parameter problems. To what extent can a seller improve revenue by pricing lotteries rather than items, and does this modification of the problem affect its computational tractability? We investigate these questions in the context of an archetypical profit maximization problem: selling heterogeneous items in unlimited supply to unit-demand bidders. The answers hinge on whether consumers can purchase only one lottery (the buy-one model) or purchase any set of lotteries and receive an independent sample from

[orc] [1.27910789673]

A Dynamic Near-Optimal Algorithm for Online Linear Programming

Location: **E51-376**
May 13th
4:15pm -



optimization model that formulates many online resource allocation and revenue management problems is the online linear program (LP) where the constraint matrix is revealed column by column along with the objective function. We provide a near-**optimal** algorithm for this surprisingly general class of online problems under the assumption of random order of arrival and some mild conditions on the size of the LP right-hand-side input. Our learning-based algorithm works by dynamically updating a threshold price vector at geometric time intervals, where the dual prices learned from revealed columns in the previous period are used to determine

[orc] [1.27910789673]

Figure 2-2: Search Results

queries, and logs this behavior for future learning ¹. The logged data includes the user identification number, the query, and the selected result(s); the selection can take the form of clicking on the event, adding it to the users calendar, or sharing it with friends. Additionally, the front-end provides enables users to register in the system, create, and modify their profiles.

The different modules of the front end interact with the back end using a Remote Procedure Call (RPC) protocol implemented using Googles Protocol Buffers technology. The passed information includes authentication information for login, search queries and results, and user clicks.

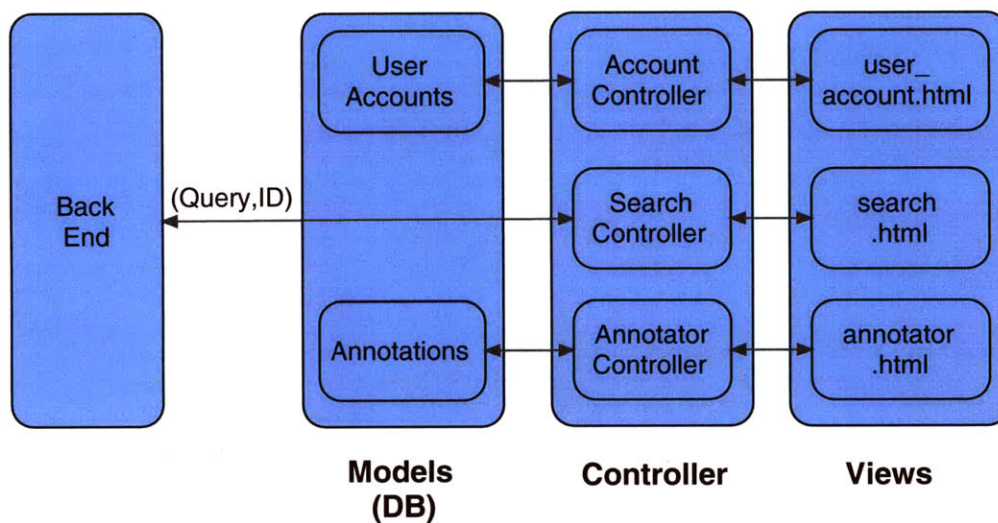


Figure 2-3: Front-end Components

¹Using the search history to learn the preferences of the user and enrich their social network

2.2 Remote Procedure Calls

In order for the different parts of the system to interact efficiently while maintaining a reasonable level of modularity, the different parts of the system are equipped with Remote Procedure Call interfaces. Each interface is run as a web service that takes arguments and return results in a fashion similar to that of local software routine. These web services can be optimized, tested, and run separately, and only interact with each other during run time. The search interface, for example, can pass the user query to the Search Server, to get a list of results, that could contain a few hundred or thousand results at a time, that is then presented for the user.

A common solution for this problem, involves representing the service arguments (e.g. a search query) and the returned result (e.g. search results) as XML documents, which are passed back and forth over some web protocol (e.g. HTTP). In this process, data structures that are used by the different modules are encoded into XML by the sender, and decoded by the receiver to recover the original data structures.

While the use of XML has the advantage of being an industry standard, and a popular choice among developers, we chose to avoid it in favor of Protocol Buffers, a serialization language implemented and widely used by Google [12]. With Protocol Buffers, data structures are serialized into a binary format, as opposed to verbose XML. In addition to their simplicity and flexibility for future extension, protocol buffers are smaller in size, provide a 20 to 100 increase in performance [12]. More information about protocol buffers can be found in the Google documentation, and the code documentation for our search engine.

2.3 Back-end Server

The back-end of our search engine is broken down into a number of modules, most of which are implemented using Java. These modules are responsible for crawling, indexing, searching, and logging, and can be seen in the diagram in figure ??.

2.3.1 Crawling and Basic Indexing

The main Crawler for our search engine is essentially a collection of specialized crawlers that deal with handle different information sources, with the option to add new crawlers as needed. For the time being, our crawls are limited to ordinary web pages, PDF documents, and links shared on Facebook. In the future, we intend to cover other social networking sources, and the collections of the MIT Libraries.

At the basic level, our system uses the Nutch crawler, implemented as a part of the Apache project, to fetch and organize the web pages and PDF documents. For the purpose of social search, we limit our web crawl to a list of MIT websites that are identified as a part of our social network. As a part of the Nutch architecture, the crawled pages are stored in a database, and an index is built; In this index, each keyword that appears in the corpus, is linked to all the documents that contain it as a part of their main, or anchor, text. The Facebook content is crawled using a custom built crawler for Facebook. This crawler, which is granted access permission by the user, crawls the users content, and adds it to the same index as the web pages. All the documents from the Facebook crawls are tagged with the Facebook identifier for that user, and the relationship of the user to that document (e.g. created, liked, posted). Facebook identifiers can be either integers (e.g. 709292) or strings (ammar.ammar), and are unique for each user or group on Facebook.

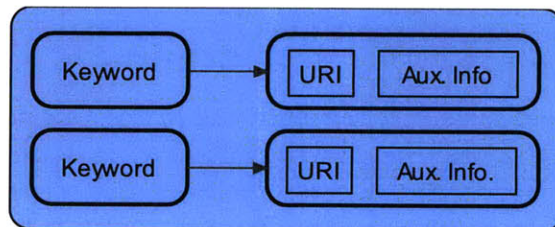


Figure 2-4: The Basic Index

2.3.2 Social Tagging

The functionality of our personalize social search relies on our ability to associate documents with social entities, before computing the affinity between these entities

and the searcher. For that purpose, all the documents previously crawled by the crawler need to be associated with a host from the social network. The tagging module is the part that achieves that goal.

The tagging module has access to a database with all social entities, and their unique identifiers. Each identifier is an unsigned 32-bit integer, with the unique identifier 0 preserved for the unrecognized social entity. Furthermore, each social entity is associated with auxiliary information to help the tagger identify objects belonging to this entity. In the case Facebook users, this information contains the Facebook user identifier, which can be a number or a string. In the case of social entities which only exist as websites, this information includes all top level domains and IP addresses associated with that entity.

The taggers is implemented using Java, and comes with an API that provides a number of functions that take a document URI or Facebook ID, and return the unique identifier for its host. It also provides a string-based look up function to find the unique identifier given the name or acronym of the entity (e.g. MIT LIDS). The tagger is also packaged with a web service that allows other parts of the system, to access these identifier as needed.

The tagging, or look up, algorithm differs depending on the type of the document in question. For example, if we want to tag a document retrieved from a URI like:

```
http://lids.mit.edu/news-and-events/seminars.html
```

or,

```
http://128.31.4.71/news-and-events/seminars.html
```

The URI is passed to the tagger together with the type of the document, which is WEB in this case. The tagger then tries to identify the host by looking up the top level domain (i.e. lids.mit.edu) in its look-up table. If no entity is found, it tries again using the next level of the URI (i.e. lids.mit.edu/news-and-events/), and repeats this until all levels are tried. If this fails, the base domain (i.e. mit.edu) is used, and if that fails, the identifier 0 is returned. In the case of Facebook, the documents

are assigned the type “FACEBOOK”, and the tagger simply tries the integer and/or string Facebook identifier associated with the document.

For performance purposes, a comprehensive tagging of all the crawled document is done off-line to improve the runtime performance of later computations. At this off-line stage, the Tagger goes over all documents, tags them with their unique social identifier, and stores them in the Enhanced Index.

2.3.3 The Enhanced Index

As we mentioned in the crawler section, the crawling step is concluded by building an index that links keywords from the main, or anchor, text pointing to URIs of the documents containing, or linked to, by these keywords. In the tagging step, this index is augmented with the unique social identifiers of these documents to prepare them for real time search.

To achieve the maximum utility of these crawled documents, and to enable future personalization, these documents undergo various types of parsing. The parsing is done based on the document type. The current version of our parsing system is limited to parsing events (e.g. seminars) that take place on the MIT campus, but the design make future extensions easy, and intuitive.

At the top level, all outputs of the parsing system are stored in a number of database tables linked to by the original index keywords, which constitute the Enhanced Index. In this index, each parsed and tagged document corresponds to a SearchResult data structure, which has title, description, and URL attributes. These SearchResults can be as simple as a web page containing HTML text, or more complicated depending on the type of the parser. MIT campus events for instance, are parsed using a special parser which extracts most of the details of the event such as the time, the location, the sponsor, and so on. These details, together with the basic information, correspond to CalendarEvent data structure, which is a subclass of SearchResult.

To reduce the complexity of this kind of fine grained parsing, and speed up the parsing procedure, the parser makes use of a collection of SVM classifiers that try

to classify documents based on the type of information they contain. The classifiers are trained using manually or semi-automatically constructed datasets containing documents of this type. Furthermore, classifiers and parsers can be added to the system easily to provide support for new types of information.

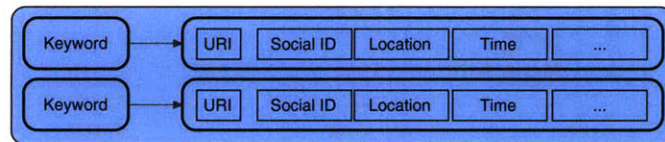


Figure 2-5: The Enhanced Index

2.3.4 The Search Server

The Search Server is the main interaction point between the front-end components, and other components of the system. The server is implemented as a web service using Protocol Buffers. The server API provides clients with the ability to perform social search queries using a keyword, a searcher ID, and a weight parameter that is later used by the Aggregator to determine the contribution of the social network to the total ranking. Furthermore, the API provides functions to limit the search to certain types of results (e.g. seminars).

In order to provide the aforementioned functionality, the server interacts with the Search Index, and the Ranking Box, to fetch and rank the results, respectively. The server then returns a list of SearchResult objects to the client for further processing and display.

2.3.5 The Query Parser

When a user submits a search query in a string format, this query is translated into a format that can be sent to the index. The supported query syntax can be found in table 2.1.

Table 2.1: Query Syntax

Syntax	Find Pages Containing
vacation Hawaii	the words vacation and Hawaii
Maui OR Hawaii	either the word Maui or the word Hawaii
"To each his own"	the exact phrase to each his own
virus compute	the word virus but NOT the word computer

2.3.6 The Ranking Box

After fetching the search results from the index, the sever then passes these results, along with the user information, to the Ranking Box to score them. This is achieved by sending the results to a number of scoring sub-modules, each specializing in a certain kind of ranking. In our system, we consider two types of scores: a social network based score, and a popularity score. The social network score, which is discussed in the Social Graph section, scores documents based on the distance of their host(s) from the searcher, and reflects the relevance of the document to the searcher. In contrast, the absolute score is an approximation of the PageRank score, and is intended to reflect the popularity of these documents on the whole web. The Ranking Box can be extended to support new types of scores such as the spatial distance of a certain event from the searchers location. When all the documents are scored, the Ranking Box sorts each scored list and passes all the lists to the Aggregator, which combines them into one list sorted list.

Absolute Scoring

absolute scores aim to capture the

The Absolute Scoring module provides an absolute, social network independent score that reflects the importance of documents using their position in the overall structure of the web. In the case of web pages, a popular choice for this score is the PageRank score, developed and used by Google [4]. Given the limited computational resources available for our prototype, the Absolute Score module only provides an approximation of the actual PageRank score. This approximation is computed beforehand using a randomized algorithm, and stored in a database.

The Absolute Score module consists of two parts: an off-line PageRank estimator, which takes a list of document URIs and saves estimates of their PageRank to a database, and a search time look-up web service to get the score for a given document.

While the value of PageRank used by Google is calculated with high precision, the publicly available estimates are limited to integer values between 0 and 10. For our application, this value is of little use, since many of the pages we are interested in share the same integer value. Since we get information about the PageRank, or absolute, importance of two documents by comparing their order in search results returned by Google, we can simply try to get as many of these pages returned by Google, and record their order in this list. Once weve gotten enough of these ordered lists, we can then combine them to come up with an ordered list of all the documents. To obtains these partial lists, we randomly pick special keywords from these documents and construct queries that are then passed to Google to get the order of the documents in question. These special keywords can be keywords in the header of the page, or capitalized keywords from the body of the document, among either classes. Once the full ordered list is constructed, we give each item in the list a numerical value that reflects its position in the order.

That said, it is important to note that this algorithm is a very rough estimate of the PageRank value, and is only intended for experimental purposes only. We plan to replace these estimates with better scores of the absolute importance of documents. Moreover, these scores are usually only available for documents on the web, and an alternative score needs to be found for content hosted on Facebook, and therefore not a part of the web crawled by Google.

Social Scoring

So far, most of the system parts discussed are typical for the functionality of any search engine. The social scoring module is what distinguishes our system from a typical search engine. The social score aims to capture the importance of the document given the relationship between its host, and the searcher. This intuitive way to score documents is inspired by the procedure users follow to obtain information in real life: they simply ask their friends and acquaintances, and if that fails, they ask these friends to relay the question to their friends, and so on. In this information retrieval paradigm, the structure of the social network which has these users as its nodes, and the presence of relationships between them as edges, is assumed to provide an encoding of these users preferences. That is to say, if user A likes a document, and if A is a friend of B, then B will probably like this document. The same kind of reasoning applies to the case of dislike, authorship, and explicit recommendation. For now, we limit our attention to positive user-document relations.

Before we attempt to translate the aforementioned intuition into a program, it is important to keep in mind that there are many functions, or metrics, on the structure of the social network, that can, to some extent, capture this intuition. For a trivial example, we can simply take all the documents preferred by the searchers friends, and return them as our results. There are a few issues with this simple approach. For one, it misses on a large amount of information available in the form of social relationships, and documents hosted by people who dont happen to be friends of the user. To address this issue, we need to go beyond the first level of friends, to their friends, and their friends friends, and so on. When we do this, however, we have to keep in mind that the social value of information decays as we go further in the social network; while I might be willing to attend a lecture recommended by a friend, I might not be as enthusiastic about a party recommended by a friend of her grandfathers. Its also important to note that this rate of decay in importance is likely to be user specific, and should be learned from the user.

Another constraint on the social score presented by the relationship between this

score and other types of scores, such as the absolute score. The preferences inferred from the social network should not dominate, or block, preferences of a global nature. For example, a lecture on alternative energy delivered by the secretary of energy should still get a high score compared to a lecture on Bethe free energy, Kichuchi approximations, and belief propagation.

In our design, we take an experimental approach to address these problems. First we provide a number of different graph metrics that reasonably capture our intuition. We also provide weighting parameters to control the scoring procedure whenever possible, with the hope that at this stage, users will be able to tweak the system to achieve maximum utility, while enabling us to collect data for future improvement.

At the implementation level, the social scoring part of the system consists of a Social Graph; an undirected graph consisting of the available social entities and their relationships. Each entity is referred to by the same unique social identifier mentioned in the section about tagging. So far, these social entities include research groups and labs at MIT, and users who opted to join and provide extra information from their Facebook accounts. In the case of research groups, a graph edge is interpreted as a research collaboration or an overlap in research topics between the groups. For scoring purposes, the Social Graph is accompanied by a number of Metric classes. Each metric class is basically an algorithm that computes a function on the graph.

The Social Graph

The Social Graph is an interface that provides the functionality expected of both directed and undirected social networks, and can be implemented in different ways, while maintaining compatibility with other parts of the system. Generally speaking, the Social Graph supports all operations needed, to create, update, and get information about the underlying social network. These operations include:

- Loading a social network from a file.
- Adding, and deleting nodes and edges.
- Checking for the existence of a node or an edge.

- Fetching the neighbors of a given node.
- Computing various graph distances.

Based on the assumption that influence in social networks is bidirectional, and varies in degree from one social relation to another, the current version of our system uses a weighted undirected graph implementation of the social network. Nonetheless, a weighted directed social network implementation is available and can be easily deployed for future experimentation. Both types of networks are implemented using an efficient adjacency matrix representation.

Finally, the Social Graph module can be instantiated and used by any module in the system, and is currently used within the Ranking Box to score search results. It can also be run as a stand-alone module, and provides a web service for interactions over the web.

Distance Metrics and Algorithms

The Distance Metric is an interface that provides the ability to compute a score relating any two nodes in the graph. This can be any score that is efficiently computed from the adjacency matrix, such as the shortest path, or the number of common neighbors, and does not have to be a metric, or a distance! Implementations of the Distance Metric class provide a function that takes a Social Graph, and two node identifiers and returns a numerical value . While this score can mean anything, we adopt the convention that this value should be proportional to the affinity, or influence, between the two social entities in question, and therefore the importance of the documents hosted by one entity to the other.

Our system implements two graph algorithms that have been used and discussed in various areas of social network analysis [13] [14], but not necessarily in the context of search. At the moment of writing, our system supports the following algorithms:

- Shortest Path: for this metric, we simply find the shortest path between two nodes, and return the reciprocal of the path weight as our score. This way, we

ensure that the further are the two nodes in the graph, the less likely that they will influence each other. We also make sure that direct friends are scored higher than distant friends, and so on.

- **Number of Common Neighbors:** as the name implies, the number of common neighbors is simply the number of neighbors that the two nodes share. While this quantity is non-zero only for nodes that are close enough to share neighbors, previous research in collaboration networks [14] has shown that the number of common neighbors is correlated with the probability that the nodes will collaborate in the future.

2.3.7 The Rank Aggregator

The Rank Aggregator is the part of the system responsible for combining the different ranks returned by the ranking modules, in our case the social and absolute rankings. At the moment of writing, our system supports two modes of aggregation, one based on Borda's method [15], the other is motivated by Michael Intriligator probabilistic model of Social Choice [16]. In both methods, the aggregator receives an ordered partial list of the search result, and return one ordered list that is then forwarded to the searcher. The aggregator can also be set to skip both of these methods and operate in a "passive mode".

Passive Aggregation

In this method, the aggregator simply uses the scores coming from the different ranking boxes. It simply adds the two scores, and reranks the lists based on the resulting total scores. This mode is intended for experimentation with the different scoring metrics used by the scoring box, to explore their usefulness. The aggregate score for each results in this mode is equal to:

$$\alpha \cdot \textit{SocialScore} + (1 - \alpha) \cdot \textit{PageRank}$$

Where α is an aggregation parameter that takes values between 0 and 1, which is provided by the user.

Borda's Method

In this method, each ordered list of results is treated as a linear order, where each search result is assigned a score corresponding to its position in the ranked list of results. The scores for each result are then combined using the aggregation parameter (α), in a fashion similar to that in the previous section, and the results are re-sorted using their total scores.

Apart from its obvious simplicity, a primary advantage of this approach is its computational efficiency, as it can be implemented in linear time. Furthermore, this method enjoys the properties called consistency, anonymity, and neutrality in Social Choice theory [17]; the latter two of which are of no apparent relevance to our problem.

Probabilistic Aggregation

In this approach, we assume that every ranking source has a limited number of units that it can distribute to the search results according to its preference. In other words, each source defines a probability distribution over the results in question, where this distribution implies a ranking of the results. The probabilities obtained from the different sources for each item are then combined using the aggregation parameter (α) to obtain the total score of the item, and the results are then ranked based on this score. The total score for item i is given by:

$$p_i = \alpha \cdot q_{i,Social} + (1 - \alpha) \cdot q_{i,PageRank}$$

In order to perform the aforementioned computation, the Aggregator is given the probability distribution to use with each source of ranking. These distributions can either be the same, or different. Furthermore, they can be chosen to capture certain patterns in the user's response to search results.

2.3.8 Caching

In order to eliminate redundant computations, and reduce the query time, we use a cache implemented using the Java Caching System. The cache stores every query and the corresponding result list, and is refreshed after each new crawl to make sure that all the results are up to date. The cache also stores graph computations, which are refreshed every time the graph is modified.

2.3.9 Logging

For future improvements and learning, the system logs all user queries, the list of results that got displayed, and the results they ended up clicking on, adding to their calendar, or sharing with friends. The logging system is implemented using Log4J; a Java library for logging.

2.4 Scalability

So far, we have discussed the design and implementation of a system that runs on one server, but that need not be the case. The modularity of our system, and the availability of web service APIs between the modules enables us to run the system on a number of different machines. Furthermore, redundancy can be introduced into the system by replicating any of the nodes as needed, and introducing the appropriate load balancing mechanisms.

To see how this can be done, lets assume that our current server can handle 10,000 users, and that the workload is equal for the different parts of the system. Now supposed we want to scale our system to serve a 100,000 users. To do this, we can run all our modules as web services, using Apache, and then configure the Apache Load Balancer to forward traffic to the appropriate nodes.

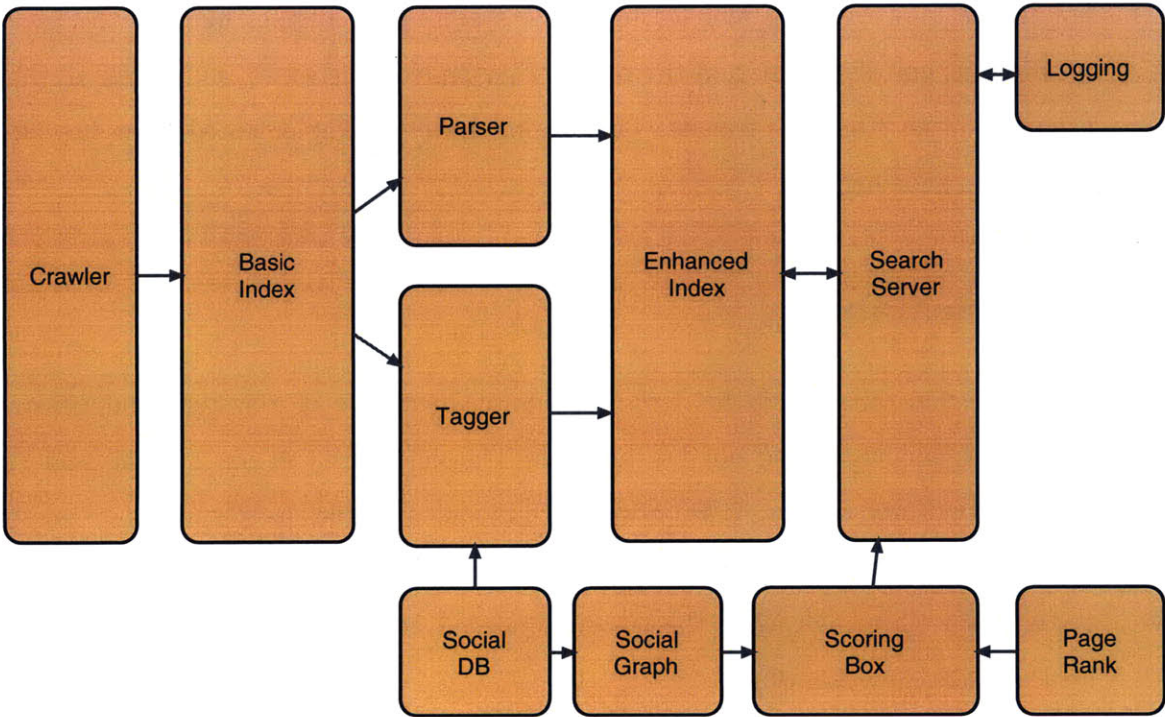


Figure 2-6: Back-end Components

Chapter 3

Performance and Evaluation

3.1 System Performance

In any practical system, performance is a concern. In this project, we tried to implement a search engine that uses a social network, and we believe that we have achieved that goal. With regards to the performance, we had to make sure that search results are returned to the user in a reasonable time compared to other search engines that the user is used to. In particular, we had to make sure that the query time grows slowly as the size of the index and the number of matching results grow. Since each of the matching results is assigned a score based on a social network “metric”, the performance in that part is also determined by the efficiency by which we compute these “metrics”. For the “metrics” we use so far, the shortest path and the number of common neighbors, we used Dijkstra’s Algorithm, and a simple set intersection procedure respectively. We believe both algorithms to be a good choice, but the only way to verify that is to run experiments on the whose system.

In this section, we report the results of some of these experiments performed using various test indices together with our MIT social network, which contains 400 labs and research groups. Our tests cover the growth in the size of the index, and the number of matching results, but not the growth in the size of the social network. We believe the latter part to be very important, and we intend to design an experiment to evaluate its effect in the near future. We also ignore the issue of computing the

PageRank score, since it's computed offline, or can be obtained from other sources, without affecting the user experience.

In each experiment, we construct an index of fictitious events that are taking place at MIT, and run a set of queries on that index. The queries, which are used in all tests, are designed to estimate the time from the submission of the query, to the display of the search results to the user.

Of all the events used in each experiment, only 20000 are targeted by the 20 queries in our test queries. The events matching each of these 20 queries can be viewed as 20 nested sets of keywords, with the first query matching only a 1000 events, the last matching 20000, and the rest matching the increments in between. While 20000 is small compared to the number of queries usually reported by search engine like Google, we believe it to be appropriate for our setting.

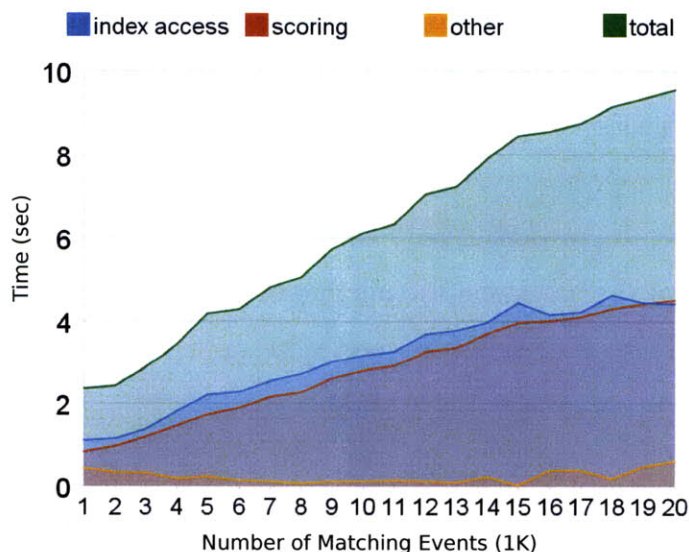


Figure 3-1: Query, index access, and scoring times, in seconds.

In our first experiment, we use an index with 600,000 events, including the aforementioned subset of 20000 events. We then query the system with the 20 queries mentioned above, and report the average time taken. Furthermore, we report the time taken to access the index, score the results, and the discrepancy between these times and the total time, usually spent in other parts of the system. We ran this

experiment on our first version of the system, without caching. The results can be seen in figure 3-1.

As expected, the uncached version of the system is relatively slow, with around 10 seconds to perform the largest query. This time can be reduced to almost one third by implementing a caching layer on top of all the performed computations. , as our second experiment indicates. The results are shown in figure 3-2.

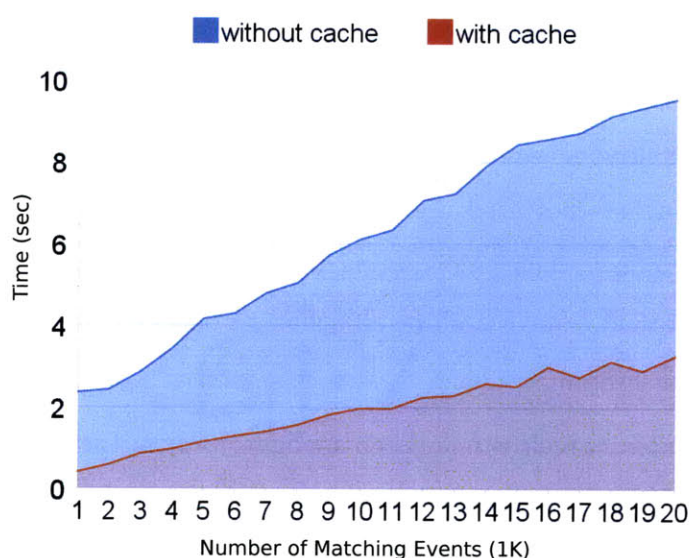


Figure 3-2: Query Time, in seconds.

Finally, we perform an experiment to test the effect of index size on the query time. We use three indices with 600,000, 1 million, and 1.5 million events. The results can be see in figure 3-3.

These experiments were performed on a workstation with two 2.1 GHz processors, and 2GB of ram. As we can see, for a fixed index size, the growth in query time is almost linear. We can also see that the growth in query time is slightly affected by the size of the index, and will likely be limited to a few seconds for the purpose of searching documents available on the MIT campus. Should the need for larger indices arise, the system will need to be scaled in different ways.

3.2 Search Quality

The evaluation of social network based search, and personalized search in general, is a challenging task. This stems from the fact that we are trying to capture the importance from each searcher's perspective. One way to carry out such evaluation is to conduct user studies where users are asked to search and indicated their satisfaction with the results. Another, more passive, approach consists of monitoring the reaction of users to the search results, and trying to extract information about the user's satisfaction. We can, for example, count the distance of the results clicked on by the user from the top of the result list, and use this distance to come up with a score. The problem with these approaches is that they take a considerable amount of time, and require a group of dedicated users. For the time being, we chose to avoid this tedious task, and follow a heuristic approach in judging the quality of our search. In this section, we present some search results that we believe to provide an illustration of the utility of our system.

The first example, which can be seen in figure 3-4, is the response to a search query that uses the word "annealing" from the point of view of a searcher at the Laboratory of Information and Decision Systems (LIDS). Given the nature of the work done at LIDS, and its location in the social network, the keyword "annealing" should preferably refer to the "simulated annealing"; an algorithm used in optimization. As we see in the results, which have been abbreviated for ease of viewing, the first two results come from LIDS, and the Operations Research Center (ORC), and do indeed refer to that algorithm.

When we repeat the same query for "annealing" from the point of view of a searcher at the Microsystems Technology Laboratory (MTL), where a lot of the semiconductor research at MIT is done, the results are reordered to capture the effect of the new context. As we can see in figure 3-6, the first two results now come from MTL itself, and Material Science Department, and refer to the metallurgical process with the same name. Note that the results which topped the query from LIDS are still present in later positions in the list.

Our last example illustrates the ability of our system to preserve the score of popular results while presenting socially relevant results. In this example, which can be seen in figure ??, we perform a query that uses the keyword “cooperation”, from the perspective of a LIDS searcher. On the social side, we are still able to capture the importance of seminars that involve the idea of cooperation from a game theoretic point of view, which are hosted by the Mathematics and Physics Departments. But we can also capture important events happening at the MIT campus, such as a lecture by Noam Chomsky, hosted by the Center for International Studies (CIS), and discussing the recent unfortunate developments in Haiti.

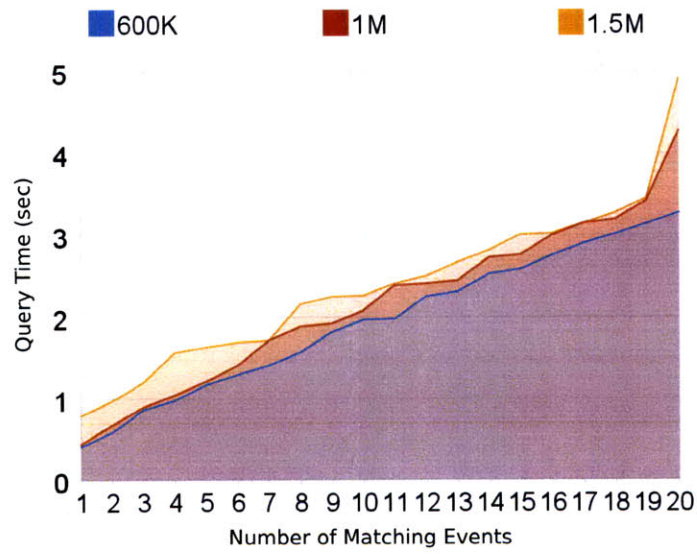



Figure 3-3: Query Time for various indices, in seconds.

Absolute

 Social



My Search



annealing Search

1

[Deterministic **Annealing** for Multiple-Instance Learning](#)



Location: 32-155
 Wednesday, September 15, 2010
 4:00P -

deterministic annealing can be applied to different SVM formulations of the multiple-instance learning (MIL) problem. Our results show that we find better local minima compared to the [LIDS]

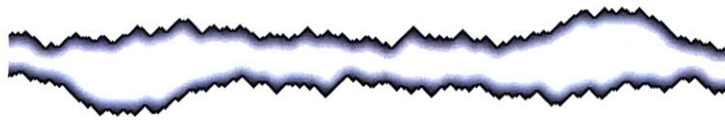
2

[Simulated **Annealing** in Convex Bodies](#)



Location: 32-155
 Thursday, August 26, 2010
 4:00P -

We apply the method known as simulated **annealing** to the following problem in convex optimization: minimize a linear function over an arbitrary convex set, where the convex set is [ORC]



6

[Scanned pulsed laser **annealing** of Cu thin films](#)



Location: 34-101
 Tuesday, October 21, 2010
 1:00P -

been much research into microstructure control in Cu thin films, primarily because grain sizes affect resistivity. Also with Cu-based interconnects, interfacial electromigration is the dominant [Material Science]

7

[3D Silicon Transformation using Hydrogen **Annealing**](#)

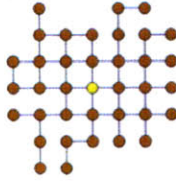


Location: 39-324
 Thursday, September 19, 2010
 2:00P -


Fabricating smooth, round structures in silicon is challenging by conventional processing methods. In this paper, we report on a novel 3D shaping technique for producing [MTL]

Figure 3-4: Search Results for “annealing” from LIDS

Absolute Social



My Search



annealing Search

1

[3D Silicon Transformation using Hydrogen Annealing](#)



Location: 39-324
Thursday, September 19, 2010
2:00P -

Fabricating smooth, round structures in silicon is challenging by conventional processing methods. In this paper, we report on a novel 3D shaping technique for producing [MTL]

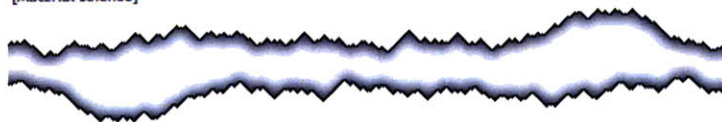
2

[Scanned pulsed laser annealing of Cu thin films](#)



Location: 34-101
Tuesday, October 21, 2010
1:00P -

been much research into microstructure control in Cu thin films, primarily because grain sizes affect resistivity. Also with Cu-based interconnects, interfacial electromigration is the dominant [Material Science]



5

[Deterministic Annealing for Multiple-Instance Learning](#)



Location: 32-155
Wednesday, September 15, 2010
4:00P -

deterministic annealing can be applied to different SVM formulations of the multiple-instance learning (MIL) problem. Our results show that we find better local minima compared to the [LIDS]

6

[Simulated Annealing in Convex Bodies](#)



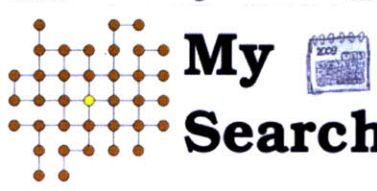
Location: 32-155
Thursday, August 26, 2010
4:00P -

We apply the method known as simulated annealing to the following problem in convex optimization: minimize a linear function over an arbitrary convex set, where the convex set is [ORC]


Figure 3-5: Search Results for “annealing” from MTL


Absolute

 Social



My Search


[Evolutionary Dynamics in Structured Populations: The Evolution of Cooperation](#) 


Location: 4-231 

Thursday, February 26, 2009

5:00P -

attempts to study the effect of population structure on evolutionary and ecological dynamics. These approaches include spatial models in ecology, viscous populations, spatial games and [Mathematics, Department of]


[Physics IAP Lecture Series - "Games microbes play: The game theory behind cooperative sucrose metabolism in yeast"](#) 


Location: 6-120 

Wednesday, January 14, 2009

1:30p - 2:30p

Understanding the conditions required for the evolution of **cooperative** behaviors is an enduring challenge in evolutionary biology. How can **cooperators** ensure that they are not [Physics Department]

[Noam Chomsky: Sustainable Cooperation with Haiti](#) 

Location: E15-070 

Friday, January 22, 2010

5:00P -

This series will commence tomorrow (Friday, January 22) at 5pm in the Bartos theatre, E15-070, with an open discussion to be led by Noam **Chomsky**. Professor **Chomsky** [Center for International Studies]




Figure 3-6: Search Results for “cooperation” from LIDS

Chapter 4

Related and Prior Work

This chapter presents a sample of the previous work on the problems of search, search personalization, social search, and rank aggregation.

4.1 Absolute Search

In their seminal paper [4], Page and Brin proposed a model of a web surfer as a random walk on a graph consisting of all the pages on the web, connected by directed links. In this model, the random surfer jumps from one page to one of its successors uniformly at random, occasionally jumping to a completely random page, to avoid getting stuck in loops. The ranking function over all pages, known as the PageRank, is then equivalent to the standing probability distribution of the random walk.

More precisely, if we let N_u be the number of outgoing links of page u , and consider the matrix A corresponding to the web graph, where each entry A_{uv} is equal to $1/N_u$ if u links to v , and 0 otherwise, then the rank vector R , which represents the standing distribution over all nodes, is the principle eigen vector of the matrix A :

$$R = AR$$

This vector can be computed by repeatedly multiplying A by a non-degenerate initialization of R . To model the random jumps, we can use a vector E whose entries

correspond to the probability of each page being the rank source. In that case, the model becomes:

$$R^i = A(R^i + E)$$

Page and Brin proposed an algorithm that computes this rank vector, and converges in a number of iterations that is roughly linear in $\log n$, where n is the number of nodes. More generally, this quick convergence is guaranteed for any expander-like graph ¹.

Kleinberg et. al. [1] [2] developed a model of the web in which important pages can be classified as hubs or authorities. In this model, a hub is a page that points to a number of important pages or authorities. To compute the hub and authority score for a page, Kleinberg et. al. proposed the HITS algorithm, which operates on a subgraph of the web obtained at query time. The subgraph is constructed by finding all the pages that match the query from an index-based search engine, adding all their parents and children, and removing all the links that are internal to a specific website. Once this subgraph is constructed, the hub and authority scores represented by the two vectors x and y , can be iteratively computed from the adjacency matrix A using the following iteration:

$$x_t = (A^T A)x_{t+1}$$

$$y_t = (A A^T)y_{t+1}$$

Upon convergence, these vectors are equal to the principle eigen vectors of $A^T A$ and $A A^T$ respectively.

¹An expander graph is a graph where any (not too large) subset of nodes is linked to a larger neighboring subset

4.2 Personalized Search

Kautz et. al. [18] designed and built a system called ReferralWeb to extract social relationships on the web and utilize them in search. To construct the social network for this purpose, they crawl the web for individual names, and take each co-occurrence of individual names on the same web page or the author list of a publication to signify a relationship between these individuals. They also augment this network using information from organizational charts and exchanges between individuals on public forums. In response to a users query, the system retrieves all the documents containing the query terms, match a topic specified by the searcher, and fall within a limited social radius from the searcher. No explicit scoring algorithm is given.

Bao et. al. [19] conduct an experiment on the use of social annotation on websites like Del.icio.us to rank search results. The aim for their experiment was to come up with a model that captures the importance of websites from the point of view of annotators; people who visit and annotate websites. In their model, they try to capture the importance of a certain annotation, and the importance of a certain page given the total number of annotations it receives. They propose two algorithms that converge and calculate a score that represents the two scores mentioned above. The scores produced by the two algorithms are then added to produce the final score of a certain document. Once thats done, these scores are then used in combination with query based search over the corpus. In their mode, the social aspect of the search is captured by annotations provided by users, and does not rely on some explicit social network from which the content is produced.

4.3 Social Search

Mislove et. al. [20] conducted an experiment on the utility of using social network in search. In their experiment, they asked ten graduate students to install a program which keeps track of the pages they use, indexes these pages, and allows other users to access these indices at query time. The score of the retrieved results is then computed

by multiplying the Google PageRank score by the sum of the index matching scores from the users who already accessed the result. By doing this, the authors found that in addition to boosting the score of certain documents, the use of the social network gave users access to a significant number of documents that are not covered by Google's index; an important result given the relatively small size of the index covered by the social network. The social network also helped disambiguate some of the search queries.

Bender et. al. [21] proposed a framework for searching content generated on social networks. In their framework, they treat users, documents, and tags as nodes in a weighted directed graph. In their graph, weights are assigned to edges to reflect different affinities between users, documents, and tags. For instance, if a user uses some tag frequently, the arc between the user and the tag receives a high weight, and so on. The directions of the arcs can be used to express relationships such as friendship, authorship, similarity (goes both ways), and rating or endorsement.

The entities in the network are scored in different ways. Users in the social network are ranked using a UserRank, which is PageRank computed on the user graph. Similarly documents are ranked using basic PageRank on the document graph. Tags, which are given a lot of significance in this model, are scored in their relationship to the user and a document using a user-specific BM25 score. When documents are retrieved at search time, the score of a certain document is computed as the sum of the scores combin

To verify the utility of their framework, Bender et. al. conducted an experiment on crawls from Flickr and del.icio.us containing around 16,000 users, 6 million items (images and documents), and 230,000 friendship relationships. Using a user study, they conclude the improvement in search quality can be perceived by the participating users.

Carmel et. al. [22] take on the problem of social search defined as the process of searching through social data collected through web 2.0 applications. Their approach is inspired by the reasonable assumption that the preferences of other people who are socially related to the user contain information that should be useful in revealing

and predicting the preferences of the user. Each user in their study is defined by a profile containing the users relationships to other users, and a list of terms that define the users topic.

They consider two kinds of social relationships: an explicit one based on familiarity, and extracted from a friendship graph, and an implicit one based on similarity, which is inferred from using the topics associated with the users through their bookmarks.

The score of each search result is then calculated as a (configurable) linear combination of a non-personalized (or absolute) score of the result, the product of the weight of the relation between the searcher and every neighbor who is related to the document and the strength of this relationship, and finally the match between the users topics and the document topics.

They evaluate their model using data collected through an in-house social software application called Lotus Connections, which allows users to create and tag content, join groups, and connect to other users. They conclude that personalization does indeed contribute to the quality of the search.

Horowitz and Kamvar [23], co-founders of Vark, a social search engine recently acquired by Google, propose and design a search engine which connects the searcher to people who might be able to answer the query.

The search engine access the users email and social networking site accounts, to construct a social graph of the users friends. It also creates a list of topics that the user might be interested in by processing their existing content on these websites. Each user in the network is then given a score that reflects the probability that the user will be able to answer a given query. This score is calculated as the product of the users affinity to the topic of the query and the social proximity of the user to the searcher. This proximity is computed as the cosine similarity of a set of features that include common friends and affiliations, demographic similarity among others, but a precise formula of this calculation is not given.

4.4 Rank Aggregation

The problem of Rank Aggregation originates in the Social Choice Theory literature as the problem of aggregating the votes of society members to determine the preference of the whole society over a number of candidates. Each member of the society casts her vote in the form of a ranked list over the available candidates, and the problem is usually to find a function that maps these votes, or individual rankings, into a ranking that has certain desirable properties.

One paradox that arises in this context was discovered by French mathematician, economist, and social scientist Marquis de Condorcet. In 1785 Condorcet published a paper in which he pointed out that in a race with three candidates, and at least 2 voters, it is possible for the voting process to yield an intransitive set of preferences over the candidates. In other words, we can end up in a situation where a majority prefers A over B, another majority prefers B over C, and another majority prefers C over A, which results in a cycle known as the Condorcet cycle [15].

In 1951, Kenneth Arrow generalized the paradox in his Generalized (Im)possibility Theorem [24] [25]. The gist of the theorem is that there is no method for aggregating individual preferences over 3 or more candidates, which satisfies several precisely defined conditions of fairness and always produces results. Arrow's conditions can be summarized as follows:

- Pareto efficiency: if every voter prefers A over B, then the society prefers A over B.
- Non-dictatorship: no single voter can determine society's preference.
- Independence of Irrelevant Alternatives (IIA): if every voter's preferences between A and B remain unchanged when C is considered, then society's preference between A and B will also remain unchanged.

These conditions, used by arrow to derive his impossibility result, have been scrutinized by other scholars, and suggestions have been made to avoid the impossibility and come up with a satisfactory choice function. One interesting attempt, for the

purpose of this work, is that of Goodman and Markowitz [26], who suggested that the IIA condition be eliminated in order to allow for cardinal utilities to be associated with individual rankings, which can then be aggregated using a number of simple rules, like adding all the utilities and ranking the candidates by the resulting total utility.

Another interesting approach to the problem was proposed by Michael Intriligator in 1973 [16]. Intriligator proposed a probabilistic model of the problem of social choice, in which each individual has a probability distribution over all the alternatives, which reflects the likelihood that she will select each of these alternatives. So, if we have n alternatives, then the preference of individual i is given by the $1 \times n$ vector:

$$\vec{q}_i = (q_{i1}, q_{i2}, \dots, q_{in})$$

If we have m such individuals, the problem is then reduced to that of using the $m \times n$ preference matrix, Q , to compute a vector $\vec{p} = (p_1, p_2, \dots, p_n)$ that satisfies certain properties or axioms, and reflects the likelihood of the society choosing between alternatives. Intriligator suggests a set of axioms and proves that the axioms can uniquely be satisfied using the averaging rule where:

$$p_j = \frac{1}{m} \sum_{i=1}^m q_{ij}$$

Note that according to this model, each probability distribution, individual or social, can imply a ranking over the alternatives if we take $q_{ij} > q_{ik}$ and $q_{ij} = q_{ik}$ to imply a preference of j over k , or indifference between them (respectively). Intriligator points out that when the alternative with the highest probability in the social probability vector is always chosen as a winner, the process is equivalent to what is known as Borda's rule [16].

More recently, Dwork et. al. [15] discuss the Rank Aggregation problem in the context of aggregating search results. Their main motivation is to combine search results from different search engines to come up with a single ranking that is robust to spam, but their approach is general and can be applied in different contexts. In

their setup, they consider a finite set of available documents, and each search engine provides a partial ranking over a subset of these results. The problem is then to use these partial rankings to compute a consensus ranking over all available documents, or at least all documents that appear in the available rankings.

To solve the problem, Dwork et. al. use a precise definition of disagreement that comes from social choice theory, and consider a method known as Kemenization which minimizes this disagreement, and has the desirable spam resistance property. They also prove that the Kemeny ranking, while optimal in some sense, is NP-hard to compute. To address that issue, they suggest a number of heuristic algorithms, and an evaluation of their performance. Among the methods they suggest are the aforementioned Borda's method, an approximation of the Kemenization method, and several Markov chain based methods. In the latter, a Markov chain is constructed using the order in each ranking, in a variety of ways, and a stationary distribution on the chain is computed to determine the aggregate ranking. In their evaluation, they conclude that one of the Markov-chain methods outperforms other methods which seem to be on par, in the context of spam reduction in search results.

Chapter 5

Future Directions

This work is both a continuation of the work that has been done on search, and search personalization, and a beginning for social search at MIT. As it is usually the case with all beginnings, there are some important issues that need to be approached in future work. In this chapter, we discuss future work related to this project. This is just a sample of the directions in which this work may grow.

5.1 Coverage

So far, we have limited our search engine to covering the MIT web, and relied on volunteers to contribute their Facebook personal information. Naturally, we would want to extend this coverage to other institutions, and to more people. In addition to the obvious gain in the quantity of the information, the increase in the size of our social network, and indexed corpus, is likely to improve the quality of our search, and increase the robustness of our inference about user preferences. For that purpose, we are hoping to get more people involved, by addressing some of the concerns that new users might have, and providing support for more popular social networking website that people might want to use.

5.2 Scalability

In order to achieve the desired coverage, our system should be scaled up to accommodate the growth in our social network, and indexed corpus. While our current design is intended to make this procedure simpler, it is hard to tell how it will perform once the system is scaled up beforehand. Moreover, our implementation of the main data structures used by the system is mostly centralized, which presents a challenge for scaling the system up to handle extremely large data sets. To address these problems, a switch to a more distributed architecture might be necessary, especially for the social graph and the index components.

5.3 More Personalized Search

So far, our search personalization has been limited to the use of social networks, but that need not be case in the future. As we have demonstrated, our architecture is suited for accommodating various kinds of information sources, both at the crawling, and ranking levels. We intend to exploit this ability to experiment with a number of areas, such as geographic location based ranking, and temporal ranking.

5.4 Privacy

When it comes to social, and personalized, technologies, privacy is probably one of the most important issues that concern both users and engineers. We understand the reason behind this concern, and believe that a full utilization of this rich information can only be achieved when users are given the ability to decide who gets to access their information, and how much of that information should be accessed for a certain application. We understand some of the challenges inherent in this goal, but we believe that a compromise between utility and privacy can be achieved.

For future work, we intend to experiment with some of the available tools to tackle this problem including cryptography, smoothing, among others.

5.5 Rank Aggregation

Given that efficiency has been our main concern in our approach to the problem of Rank Aggregation, our future work will focus on trying to come up with efficient implementation of better aggregation methods, and experiment with more search sources. In particular, it would be interesting to experiment with the Markov Chain based aggregation methods discussed in [15].

5.6 Learning

In this thesis, so far, we have avoided the important issue of learning: given a “sufficient” number of samples about the preferences of various users, can we learn the underlying structure of the “social network”? For practical purposes, our system eschews this question by depending on manually and carefully constructed social network, which enables us to deal with the problem of search, and provides a base line that can be used to evaluate future learning. Our system also provides a mechanism for logging user decisions to construct a dataset that can be used in the learning phase.

Before we address the learning issue, there are a few questions that need to be addressed, some of which are:

- What is the interpretation, or function, of the social network ?
- Is the current simple graph model sufficient for that purpose? What are the alternative models ?
- What type of data needs to be collected to learn this network?
- What is an efficient way to learn the parameters of these models?
- What guarantees does this learning method give us ?

Bibliography

- [1] J. Kleinberg, “Authoritative sources in a hyperlinked environment,” *Proc. 9th ACM-SIAM Symposium on Discrete Algorithms*, 1998.
- [2] J. Kleinberg, D. Gibson, P. Raghavan, R. Kumar, A. Tomkins, B. Dom, and S. Rajagoplan, “Mining the link structure of the world wide web,” *IEEE Computer*, 1999.
- [3] D. Gibson, J. Kleinberg, and P. Raghavan, “Inferring web communities from link topology,” *Proc. 9th ACM Conference on Hypertext and Hypermedia*, 1998.
- [4] L. Page, S. Brin, R. Motwani, and T. Winograd, “The pagerank citation ranking: Bringing order to the web,” *Technical Report, Stanford University*, 1998.
- [5] L. Page and S. Brin, “The anatomy of a large-scale hypertextual web search engine,” *Computer networks and ISDN systems*, 1998.
- [6] N. Eiron and K. S. McCurley, “Analysis of anchor text for web search,” in *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, (New York, NY, USA), pp. 459–460, ACM, 2003.
- [7] R. Kraft and J. Zien, “Mining anchor text for query refinement,” in *WWW '04: Proceedings of the 13th international conference on World Wide Web*, (New York, NY, USA), pp. 666–674, ACM, 2004.
- [8] O. Buyukkokten, J. Cho, H. Garcia-molina, L. Gravano, and N. Shivakumar, “Exploiting geographical location information of web pages,” *Proc. of the ACM SIGMOD Workshop on the Web and Databases*, 1999.
- [9] T. Joachims, “Optimizing search engines using clickthrough data,” in *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, (New York, NY, USA), pp. 133–142, ACM, 2002.
- [10] G.-R. Xue, H.-J. Zeng, Z. Chen, Y. Yu, W.-Y. Ma, W. Xi, and W. Fan, “Optimizing web search using web click-through data,” in *CIKM '04: Proceedings of the thirteenth ACM international conference on Information and knowledge management*, (New York, NY, USA), pp. 118–126, ACM, 2004.

- [11] E. Agichtein, E. Brill, and S. Dumais, “Improving web search ranking by incorporating user behavior information,” in *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, (New York, NY, USA), pp. 19–26, ACM, 2006.
- [12] Google Inc., “Protocol buffers developer guide.” <http://code.google.com/apis/protocolbuffers/docs/overview.html>, 2010.
- [13] D. Liben-Nowell and J. Kleinberg, “The link-prediction problem for social networks,” *Journal of the American Society for Information Science and Technology*, vol. 58, no. 7, pp. 1019–1031, 2007.
- [14] M. Newman, “Scientific collaboration networks. II. Shortest paths, weighted networks, and centrality,” *Physical review E*, vol. 64, no. 1, p. 16132, 2001.
- [15] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar, “Rank aggregation methods for the web,” in *WWW '01: Proceedings of the 10th international conference on World Wide Web*, (New York, NY, USA), pp. 613–622, ACM, 2001.
- [16] M. Intriligator, “A probabilistic model of social choice,” *The Review of Economic Studies*, vol. 40, no. 4, pp. 553–560, 1973.
- [17] H. Young, “An axiomatization of Borda’s rule,” *Journal of Economic Theory*, vol. 9, no. 1, pp. 43–52, 1974.
- [18] H. Kautz, B. Selman, and M. Shah, “Referral web: combining social networks and collaborative filtering,” *Commun. ACM*, vol. 40, no. 3, pp. 63–65, 1997.
- [19] S. Bao, G. Xue, X. Wu, Y. Yu, B. Fei, and Z. Su, “Optimizing web search using social annotations,” in *Proceedings of the 16th international conference on World Wide Web*, pp. 501–510, ACM, 2007.
- [20] A. Mislove, K. P. Gummadi, and P. Druschel, “Exploiting social networks for internet search,” in *Proceedings of the 5th Workshop on Hot Topics in Networks (HotNets-V)*, 2006.
- [21] M. Bender, T. Crecelius, M. Kacimi, S. Michel, T. Neumann, J. Parreira, R. Schenkel, and G. Weikum, “Exploiting social relations for query expansion and result ranking,” *Data Engineering for Blogs, Social Media, and Web*, vol. 2, pp. 501–506, 2008.
- [22] D. Carmel, N. Zwerdling, I. Guy, S. Ofek-Koifman, N. Har’el, I. Ronen, E. Uziel, S. Yogev, and S. Chernov, “Personalized social search based on the user’s social network,” in *CIKM '09: Proceeding of the 18th ACM conference on Information and knowledge management*, (New York, NY, USA), pp. 1227–1236, ACM, 2009.
- [23] D. Horowitz and S. D. Kamvar, “The anatomy of a large-scale social search engine,” in *WWW '10: Proceedings of the 19th international conference on World wide web*, (New York, NY, USA), pp. 431–440, ACM, 2010.

- [24] K. Arrow, "A difficulty in the concept of social welfare," *The Journal of Political Economy*, vol. 58, no. 4, pp. 328–346, 1950.
- [25] K. Arrow, *Social choice and individual values*. Yale Univ Pr, 1963.
- [26] L. Goodman and H. Markowitz, "Social welfare functions based on individual rankings," *American Journal of Sociology*, vol. 58, no. 3, pp. 257–262, 1952.