

SmartSeek: A Semantic Search in Social Networks

Project Report: CS8803AIA

Ajay Choudhari, Avik Sinharoy, Mohit Jain , Min Zhang.
{ajay.choudhari, avik.sinharoy, mjain8, mzhang7}@gatech.edu
College of Computing,
Georgia Institute of Technology

Contents:

1. Abstract
2. Introduction
 - 2.1 Traditional Search
 - 2.2 Our Approach
 - 2.3 Formal Goals
3. Related Work
4. Architecture
 - 4.1 The Social network interface - Facebook App
 - 4.2 Search and Indexing - Lucene API
 - 4.3 Dictionary and Lexicon - WordNet and LSI
 - 4.4 Communication Method - XML-RPC
5. Detailed Design
6. Results (Screenshots of the results)
7. Challenges Faced
8. Future Work
9. Bibliography

1. Abstract

Social Networks [1] are densely connected web-graphs consisting of people, groups, files connected over the World Wide Web. They can be visualized as a connected graph, where each *node* represents an actor and the edges (or *ties*) represent the association or behavioral relationship between two actors. As the size of a social network grows, problem arises in organization and searching for information in this network. Traditional solution to this problem is to perform a **keyword based** search in this network. However as the data in the network is unstructured and no strict rules govern the format of data, keyword based searching does not help. In this project we tried to address this problem by incorporating **semantic based** knowledge while searching. The basic idea is, instead of relying on user to query the system perfectly, we try to infer what users want to find and restructure the query accordingly to get better results. This prototype implementation of our system is currently developed for popular social network **Facebook**.

Keywords: Social Network, Semantic Search, Facebook Application.

2. INTRODUCTION

Users of social networks often need to find relevant data about other users. The search is conducted over the content generated by target users on the social network (their user profiles, blog entries etc.). However, the search model used is the same as the web, i.e. the user formulates a keyword based query (possibly with some qualifiers) and the system performs a keyword matching over the target users' content to come up with matches. This approach has a high precision but for the user's purposes the recall value might be low. One of the main drawbacks of this method also is that it is highly sensitive to the formulation of the query. In case of advanced searches, the user has to select sub-categories and ranges with precision in order to be able to return expected results.

Our main hypothesis is that this system is highly inefficient and search queries can be made to return better results without constraining the user to formulate 'perfect' queries. Here we discuss the drawbacks of the traditional search system and how our scheme can improve the recall value of searches without burdening the user with learning more advanced search methods.

2.1 Traditional Search

Nearly all social networks have inbuilt search facility. However they are based on keyword matches. This limits them to the ability of user to formulate the query well. However, there are many naive users who are not so perfect in formulating queries. This result in poor recall values.

For example: If a user wants to search for people having taste for sports, he queries the social network by keyword *sports*. This query is rather insufficient to get complete

set of results. It is because sport has many sub-categories such as golf, chess, swimming, tennis etc. Traditional search methods ignore the sub-categories and search is performed based only on the keyword *sport*. Thus searches omits pages which do not have word "*sport*" in it, instead have words "*running*", "*swimming*" etc in them.

There is a related problem of content that also inhibits better search results in social networks. Since the data on social networks is user generated, and it is difficult (or not possible) to enforce a strict structure to it, this data is often difficult to index. For instance, on the Facebook social network, a user is expected to fill out her *User Profile* with entries under categories like *Activities*, *Interests*, *About-me*. Since there is no strict definition (and understandably so) for the contents of each category, the user is free to enter his hobbies (say *running*) under either *Activities* or *Interests* or even in the *About-me* category. There is also no formatting of the contents, making the entries in these fields difficult to search over.

2.2 Our Approach

We aim to increase this recall value by inferring the semantics of a query and adding the query with more relevant keywords before we pass it on to search engine.

A typical social networking service like Facebook provides a simple look up service to search a friend or group just based on keyword search. It's a problem for user if he wants to semantically search for a friend or a group. Consider the following two scenarios:

- i. Suppose a user wants to search for "*people who love sports*" - social network sites' search answers this question by searching for the keyword "*sports*" in the profile description of the users. But if a user mentions particular instance of sports - for instance playing basketball - as his hobby, then the search for keyword sports will not match his profile, even though "*playing basketball*" is a sport.
- ii. Another scenario where a keyword search fails completely is querying, i.e., if a user wants to search for people between the age group 21-25 years. Then present search won't understand the semantics of the query and will return unwanted/random results.

Clearly there is a substantial gap in the capabilities of social network search functionality and the requirements of user queries.

Our project aims to develop an application which sits on the top of a social networking website and provides semantic search capabilities rather than simple keyword matching.

2.3 Formal Goals

The goals of the project can be formally stated as:

- To increase the recall of the traditional keyword based search mechanism.
- To reduce the requirement of forming 'perfect' search queries for the user.
- To interpret the user's intentions through a specialized dictionary look up, so that the user need not provide all possible variations of the search term.
- To explore the requirements of a specialized lexicon that can return relevant semantic matches for a given area of expertise.
- To provide a generic framework independent of the constraints of a particular social network and its access API.

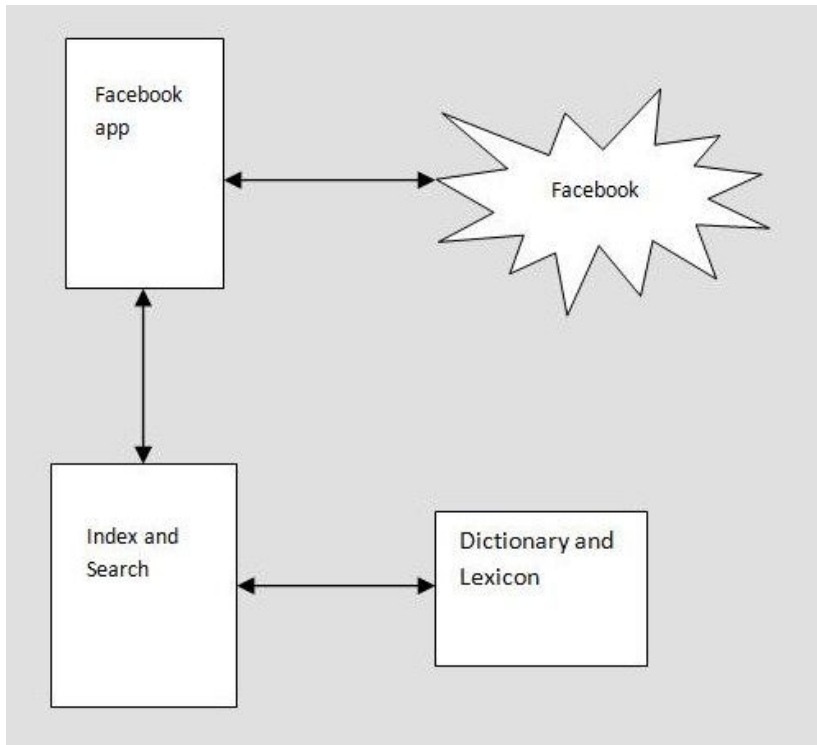
3. RELATED WORK

Current directions of web growth focus like web services and the semantic web focus on creating a web of distributed machine understandable data. TAP[2] provides an application framework upon which the semantic search is built. The paper described two implemented Semantic Search systems which are based on the denotation of the search query, augment traditional search results with relevant data aggregated from distributed sources. Lots of effort has been done in semantic knowledge extraction by search engine in web search like [3], [4] but a similar semantics-search oriented work has not been implemented in the sphere of social networks.

4. SYSTEM ARCHITECTURE

One of the main focus areas of our design has been to keep the various components of our system fairly independent, as each of them perform a specialized function that might be plugged-in or out (for instance in preference for another technology for search and index mechanism). In particular we have tried to limit the interaction and dependence of the rest of the system with the Facebook application that is used to interface with the Facebook social network and is the main interface of the search application.

Smart Seek architecture broad diagram

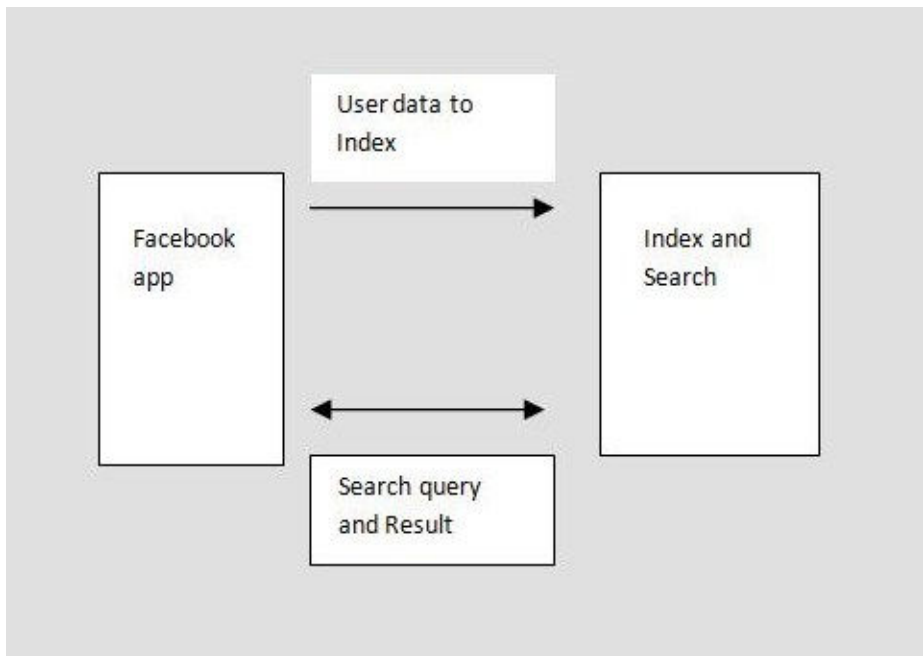


4.1 The Social network interface - Facebook App

The social network interface is implemented as a Facebook application. This application has the following main functions -

- To allow users of the social network to find and use the application
- To allow users to sign up for the application and to record their user data for the indexer
- To provide the UI for executing a search query
- To provide results with relevant links for every search query
- To interact with the Search and Index module and provide search queries and user database entries to it.

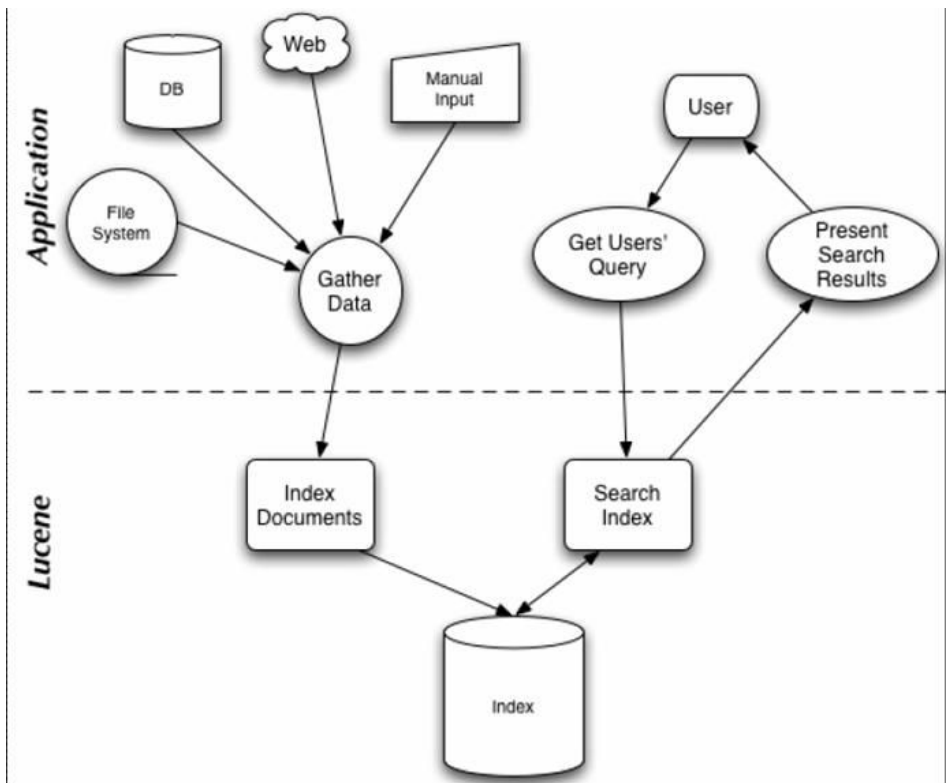
In order to maintain independence of the rest of the system from the Facebook module, the details of a Facebook user profile as well as the interaction with the Facebook API is kept hidden from the rest of the system. There are only two API points between the Facebook application and the Index and Search module as shown in the diagram below.



The details of the interaction of the Facebook app with the Facebook network, as also the interface details are provided in the detailed design section.

4.2 Search and Indexing - Lucene API

For the setting up the searching and indexing facility we have used Lucene API. It's a software library and concerns with text indexing and searching. Also It's not a ready-to-use application like a file-search program, a web crawler, or a web site search engine.



First the data needs to be indexed, so that a systematic search can be performed on it. Using Lucene we can index the pages/documents and **here Indexing breaks down into three main operations:**

- i. Conversion from data to text: The data (of different formats like database, doc, web pages etc) is converted to text files.
- ii. Analyzing/stemming: In this method stemming is performed, i.e., data cleansing which requires removal of stop word and converting the words into their base words.
- iii. Saving it to the index: The documents are then stored as an inverted index.

After successful indexing of the documents, **we can perform efficient search on that, and this involves following 3 steps:**

- i. Parsing the Query: The input query is parsed.
- ii. Analyzing the Query: The query also undergoes through the same procedure of stemming and stop word removal.
- iii. Search in inverted index: The query got in the above step is searched into the indexes (inverted) of the indexed documents.

The above procedure of indexing and searching can be showed as in the following figure.

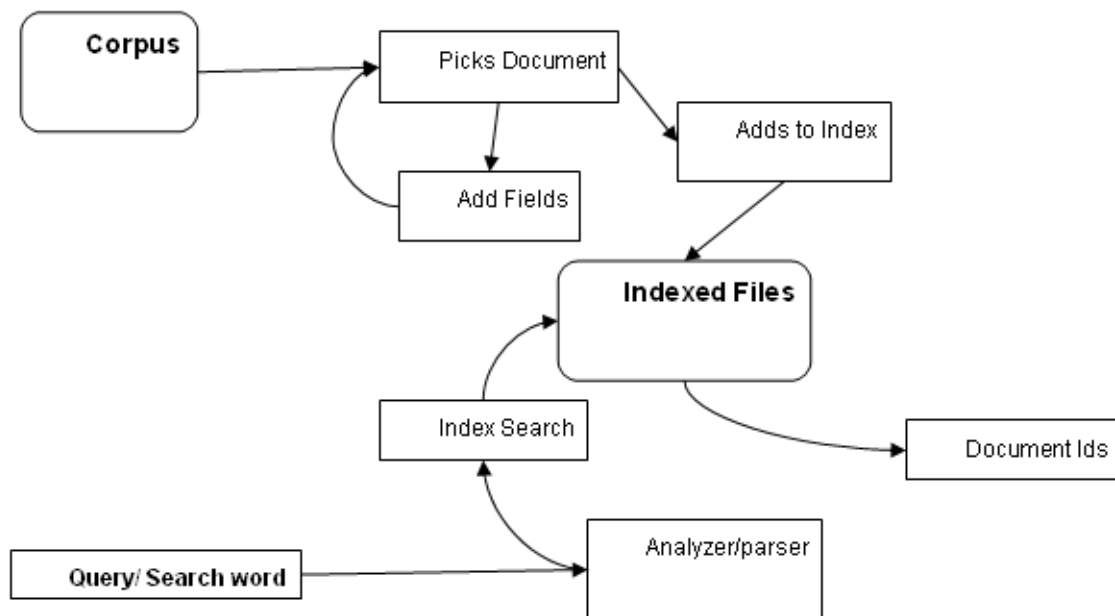


Figure: Indexing and searching in the Lucene.

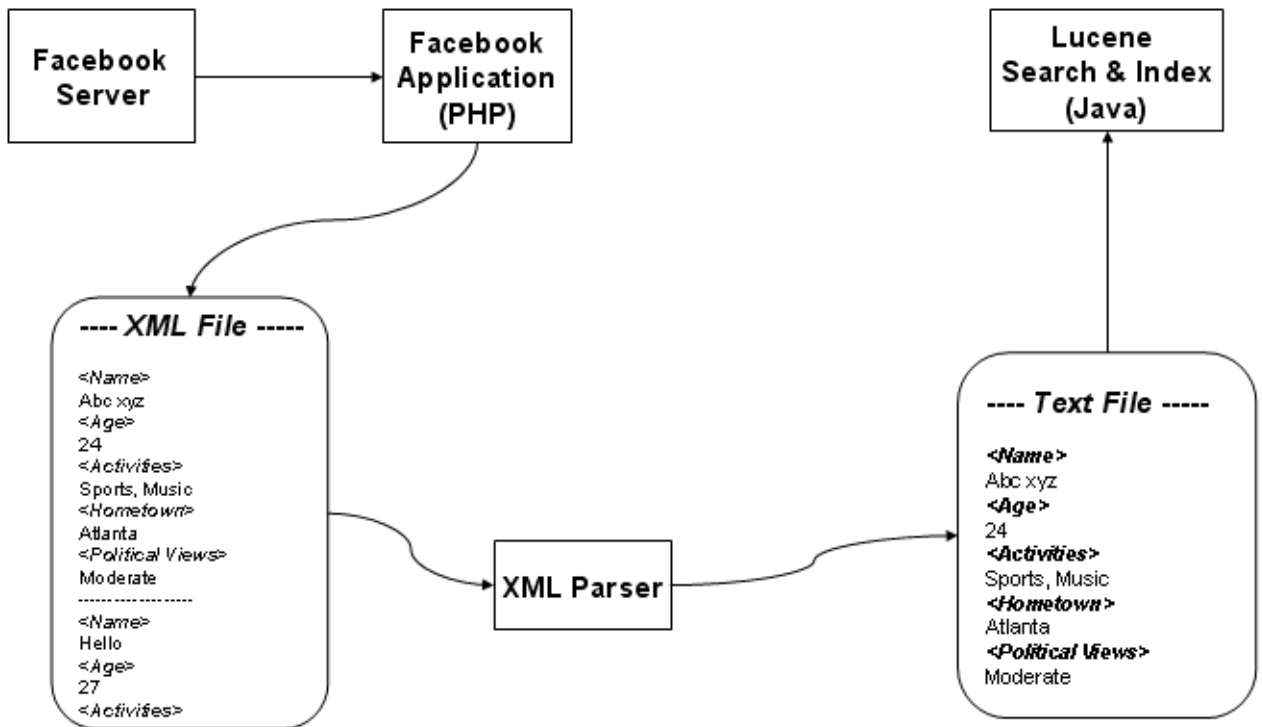
Some properties of Lucene utilized for semantic search:

- i. Analyser : Eliminates the stop words & stores words in it base form.
- ii. Keyword : These are the specified words which are not analyzed/stemmed, but these are indexed, e.g. we don't want Technologies in "Agilent Technologies" to be

- stemmed as Technology because it's a company's name.
- iii. Updating the indexes on regular basis : If a document needs to be updated then that Document needs to be deleted from an index and then re-added to it
- iv. Search Facility extended:
 - a. Keyword1 AND/OR Keyword2 : The Boolean search facility can be utilized for the multiple query terms.
 - b. + Keyword1 – Keywords2 (Extended) : This can be found out by subtracting the documents containing keyword2 from the list of documents containing the keyword1.
- v. Ranking formula for results:

$$\sum_{t \text{ in } q} tf(t \text{ in } d) \cdot idf(t) \cdot boost(t, \text{field in } d).$$

This formula takes care that if a single document contains lot of search related terms above a threshold (which is usually a case of spamming), then that document doesn't occur quite high in the results.



The above diagram shows how the data comes into the system for indexing and afterwards similarly the query comes in the Lucene search engine.

4.3 Dictionary and Lexicon - WordNet and LSI

The dictionary or lexicon serves as the pool of keywords, from which we could extract the words with close relation with the query.

We crawled around 10,000 documents. The dictionary is formed by extract all the words in the document but not in our stop word list. The initial dictionary consists of 60,000 keywords. After removing the uni-coded strings manually, we finally get the lexicon with size 11,625.

The semantic relation is detected by analyzing the magnitude of documents. We generate the term-document matrix with respect to our lexicon. Since the this matrix have very high size and is very sparse, we store the term-document matrix in a linked list(sparse matrix) instead of an array.

Our program will output 3 files which indicate the row index, column index and entry value. These 3 files could be used as inputs to generate a sparse term-document matrix in Matlab. Using the Singular Value Decomposition (SVD) of sparse matrix in Matlab, we generate a left singular vector U of the term-document matrix. Actually, SVD of sparse matrix only calculates the first 6 columns of the singular vector, it is an approximate method. The relatedness score between each pair of terms are calculated by $U*U'$.

The relatedness score matrix is really a large matrix, the file we generated to store this matrix is 1.2 Gigabytes. It is not feasible to load this file when our API runs.

As an alternative, we calculate the index of top 100 words that has the highest relatedness score to a given query. This index matrix requires much smaller memory space than the relatedness score matrix due to two main reasons: 1). the entry in index matrix is int type instead of double, 2). the size of the index matrix is only 1% of the relatedness score matrix.

When the LSI Java API runs, the program will automatically load two files: lexicon file and index file in to memory. The user could feed any query to the API, then the API locates the word in the lexicon, finds the indices of the most related words by the index matrix, then look up the words in the lexicon given the indices, return the most related words.

In fact, all the LSI process is pre-calculated, which means, the relatedness score are calculated before, all the information are stored in file and our API will load the file and then return the related words based on the file.

4.4 Communication Method - XML-RPC

The interaction of the Facebook app and the Index and Search modules involves a small challenge in the volume of data that can possibly be sent over this link. In order to provide a simple yet powerful and extendable implementation, we have chosen to go with XML-RPC standard for communication. The Index and Search module runs an

XML-RPC server (using the Apache implementation of XML-RPC for Java) and publishes the methods for updating the Index and performing a search. The Facebook application uses the PHPXMLRPC package available in PHP5 to run an XML-RPC client that encodes the user data and sends it over an HTTP POST request. A sample encoding of XML-RPC on the wire looks something like this:

```
<methodCall>
  <methodName>sample.sumAndDifference</methodName>
  <params>
    <param><value><int>5</int></value></param>
    <param><value><int>3</int></value></param>
  </params>
</methodCall>
```

5. Detailed Design

Facebook App

The Facebook application uses the Facebook PHP API and is developed using the same language. The application is hosted on a public IP and Facebook application is registered under the application name *SmartSeek*.

The database

The user data is cached in the Facebook application as a MySQL data table, and contains the following entries:

Field name	Type	Allow nulls?	Key	Default value
<input type="checkbox"/> userid	int(11)	No	Primary	
<input type="checkbox"/> Name	varchar(15)	No	None	
<input type="checkbox"/> bday	date	Yes	None	
<input type="checkbox"/> sex	enum('m','f')	Yes	None	
<input type="checkbox"/> hometown	varchar(20)	Yes	None	
<input type="checkbox"/> religion	varchar(10)	Yes	None	
<input type="checkbox"/> political	varchar(10)	Yes	None	
<input type="checkbox"/> activities	text	Yes	None	
<input type="checkbox"/> interests	text	Yes	None	
<input type="checkbox"/> favMusic	text	Yes	None	
<input type="checkbox"/> favShows	text	Yes	None	
<input type="checkbox"/> favBooks	text	Yes	None	
<input type="checkbox"/> favMovies	text	Yes	None	
<input type="checkbox"/> favQuotes	text	Yes	None	
<input type="checkbox"/> about	text	Yes	None	
<input type="checkbox"/> profileUpdateTime	datetime	Yes	None	

The `userId` provided by Facebook for each user is used as the Primary key for this database. In designing the database we are also faced with some problems unique to the Facebook environment. The Facebook platform privacy restrictions prevent the profile data for un-registered users from being stored in the application for more than 24 hours. As a result we maintain a timer for the last DB update time in the application. Then we run a server-side cron job to find the database entries (and hence the users) for whom the profile update is due (by comparing the DB update time with the current time). If a user profile has become stale the cron-job then visits the user's profile and reloads the data through the Facebook API. In order to reduce the load from the application on the Facebook server, we also limit the number of sequential updates in a single invoking of the cron-job to 10. This way the stale entries might take longer to update, but that is trade-off in order to preserve the network usage.

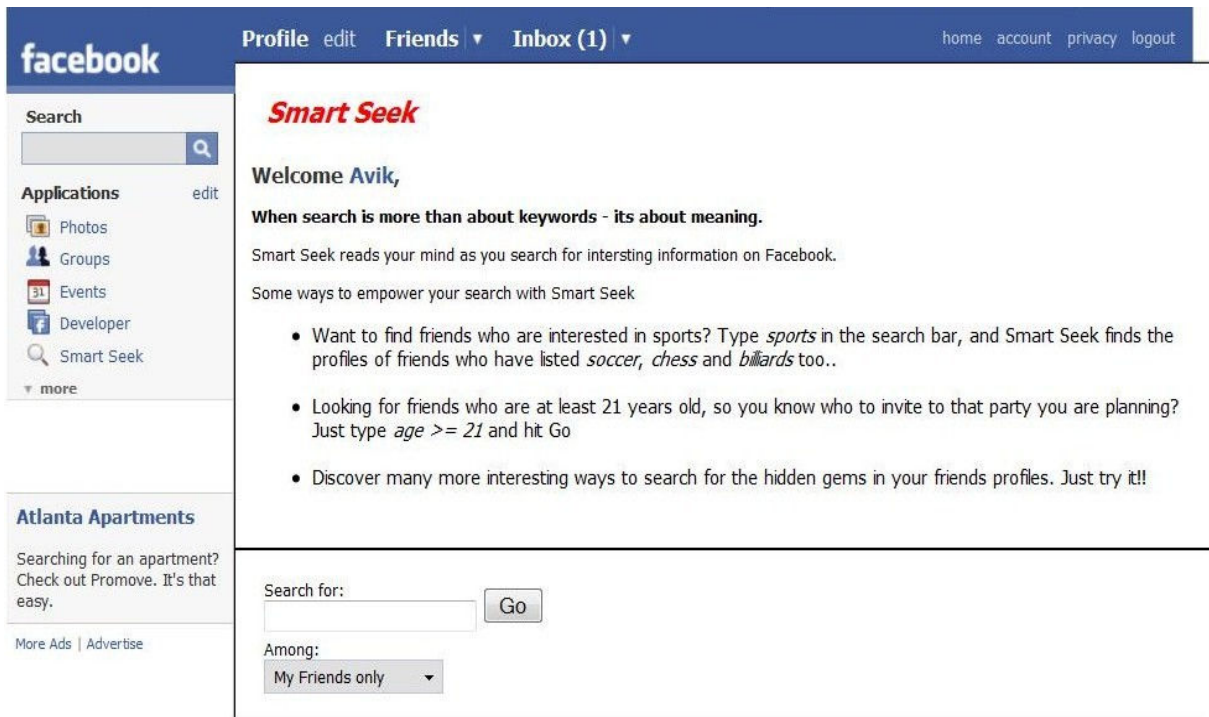
Data formatting and lack of guarantees from Facebook

The Facebook user data is available in certain categories as defined by the API and Facebook user interface. These categories however do not always have strict formatting restrictions. For instance the *Name*, *FirstName* and *Sex* categories have strict formatting, but the birthday and the rest of the categories do not have strict formatting. This causes a problem with the application, since we cannot run checks of the age of the user if the user has not entered the birth year (or if it is not visible to the application because of privacy restrictions). Currently we format the date and time entries by converting them to a timestamp based on the assumption that full date and time values are present. Consequently, if the user has not entered the birth year for instance, the value will default to the current year and a search on the age category for this user will probably result in a mismatch.

The search interface

Since this application is not meant to provide the experience of a full-fledged Facebook application, we have skipped the use of some of the integration points. In particular we do not use the *profile page* of the user (as it does not make sense to publish search results or interface there). Instead the only integration point provided is a link to the application canvas page in the *left-nav* bar, and an icon link below the profile-picture for the user. The intention here is to provide a simple clean interface that even Facebook users suffering from 'application-overload' will not mind adding to their profiles.

The canvas page is the main page for the application and provides a small introduction to the user. It also provides the search form and the results (if any) below the form. The search interface for the Facebook application is shown below -



Communicating with the Index and Search module

The Index and Search module requires two services from the Facebook application, namely - to provide the user data and to provide the search queries. It returns the search results as a return value for the XML-RPC method call for the Search function.

How "Query" travels in the system

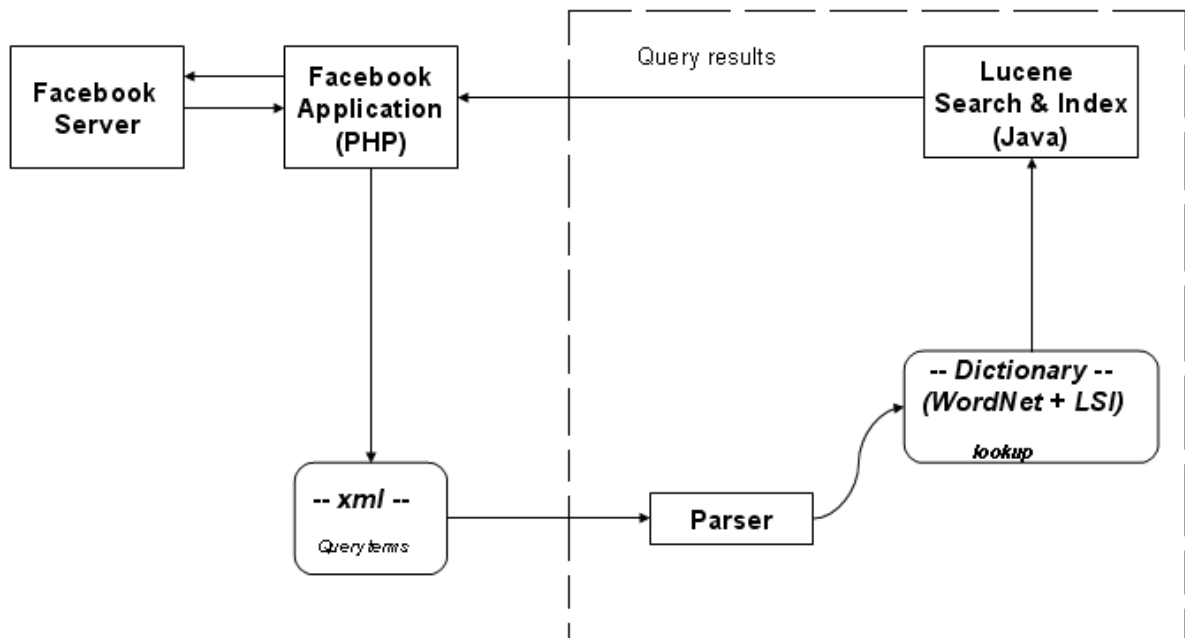


Figure: How "Query" travels in the system.

The above figure shows how query travels through the system. Users who are registered to the SmartSeek application inputs a query in the Facebook. The Facebook server sends that query to our application server, this query (through XML RPC) comes into our searching client where it is parsed. The parsed query is then searched into our "Dictionary", which is a combination of "WordNet online dictionary" and "dictionary formed by using LSI method based on the relatedness between the words". Here words which are semantically related to the query terms are fetched and now this array of words are searched in the Lucene indexed documents (which are basically Facebook crawled profiles). The top 5 documents/profiles matching the "semantically related search keywords" are returned to the Facebook application, which in turn returns the results to the Facebook where these are prompted.

6. Results

7. Challenges Faced

1. Communicating with external server: The server on which we hosted the facebook application did not have a JAVA stack. So we were not able to put our dictionary and search

code on the server. We require to communicate with some other server which can run JAVA machine and has static IP. But we were not able to get one from school. Then we tweaked little bit. We hosted one server at our home. Since we were not given static IP, we registered a domain name on some server. We ran two scripts, one on that server and one on home computer. Both scripts communicate with each other periodically and updates the computer's IP address at the server where application was hosted. Thus we simulated an static IP without actually having it. This was a crucial part of our application because we would not be able to run anything without JAVA.

2. Crawling Data: Our application needed lot of data to build a dictionary. For that it was necessary to crawl a good amount of data from web. But the crawling has a strict requirement. We need very specific data. For this we had two approaches, either crawl it manually and gather relevant data. But this approach was not sufficient. So the second approach was to crawl a lot of data. This way we can have lot of related words and then we can filter out the unwanted words. This looked fine, but there was a small catch in this approach. This way matrix size would reach to a maximum and there would be an overflow. So we decided for a tradeoff. We crawled few pages manually and then plugged in our crawler for gathering more data so that we may crawl only a subset of data not all.

3. Gathering user data in the absence of user activity: It is important for the usability of this system that user data be always available to the indexer, even when the facebook user has not logged in to the application in some time. This includes the requirement for capturing changes to the user profile in the absence of a login to the application. Since facebook API requires a user session (and hence a user login) to provide data access, we used the developers user id to create an always on session. This session is used by the server side cron job explained earlier to query the facebook API for updates to the user profiles stored in the database.

4. Scalability and lexicon size

One of the problems we are facing is that the size of lexicon is really large, it is around 60,000 words before we did some preprocessing. The method we currently used to reduce the lexicon size is removing the words with low frequency. It is simple but not so good as we expected since occurrence of the words is the unique criteria for keywords.

5. Use of word stems in Lexicon

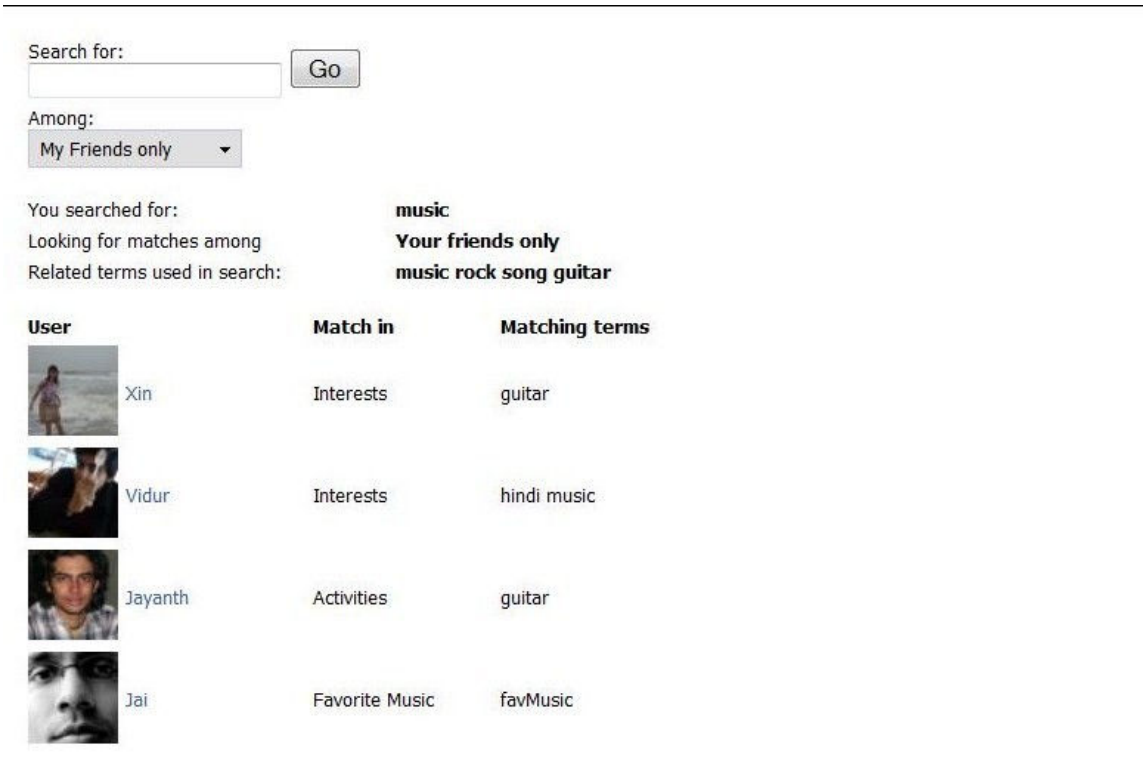
One reason for the large size of term-document matrix is that we do not use stemming when we build our lexicon. Potential benefits applying stemming in our approach include the reduction of the size of the term-document matrix and more accurate calculation of relatedness score.

By using stemming, the size of lexicon does not change, but the size of term-document matrix will reduce apparently. E.g., without stemming, the "accept", "acceptance", "accepted", "accepting", "accepts" will occupy 5 dimensions in our final term-document matrix. While stemming is applied, only 1 dimension would present in the final term-document matrix.





Another benefit of stemming is that we could improve the accuracy of relatedness score. For the above example, using 5 words -- "accept", "acceptance", "accepted", "accepting", "accepts" -- will weaken the relatedness scores since we distributed these scores into 5 words. Actually, the five words are originated from "accept" and have the identical meaning in most scenarios. We could deem all the above words as one word -- "accept", all the other 4 words relatedness score could merge into one. By this way, a more accurate relatedness could be calculated.

6. Results

The search application can be used from the facebook application canvas page to perform queries over the entire database of registered users or only in the network of the user's friends. The following screenshots show two queries and the associated responses.



The screenshot shows a search interface with a search bar containing 'music' and a 'Go' button. Below the search bar, there is a dropdown menu set to 'My Friends only'. The search results are displayed in a table format with columns for 'User', 'Match in', and 'Matching terms'. The results are as follows:





User	Match in	Matching terms
 Xin	Interests	guitar
 Vidur	Interests	hindi music
 Jayanth	Activities	guitar
 Jai	Favorite Music	favMusic

The screenshot above shows the results for a search with the keyword *music*. Note that the application finds matches among friends' profile entries that have the keyword as a whole word, as part of a word and also matches for the semantically related terms. The semantically related terms are also displayed to help the user understand the search results. One of the major difficulties in our approach is to ensure that the correct semantic matches are chosen from the list of related terms. Below is another example for a search with the keyword *sports*.

Search for:

Among:

You searched for: **sports**
 Looking for matches among **Your friends only**
 Related terms used in search: **sports soccer football swimming**

User	Match in	Matching terms
 Xin	Activities	playing soccer
 Gautam	Interests	Sports
 Priyam	Activities	sports
 Archisman	Interests	playing Football

7. Future Work

Coupling semantic search with existing search enhancements

One of the ways of immediately making semantic search usable and user-friendly is to couple it with existing search enhancement schemes like relevance feedback on search results. Semantic matching is a tricky art, and it would need to be tuned to particular users to make it more usable. This is because we are using generic text (blogs and Wikipedia articles) to train the lexicon. Thus while the word *sport* might be semantically related to many generic words, a particular user might find only some of these words (possibly with a low *relatedness score*) useful. In these cases the standard search method and our method will not really produce any different results for the user. However, training the search module on the basis of relevance feedback from the user can be used to provide better results to the user at the next search by selecting terms based on previous choices.

Performance testing for semantic search

At present our prototype implementation is not in a position where we can do search

performance testing. However, in order to make any substantial claims about the benefits of our method, we do need to get search recall values for semantic search as opposed to traditional search for a wide variety of search topics. Since our lexicon is trained on specific topics and areas of expertise, it tends to perform poorly in other areas.

Relevance sorting of search results

Sorting search results by relevance is a particularly difficult problem in social networks. Is a match with a *friend* more important than one with an *acquaintance*, even though the relatedness score of the matching term might be in the reverse order? Similarly, the relatedness score is only a guiding value in selecting terms we should search over, and it cannot be directly assumed to be the order of preference of the user when it comes to viewing and selecting search results. These and other questions like them have been encountered by us in the process of implementing this prototype. Due to limitations of time and the complexity of this problem, we have decided not to focus on this issue for the prototype. However, for a usable system this is a very important feature, and we would like to incorporate it in our prototype in the future.

9. Bibliography

[1]. Mohsen Jamali, Hassan Abolhassani; Different Aspects of Social Network Analysis; *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*.

[2]. R. Guha, Rob McCool and Eric Miller; Semantic Search; World Wide Web Conference, May 20-24, 2003, Budapest, Hungary

[3].Bahadorreza Ofoghi, John Yearwood, Ranadhir Ghosh; A semantic approach to boost passage retrieval effectiveness for question answering. *Proceedings of the 29th Australasian Computer Science Conference - Volume 48*.

[4]. Fuchun Peng, Nawaaz Ahmed, Xin Li, Yumao Lu; Context sensitive stemming for web search; *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*.