Fall 2012

SEMANTIC DISCOVERY THROUGH TEXT PROCESSING

Bieu Binh Do
San Jose State University

SEMANTIC DISCOVERY THROUGH TEXT PROCESSING

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

By

Bieu Binh Do

December 2012

The Designated Committee Approves the Writing Project Entitled

SEMANTIC DISCOVERY THROUGH TEXT PROCESSING

by

Bieu Binh Do

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

December 2012

Dr. Tsau Young Lin	Department of Computer Science
Dr. Robert Chun	Department of Computer Science
Dr. Howard Ho	IBM Almaden Research Center

ABSTRACT

SEMANTIC DISCOVERY THROUGH TEXT PROCESSING

by Bieu Binh Do

As the world embraces the digital era, unprecedented volumes of information are generated and consumed daily. It becomes difficult to comb through mountains of documents to locate search topics. With inherent ambiguity in human languages, conventional methods using straight text pattern match cannot resolve words having multiple meanings and often misinterpret user intent. There is a need to develop a system able to identify the target topic and return quality relevant links, ending the tedium of rummaging through piles of unrelated links that may get lost in the rubble. An example search of the words "sound investment" helps to illustrate this point. Both Google and Bing return result sets that disorderly interleave musical services and financial planning links, two very different subject matters. User is left to cherry pick manually among the results for the intended links. To combat this problem, this project seeks to develop a new automated methodology for classifying web content by semantics, featuring machine learning capability that can adapt to a rapidly changing environment. This will enable a new type of search engine that organizes results according to related topics.

ACKNOWLEDGEMENTS

I would like to thank Dr. Tsau Young Lin, my teacher and advisor, for the honor of working with him on this innovative project.

I'd like to thank Dr. Robert Chun and Dr. Howard Ho for their gracious participation in my thesis committee.

Finally, I'd like to thank my wife Cathy for encouraging me to return to school to pursue my advanced degree.

TABLE OF CONTENT

1 Introduction

The enormity and wealth of information available on the net is undeniable. Searching for a topic can end in frustration when it returns pages upon pages of disorganized results to scan through until the relevant ones emerge. We are hitting the limitations of the traditional search engine model, which are not semantically aware. By using primarily pattern matching to index web pages, conventional search models cannot automatically detect keyword semantics and therefore cannot organize results into meaningful groups.

The purpose of this project is to develop an automated clustering technology that self discovers document semantics through textual analysis. Through cumulative machine learning by continually crawling the web and processing new documents, it will build up a comprehensive knowledge base capable of supporting semantic aware applications. The obvious first of such applications is the semantic aware search engine that automatically classifies web pages by meaningful topics. This innovative approach extrapolates semantic knowledge without requiring web sites to alter or augment their HTML pages with special tags. Working with existing web content, this passive approach to semantic learning is a major advantage over methods which require authors to tag their web pages with special constructs or parallel metadata publishing, such as using the Resource Description Framework (RDF). Such manual techniques are labor intensive and are error prone.

The methodology discussed in this paper is based on Dr. Tsau Young Lin's research paper on Granular Computing in 2005 (Lin and Chiang) and subsequent work in 2008

(Lin and Hsu). He proposed semantic knowledge can be captured by using the geometric structure called simplicial complex and explained how to apply it in actual implementation. This project focuses on the application of that theory and delivers working software that illustrates the methodology and analyzes the results as compared to those from Google.

2 Ordered Simplicial Complex to Model Human Concepts

In Dr. Tsau Young Lin's research paper on Granular Computing (GrC), he transforms the knowledge learning problem into a mathematical model. He proposed the semantics of a document can be captured using the simplicial complex structure. Built upon processing comprehensive text documents, this structural model then becomes the knowledge base from which to build a semantic search engine.

The term *human concept* is used to represent an idea, an event, a topic, an abstract, a unit of knowledge. We propose a method to capture human concepts through textual analysis of regular documents. The basic premise of the theory is to scan documents to mine for high frequency co-occurring set of keywords that appear in the same paragraph or are close to each other, forming a list of *keywords* call *keyword sets*. Closeness is defined by works appearing within the same paragraph or within a predefined maximum distance, which is a word count value that determines how far apart words can be in a document and still be considered co-occurring. In this way, keyword sets are not required to appear consecutively in the document. They just need to satisfy this closeness property. When a keyword set appear in significant frequency in many documents then it

becomes a human concept. In the simplicial complex model, the keywords are mapped to vertices and the keyword sets are mapped to groups of vertices connected by edges to co-occurring keywords. Each keyword set thusly forms an *n*-simplex (also called granule), with *n* denoting the number of points (keywords) in the granule. Further, we only collect the maximal *n*-simplexes in the graph in our final structure. A maximal simplex is defined to be a simplex that is not the face of another simplex. Illustrated in Figure 1 is an example of a partial simplicial complex structure that can be derived from a set of documents relating to social networking study in a university. The following simplices are found:

1) Two maximal 3-simplices (tetrahedron)
    - computer science department university
    - online social network analysis
2) Three maximal 2-simplices (triangle)
    - computer network department
    - computer network application
    - social network application
3) Three maximal 1-simplices (segment)
    - network cluster
    - cluster coefficient
    - mobile application

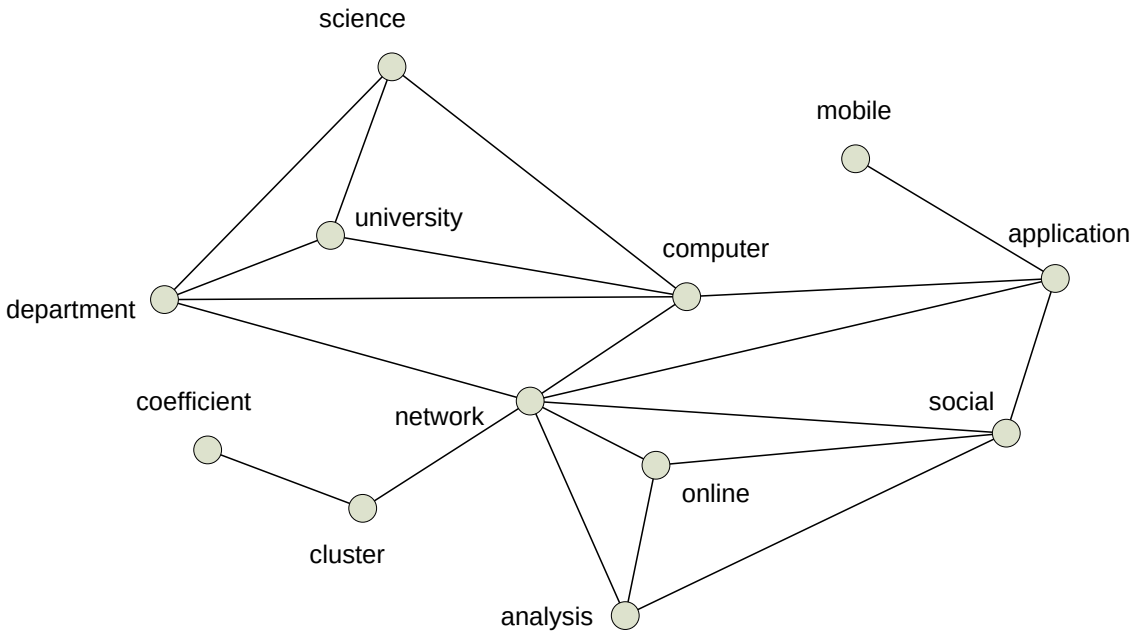


Figure 1: Keyword Simplicial Complex

2.1 Text Ordering

There is an underlying order assumption made that is not illustrated in the graph. Observe that in real textual content, keyword (vertex) order matters. Most of the time, a concept made up of several words only makes sense in one unique relative order. Therefore, an *n*-simplex built from real text usually only originated from *n* words that were used in a unique relative order within a paragraph, or otherwise the same set of words is unintelligible and probably never occurs in real text. For example, "application mobile" is not a normal way to speak about "mobile application," at least not in the English language. In the above graph example, the 3-simplex "online social network analysis" was derived from a document which had a paragraph using these words in the following context and relative order. Here is the excerpt:

> The research of online social relations has indicated that there exist different levels of engagement between users
> …
> The social network under analysis was first identified using the SMS and phone call logs of the handset-based measurement data.

In this example, the four words were formed from multiple sentences that were close together and were advanced as a human concept because the same four words occurred in sufficiently high document frequency. We argue the same four words would not have been derived from text with alternative relative order such as "online analysis network social," which is non-sensible in normal English speech. That relative sequence may still occur in some documents but existence of such sequence will be coincidental and will not occur in significant frequency to elevate it into 3-simplex structure in the way that the natural English language order did.

However, there are some cases where a keyword set does have intelligible and meaningful semantic when arranged in alternative ways. As another example above, the 2-simplex "social network application" takes on a different meaning when rearranged to "social application network." So it is conceivable another set of documents with such meaningful alternative word arrangements can generate the same 2-simplex structure. Therefore, we must account for that and preserve the order property in our graph. To that end, we are employing ordered simplicial complex so that such situation will result in two separate simplices. This is one key advantage of our methodology over conventional search engines because it is able to capture and distinguish such ambiguities. For

uncluttered clarity, the ordered property is not illustrated in the simplicial complex graph above but should be assumed to be part of the stored information in each simplex.

2.2 Keyword Simplicial Complex

We call the simplicial complex graph the Keyword Simplicial Complex (KSC) (Lin and Hsu) or equivalently for this paper, the knowledge structure. Dr. Lin's theory implies that two documents of similar content would have the same or similar KSC. Furthermore, documents with overlapping ideas will also share similar portions of the KSC. Using this notion, we can build a comprehensive knowledge base by accumulating and combining knowledge graphs from scanning many different documents.

The main focus of this paper is in the construction of such a comprehensive knowledge base. We envision real world construction of such knowledge base to be scanning all documents on the Internet and constantly monitoring for new additions to ingest. The more documents ingested, the more complex the structure becomes and the more knowledge it holds. From this extensive knowledge base, then smarter and semantically aware applications can be developed.

This paper will also demonstrate the usage of the knowledge base in two applications: a semantic search engine and in an examiner utility that attempts to decipher the main topics of a previously unseen document.

3 Algorithm

3.1 Techniques from data mining

3.1.1 Apriori

There are a few existing data mining techniques that are utilized by this algorithm. In determining the frequency of a keyword set, the Apriori principle is used to quickly eliminate keyword set candidates that can never become frequent. Because Apriori states that a set's frequency of occurrence cannot be more than any of its subsets, it follows that infrequent subsets implies infrequent superset. This notion is applied during candidate generation. With 2^d possible subsets within a distance of *d* keywords, this theory helps to eliminate many possible candidates, reducing the computation and storage requirements by orders of magnitude.

3.1.2 Support

Another definition that comes from existing data mining techniques is the measurement of support of an association rule. This value ranks the relative frequency among all of the keywords found.

For this paper, the support of a keyword or keyword set is the percentage of scanned documents that contains the keyword set. The keyword set is formed and retained when its support satisfies a minimum threshold value.

Let keyword set $K = \{k_1, k_{1, \ldots,} k_n\}$,

$$support(K) = \frac{\#documents\ containing\ K}{\#all\ documents}$$

The *confidence* measure in data mining is not important and not used in this methodology.

3.1.3 TF-IDF

Term Frequency-Inverse Document Frequency (TF-IDF) is a basic measure of not only the frequency of occurrence of a term but it also attempts to rank its importance. Based on this formula, a term may be accepted when it exceeds a frequency threshold minimum but may be eliminated if it becomes too frequent. We modify this formula to apply to a keyword set.

Let keyword set $K = \{k_1, k_{1, \ldots,} k_n\}$ and let d be the document containing K
Let D be the set of all documents, $d \in D$
Let frequency F(K, d) to be count of K in document d

$$TF(K,d) = \frac{F(K,d)}{\max\{F(any\ K,d)\}}$$

$$IDF(K,D) = \log\left(\frac{\#\ documents\ D}{\#documents\ containing\ K}\right)$$

$$TF - IDF(K,D) = TF(K,d) * IDF(K,D)$$

As seen in the TF-IDF formula, there is a penalty to a term occurring in too many documents. This may seem counter-intuitive but the reasoning is that a common phrase (stop words, conjunctions, articles) that appears in every single document is not important and therefore its IDF approaches zero as it approaches 100% occurrence, consequently resulting in zero TF-IDF regardless of the TF term.

In our algorithm, the concept of TF-IDF is used at different stages of the algorithm to help prune candidate keyword sets.

3.2 Software Design

3.2.1 Top level design

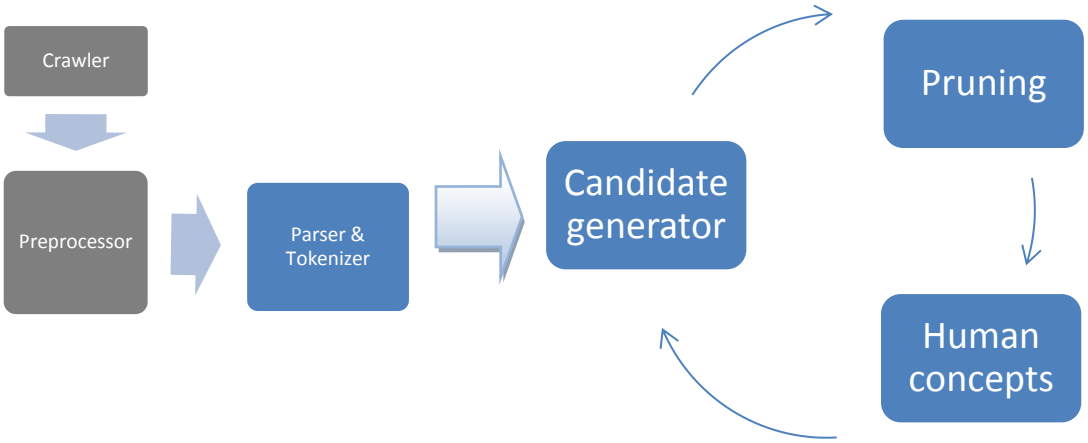


Figure 2: Processing stages

The software (herein also described as the system) work flow diagram presents a high level view of the key software components, illustrating the basic processing stages. The processing stages to the right of the parser form the core of our design model and perform the important work while the others are compacting and priming the data.

3.2.2 Crawler

The Crawler is a process that goes out and scans websites and downloads their files into the local drive for processing. Although the Crawler is not an integral part of the algorithm, for completeness, it is shown here as an entry point. This project does not develop the Crawler and this project obtained a Crawler implemented by Dr. Lin's past classes.

3.2.3 Preprocessor

The input data to the system is a set of documents provided by the Crawler. The chain of processes from the preprocessor and on to the final human concept discovery

constitutes the software system that was developed for this project. There are two types of files encountered as input to this software: HTML and plain text. Strictly speaking they are both text files.

Raw HTML data from the web will typically contain formatting information, layout information, margins, hypertext, and other metadata that must be removed before feeding it to our algorithm. This extraneous noise in the document, i.e. tags like <html><link>style><bold><href>
, contribute nothing to the knowledge base and would waste computation cycle and storage. At worst these taints and distorts the knowledge base.

Since we are concerned about a paragraph of text at a time, it is important to be about to identify paragraph boundaries. This is a problem in both HTML and regular text files. The actual text content is free flow and the author could have used any number of ways to demarcate a paragraph. While a person reading a text document can visually determine the beginning and end of a paragraph by cues such as blocks of text appearing together, or double space between paragraphs, or indents, it is not easy for a program to do so. If author chose to use a carriage return on every line, then programmatically, every line may be detected as a separate paragraph. HTML tags can be used to detect paragraph boundaries but that is also not used consistently nor enforced by the HTML protocol, so it can't be relied upon.

We still need to normalize the data to ensure the downstream stages are fed the expected kind of data so they can spend their energy on the main functional processing rather than having to anticipate errant inputs. In both types of files, proper preprocessing

requires some amount of heuristics and guesses to achieve good results. Because HTML is a very loose standard and is not semantic aware, there is no easy way to extract only the relevant content and throw away the junk data such as advertisements, site navigation menus, command contacts information, and other irrelevant snippets. Such smart content extraction and paragraph boundary determination would require an algorithm that is beyond the focus of this project. Since the scope of this project is semantic extraction, correctly detecting paragraph boundaries and removing all the extraneous noise in free text is left as a future exercise. Therefore, the preprocessing stage was done manually to extract only the main topic content from each document and discard the rest of the file.

3.2.4 Parser and Tokenizer

The parser starts the actual examination of each document and scans it for keywords, essentially tokenizing the document and recording the term frequency of each keyword. Any stop words encountered by it will be discarded. Stop words are words that do not contribute to the core sentence topic. They are auxiliary words that help to connect phrases and combine ideas but do not have core meanings by themselves. Such words are the prepositions, articles, and modifiers. As it builds the keyword list, decapitalization and stemming is simultaneously employed to find the unique root form of each word so that there is one record that represent the same term. Removing capitalization is straight forward as we don't want to store something like "Spartans," "Spartans," or "SPARTANS" as three different terms. Stemming is the process of finding the unique root of a word. It is necessary for English text processing as many forms of the same word occur due to conjugation, present and past tenses, and plurality.

The process of stemming decomposes the word to its root base so that we have just one way to record a word. The code used is the Porter Stemming code that was downloaded from the Internet (Martin Porter, 2006) and integrated into the code base.

After the document is tokenized and term frequency recorded, it will go through a round of term frequency pruning. Only frequent keywords with at least minimum, threshold frequency will be kept and the rest discarded. The output of this stage is a list of frequent keywords.

3.3 Candidate Generator and Pruning

This stage starts the core of the algorithm. The algorithm's uniqueness is in finding co-occurring keywords that appear close to each other in a paragraph and record their document frequencies. Note in the diagram, this stage is drawn as part of a cycle of three processes. The cycle illustrates we will make multiple passes through it to generate the candidates. It is essentially a bread-first search algorithm that finds all 1-keyword candidates, then 2-keyword candidates, and so on. Therefore, each iteration results in a new *k*-keyword list, starting with k=1. The algorithm stops when it gets to a *k* where there is no more frequent *k*-keyword sets found or we artificially limit our algorithm run to a maximum *k*.

3.3.1 Breadth first rather than depth first

Having to do one pass per *k* seem inefficient. Indeed, it is necessary to complete the processing for one *k* before moving on to do it for *k+1*. Why not do depth-first and simultaneously produce multiple *k*-lengths candidates? From the onset, it appears to provide a more efficient generation strategy. That seems true until we look at the size of

the potential candidate space that must be generated at each stage. Using a bread-first strategy allows us to employ some routines to mitigate this cardinality problem while depth-first search will not. The bread-first strategy will allow us to discard many candidates early on but not in depth-first search. In depth-first, that decision cannot be made until the time and storage would have been expended. To better appreciate the problem, let's first let look at the exponential candidate growth problem in detail.

To find all the co-occurring keywords of length *k* within a paragraph, we need to consider all the possible *k*-length "permutations" of the words in the paragraph. "Permutations" is qualified here to be not permutations in its pure definition. We are not permuting all the words in the paragraph. Instead, we are only considering permutations that retain the relative order of the original word position. While this is a much smaller set than the full permutation, it is still an exponentially large data set. The cardinality of the candidate set is the following formula.

Let p = paragraph size, k = keyword set length

$$all\ possible\ candidates = \sum_{n=p}^{k} \left(\prod_{\mathrm{m=n}}^{\mathrm{n-k+1}} \mathrm{m} \right)$$

Remember each keyword set retains the relative order in a paragraph so this formula is a subset of *k*-permutations. Still, the growth of candidates is exponential and is illustrated by the following example. For a short 20-word paragraph,

2-keyword possible candidates
*= (20*19) + (19*18) + (17*16) + ⋯ + (3*2) + (2*1) = 2660*

3-keyword possible candidates
*= (20*19*18) + (19*18*17) + ⋯ + (4*3*2) + (3*2*1) = 35910*

Suffice it to say, at higher *k*, the *k*-keyword candidates set quickly becomes unmanageable. Luckily, to help mitigate this problem we have two tools to reduce the candidate space. The first is Apriori and the second is TF-IDF.

Recall the Apriori principle has the property where a *k*-keyword set is frequent only if all of its subsets are frequent. Turning it around with the antimonotone rule, if a *k*-keyword set is not frequent, then any of its superset cannot be frequent. Leveraging this fact, it implies if a *k*-keyword set is determined not frequent, then there is no need to consider any of its superset for frequency. The bread-first search algorithm uses this fact in each iteration to produce a list of *retained* candidates. That list becomes the only candidate prefix to consider in the next iteration. In each successive iteration, it only needs to consider those candidatete with prefixes that are part of that list handed off from the last iteration. This is why the bread-first strategy is so desirable because such a *k*-keyword set list cannot be determined until its iteration is complete. Only then can the *k+1* iteration be able to safely ignore the dead end keyword sets. The Apriori is tremendously helpful and provides a huge reduction in the candidate set space.

Now we explain what is meant by *retained* candidates in the previous paragraph. The implication is there is a test performed on each keyword set to see if it should be kept or discarded. That brings us to the second tool used to help mitigate the cardinality problem, using principles of TF-IDF. What is attractive about the formula is the principle and behavior of it but we do not use the pure formula itself. For our algorithm, we are interested in document frequency so term frequency (TF) is not so important to track. We still use it as part of the calculation, but its count is not used directly. The IDF

formula behavior is more relevant to us because we like that it can help discover frequent keyword set while at the same time will eliminate keyword sets that are too frequent, too commonplace. Instead of using the actual IDF formula we only use a simple document frequency (DF) percentage calculation for each keyword set, and then set a minimum and maximum threshold points to determine if a keyword set is to be discarded. This preserves IDF's general behavior and avoids having to calculate the log function or track term frequency.

Let keyword set $K = \{k_1, k_{1, \ldots,} k_n\}$
Let D be the set of all documents

$$DF(K, D) = \frac{\#documents\ containing\ K}{\#documents\ D}$$

By using these two tools to reduce the candidate set, we have effectively turned around an insurmountable problem into a manageable one. The actual pruning threshold (user configurable) determines how much of the candidate space has been reduced. Using an aggressive pruning threshold would give us smaller candidate space but can risk discarding the important candidates (false negatives). On the other hand, using a relaxed pruning threshold might bring back the cardinality problem if almost all of the candidates are kept (too many false positives). It is a balance that must be fine-tuned. Experience with several types of data set shows that the right threshold value depends on the data content being processed so ultimately it may depend on the type of content being processed.

3.3.2 Human Concepts

After the pruning stage at the end of each iteration, the resulting list of *k*-keyword set is the human concepts of *k*-length. This is the target unit of knowledge the algorithm set out to produce. However, note the human concepts stage is part of the iterative cycle, which means it is still susceptible pruning at a later iteration. The type of pruning employed at this stage is slightly different. Instead of examining just the DF formula, we employ the maximal simplex property, which is to say we find all the simplex that cannot be a face of another simplex. What this means is if a *k*-length human concept is determined to be frequent (meaning it forms a simplex), then any human concept formed by its subset should be discarded. Following our previous sample example of "online social network analysis," it would have progressed in the following way:

Iteration k	**Resulting k-keyword set**	**Discard**
1	online	
2	online social	online
3	online social network	online social
4	online social network analysis	online social network

Figure 3: k-keyword set evolution

While the maximal simplex property is clear in mathematics, it seems to present a problem when applied to real life text. By this rule, it precludes the concepts of "online social network" and "online social network analysis" from simultaneously being different concepts since the longer 4-keyword set would prune the shorter 3-keyword set. Certainly, there is a problem with mapping this mathematics model to real life text because the two topics in this case should be recorded as two different concepts. Therefore, we have to make an adjustment to the maximal simplex rule. We have to allow subsets of it to exist in the final simplicial complex graph. Instead of blindly

discarding the shorter concept once the longer one is formed, we add an additional document frequency comparison before discarding. In summary, we try to determine if, in fact, the shorter keyword set is another concept altogether. Here is the formula to determine if a non-maximal simplex should be retained:

Let M_k be a maximal simplex of size k

M_{k-1} be a simplex contained by M_k.

Retain M_{k-1} if ($| M_{k-1} | = 2 * | M_k |$), otherwise discard

The modification is to consider the frequency of the subset keyword set to determine if it itself is more than twice as frequent as the maximal simplex. The reasoning behind this formula is that normally a human concept is described by a unique *k*-keyword set and any of its subset counts only existed because they were temporary keyword sets formed on the way to the final *k*-keyword set. The subset counts were merely artifacts of finding the final *k*-keyword set. However, when the subset exists in significantly frequency higher than the final *k*-keyword set, then we have to conclude that the subset is itself a concept. This can be better explained by example. The following table shows the evolution of "social network analysis" formation, with the additional test of whether to discard or keep its subset keyword sets. Suppose the minimum threshold frequency is set at 50.

Iteration k	Resulting k-keyword set	Frequency	Discard or Keep
1	social	130	Discard
2	social network	105	Keep
3	social network analysis	51	Keep

Figure 4: Deciding whether to keep shorter keyword set

By Apriori, the table clearly shows the frequency of "social network analysis" is less than that of "social network" whose frequency is in turn less than that of "social." All of them meet the minimum frequency threshold so are potential candidates to keep. Now we compare adjacent iteration frequency counts using the new formula. The keyword "social" count of 130 is compared to its adjacent iteration for "social network" count of 105. While the count of "social" is more, it is not two times more, and so it is discarded. Next, the keyword set "social network" count of 105 is compared to the count of "social network analysis" of 51. 105 is more than twice 51, and therefore "social network" itself is retained as a concept along with "social network analysis."

3.3.3 Knowledge Base

After surviving this last pruning, we have the resulting human concept list and collectively this forms the semantic knowledge base. This is the final knowledge table we set out to produce and here is a sample query of it from the database in the capture below. It is from this table of information alone that stores all of the semantic knowledge learned. How well the algorithm has performed is determined by the results in this table. More true positive concepts discovered into this table means we will be able to identify real world concepts accurately. Too many false positives or false negatives would mean we will misclassify often. The process of fine-tuning the algorithm and judging improvements will often look at the results of this final table. In an ideal case, this table will only contain relevant human concepts and grow automatically as new concepts arrive on the web.

	ID	Tokens	TokenCount	Frequency	DocFrequency	Perm	TokensOrigin	FirstPos	Distance
1	8874	network social	2	203	15	0	network social	177	30
2	8889	network structur	2	59	15	0	network, structure	3865	22
3	8711	network data	2	81	13	0	network data.	8046	19
4	14738	network connect	2	52	12	0	network connected	1745	12
5	121513	network analysi social	3	41	12	0	Network analysis social	445	42
6	8669	network base	2	69	12	0	network based	1249	54
7	14869	network set	2	36	12	0	network set	561	26
8	8805	network node	2	106	12	0	networks nodes.	639	60
9	8753	network gener	2	25	11	0	networks general	335	22
10	8861	network scienc	2	22	11	0	network science.	8046	23
11	9031	node network	2	82	11	0	nodes. networks	660	58
12	5898	central network	2	59	11	0	centrality network	2068	47

Figure 5: Knowledge base

3.4 Implementation Details

The software is designed as an object oriented program and is implemented in Microsoft Visual C++ as a multithreaded process. It is largely ANSI-C++ compliant except in thread management, IO system calls, and SQL database access.

3.4.1 Pseudo code

```
Load from database knowledge learned so far
For each k starting with k=1
{
      For each file f
            Parse and tokenize file f
            keywords <= decapitalize and stem
            keywordsets <= Permute candidates(keywords)
            keywordsets <= PruneUsingDF(keywordsets)
            keywordsets <= PruneSubsets(keywordsets)
            Record human concept to database
      If no more frequent k-keyword set found, exit loop
}
```

3.4.2 Runtime parameters of paragraph size and frequency thresholds

The software is configurable through some input runtime parameters so that there is a level of control over how it behaves. This is necessary to allow fine tuning the analysis without reprogramming. The exposed parameters are:

- Term frequency min and max threshold (as percentage)
- Document frequency min and max threshold (as percentage)
- Maximum keyword set length (in word count)
- Min and max paragraph size (in word count)
- Number of threads

It is the case that certain threshold values are better suited for certain types of data. Only through many runs of the program and analyzing the results and fine tuning these values do we arrive at some balanced approach.

The paragraph size parameter is an important one. It determines how far apart words can be apart and still considered as potential concepts. By trial and error, we find a good size for this to be 30. A smaller number may miss conceptual words that span over many sentences. A number too small will end up processing short sentences, which most likely do not have concepts to capture. Therefore, we specify a minimum paragraph size and if a paragraph does not meet that minimum, it is merged together with the next paragraph. On the other hand, a larger paragraph size will capture more but also may capture more unrelated words and form false positives. It will also exponentially increase processing time. As the next section shows, the algorithm's complexity is largely driven by the paragraph size. Therefore, it is necessary to specify an upper bound the maximum paragraph size that will be analyzed so that the algorithm will finish. A good boundary seems to be from a minimum of 25 to maximum of 30 words.

It is true that this algorithm processes keyword concepts by chunks of paragraphs and it can miss those cases where a concept can span the paragraph boundary chosen. We recognize this can happen, but if a concept is a true concept that occurs in many documents, we may miss one but will capture the majority of its occurence.

The term and document frequency thresholds are obvious as their namesake. It is found there are two basic ranges for these values. When the input data set is small, 20 files for example, the threshold percentage can be achieved quite easily by just a few occurrences. For example, by just being found in 4 files in a 20 file data set, it is already 20% occurrence. However, if the entire data set was 1 million files, then demanding a threshold of 20% (which is 200,000 occurrences) may seem too excessive. That is because the criteria for determining keyword frequency are not linear as data set grows. For example, if 10,000 unrelated people blogs about a particular topic, it ought to start turning into a human concepts, regardless of whether it has passed a threshold minimum percentage. However, for our project and in dealing with processing hundreds or even thousands of files, the frequency threshold values chosen was between 0 to 95%.

3.4.3 Known words and stop words

The system allows manual input of known words or keyword sets directly into the final human concepts database table. These become overrides and let the algorithm know that they are real human concepts and need no further analysis. These entries become permanent and will never be pruned during processing. This is a way to make the system more accurate and identify known concepts to help jumpstart the system.

On the other hand, the system maintains a table of stop words. These are words that are to be removed immediately when encountered. Stop words are defined to be things like conjunction, preposition, articles, and abbreviations. The list is exhaustive and is maintained in a special database table. There are already hundreds defined and user can

add more. If the word is in this table, it will be discarded immediately during parsing time.

3.4.4 Algorithm Analysis

The algorithm runtime is dominated by the maximum desired keyword set length (maximum *k*) and the number of potential candidates. In turn, the size of potential candidates is determined by the size of an average paragraph. Therefore it is useful to express the big-O analysis in terms of paragraph size and maximum *k*.

Recall that in processing one *k*-length keyword set in one paragraph, the number of candidates generated was the sum of product formula. To simplify it, it can be bounded by expressing it in terms of the average paragraph size.

Let p = paragraph size, k = keyword set length

$$all\ candidates\ in\ 1\ paragraph, 1-keyword\ length = \sum_{n=p}^{k}\left(\prod_{\mathrm{m=n}}^{\mathrm{n-k+1}}\mathrm{m}\right)$$

Then this formula is bounded by $(p-k)p^k < p^{k+1}$. This must be repeated for each *k* up to the maximum *k* desired, call it K. So we have total

$$all\ candidates\ in\ 1\ paragraph, up\ to\ K\ keyword\ length < Kp^{K+1}$$

Let W = count of words in all documents processed, and let a constant C = $\left(\frac{W}{p}\right)K$, then the total candidates to process is

$$all\ candidates\ in\ all\ documents = \left(\frac{W}{p}\right)Kp^{K+1} < Cp^{K+1}$$

With Cp^{K+1}, the resulting big-O complexity is **O (p^K).** That means the runtime is determined by two factors, the maximum keyword length and the paragraph size. This exponential complexity looks to run horribly infinite. If coded without an optimization and pruning strategy, the execution runtime would never end in a real world system. Luckily, as already explained, Apriori allowed us to safely ignore a sizable number of dead end candidates in the early *k*-iterations. The algorithm can be viewed as front loaded since it must fully process the initial iterations of k=1 and most of k=2 before it has the ability to make decisions to ignore larger length candidates. As it progresses to the higher *k* iterations, most of the permutations would have been eliminated. Therefore, in read world situations, the behavior of the algorithm is probably closer to **O (p^3)**.

3.4.5 Code

The code was compiled as a true 64-bit executable and is designed to spawn multiple threads in a multiple core system to speed up the process. The threading happens in the parsing and tokenizing stage. Then all the candidates and final human concepts determination are synchronized among all threads. A single database thread writes the results to the database. Here are the specs of the computer system and retail software used to compile and run the system:

- Core i7 dual core CPU
- Windows 7 64-bit
- Visual Studio 2010
- SQL Server 2008 R2

Running the algorithm in a modern dual or quad core CPU with hyper-threading makes a big difference. Compiling in 64-bit as opposed to 32-bit also improved the

performance almost 100%. The number of threads to spawn is configurable so it can take advantage of all cores available in the system. For example, a Dell PowerEdge R620 server with a dual Xeon six core CPU with hyperthreading has a total of 2 * 6 * 2 = 24 hardware threads. The program can utilize all of the 24 threads to process the files.

3.4.6 Bottlenecks

There are two major bottlenecks to the implementation, file IO and RAM memory. The processing requires a lot of file reads and database reads, writes, and deletes. Although keywords sets are not written to database until they are completed per iteration, all records are subject to be further analyzed and pruned even after being recorded in the database. All of this IO takes a toll on the system and tremendously depends on disk drive performance. The other limitation to the algorithm is RAM. During processing, all temporary objects and potential candidates are managed in RAM until they are relatively permanent. The potential candidates, despite Apriori, can still be enormous, so the program will require enough RAM in the system to operate properly. The larger the data set the more physical RAM are required to run at acceptable speed. Reliance on virtual memory and excessive paging should be avoided.

3.4.7 Detailed description of each C++ classes

This following describes the C++ modules of the Semantic Search project.

Filename	**Content**
Sutoken.cpp	main() function entry point to program • starts running the algorithm • initializes variables

	• parses the command line • open and close database connection • manages the child threads.
Config.cpp/.h	Defines class CConfig for command line processing
PorterStemmer.cpp/.h	Stemmer functions • Contains the stemmer function to find the root of a word. This was code obtained from the Porter stemming algorithm site http://www.tartarus.org/~martin/PorterStemmer. It was modified to interface with the project.
TokenizerUtils.cpp/.h	Defines various utility functions for initialization. It also contains the CBenchmark class, which measures execution time of the entire analysis, useful information for rating how well the algorithm runs.
URLPage.cpp/.h	Defines class CURLPage for storing a webpage address and its attributes.
DataStorage.cpp/.h	Defines class CDataStorage, which accesses the SQL Server database. To support another DBMS such as Oracle, this is the class to replace.
Token.cpp/.h	Defines class CToken which is used to represent a token/keyword or a human concept.
FreqList.cpp/.h	Defines class CFreqList, which holds a list of CToken objects; this would be a list of token or human concept objects. This class will use CDataStorage class to save to the database.
ThreadManager.cpp/.h	Defines class CThreadManager. This is the heart of the application and has all the core functionality. Its responsibilities: • database • load existing data from database • save analyzed data to database • facilitates accumulating knowledge from the past runs • manage start and stop child threads • defines the child thread body itself • defines the CSLock class for thread synchronization • parse text files • analyze permutations of co-occurring words (maintains relative order)

	• data pruning • determine stop words • detect US address (currently not used) • output to text files (currently not used since it writes to DB)

4 Analyses and Results

4.1 Data set IEEE International Conference on Social Computing

This is the training set of 21 file that was used to develop the algorithm. The content of these files are the topics coming from the 2006 conference. The main topic is on social computing. Running the files through the system, we would expect it to find all the concepts related to social computing. If we take the top 50 concepts found, this is what it found as human concepts:

	Human concepts (stemmed)	Keyword count	Freq	Doc Freq	Original Text	Distance
1	network social	2	207	16	networks Social	12
2	network structur	2	71	15	networks structural	42
3	network connect	2	66	13	networks connectivity,	52
4	network data	2	88	13	network data.	19
5	network analysi social	3	46	12	Network analysis social	42
6	network set	2	29	12	network. set	46
7	node network	2	76	12	nodes. networks	62
8	et al	2	67	12	et al.	3
9	network type	2	22	11	Network types	23
10	network result	2	31	11	networks, results	37
11	measur number	2	20	11	Measurements number	15
12	network commun	2	47	11	networks, communities.	36
13	network base	2	69	11	Network based	43
14	network comput	2	21	10	network. Computing	14
15	network edg	2	28	10	Networks edge	18
16	network group	2	47	10	networks. groups	14

17	network method	2	29	10	network method	12
18	network link	2	32	10	network links.	60
19	level differ	2	18	10	level different	36
20	network scienc	2	23	10	network science.	41
21	network relat	2	70	10	networks, related	56
22	node edg	2	27	10	nodes edge	12
23	identifi network	2	41	10	identify networks	15
24	futur work	2	18	9	future worked	39
25	group network	2	36	9	group network	50
26	figur network	2	34	9	Figure networks.	31
27	analyz network	2	26	9	analyzed Network	15
28	cluster network	2	39	9	clusters network	50
29	construct network	2	33	9	construct network	28
30	network studi	2	18	9	networks studied	34
31	network technolog	2	18	9	Networks Technology	13
32	paper network	2	32	9	paper network.	69
33	pp vol	2	28	9	pp. Vol.	50
34	proceed confer	2	20	9	Proceedings Conference	8
35	larg number	2	15	9	large number	2
36	network model	2	48	9	networks, model	46
37	network number	2	29	9	network number	23
38	network introduct	2	17	9	networks, INTRODUCTION	12
39	network gener	2	27	9	networks general	22
40	network includ	2	22	9	networks. include	16
41	network individu	2	32	9	network individual.	26
42	network cluster	2	47	9	networks cluster	38
43	user data	2	68	9	users data	43
44	small group	2	19	8	small groups	4
45	solv problem	2	18	8	solve problems	4
46	small network	2	14	8	small networks	4
47	vol social	2	14	8	vol. social	38
48	user research	2	22	8	users research	26
49	web network	2	18	8	Web network	46

Figure 6: Social network

As expected, what turned up as the primary topic was that of social networking. At the top, the keyword sets have something to do with networking. Just by looking at the

results in the database, it is apparent that the topics being discussed in the papers were networking. While the topic of the conference is social networking, which did turn up at the top, the majority of the concepts picked up seem to be just networking topics in general such as structure, connectivity, clusters, etc. Indeed, through reading the papers, those are the topics being addressed as it discusses problems and solutions to social networking.

4.2 Anomalies

As discussed, the methodology preserves the relative order of the keywords in the original text. A concept should appear as a unique sequence and generally, the scrambled sequence will not occur as frequent as the natural sequence. So why is it "network social" and not "social network" that turned up at the top? Where is "social network," it is not even the top 50 returned? The short answer is that this exercise operated on a rather limited 21 set of files and so it was not large enough to filter out anomalies such as these. Still, we're interested in how this could have happened. Here is an explanation. The concept "social network" was captured by the software in the form of 3-keyword sets. As indicated in the following table of results, it shows the "social network" counts were divided and consequently were less that the number of "network social" found.

Human concepts (stemmed)	Keyword count	Freq	Doc Freq	Original Text
pp social network	3	13	7	pp. social networks
repres social network	3	29	6	Representing social networks,
servic social network	3	24	6	Service social network
web social network	3	13	6	Web, social networks
complet social network	3	9	6	complete Social networks.
interact social network	3	19	6	interactions Social Networking

complex social network	3	6	5	complex social network

Figure 7: Social network topic

Note that it found "pp. social networks" as a concept. This illustrates the importance of preprocessing where noise such as these should be eliminated. This was discovered by the software due to the bibliography construct:

> Gruhl, D., Guha, R., Liben-Nowell, D., and Tomkins, A. (2004) Information diffusion through blogspace, Proceedings of the 13th international conference on World Wide Web, New York, NY, USA, pp.491-501
>
> Yoo, S., Yang, Y., Lin, F., and Moon, I. (2009) Mining social networks for personalized email prioritization, Proceedings of the Knowledge Discovery and Datamining 2009, Paris, France, pp. 967-976

In this small data set of 21 files, the bibliography represents a sizeable percentage of the data and apparently this combination appeared so often that it was identified as a human concept.

It still begs the question how did "network social" found its way to the top? Analyzing the original text, here are a few sentences that illustrate how that happened.

> This work demonstrates the importance of inter-disciplinary work, particularly between the content and the network analyses, in the social web study.
>
> In these communities, the social patterns that people form are often organized without explicit leaders, chains of command, or fixed communication networks. Examples of such spontaneously emerging social groups include fans at a sport stadium …
>
> In research on how OSS developers collaborate, research considers the OSS movement as a self-organizing system

> and a collaborative social network [11]. Social Network Analysis was used for analyzing …
>
> Our social network is based on the entire Wikipedia edit history, and therefore is a summary of all recorded author interactions. This social network can be used to assess …

By nature of the algorithm to analyze closeness of words without regards to any other factors such as punctuations and sentence structure, such abnormality will occur. In a small data set, these occurrences can overtake the real natural order. Also note in the third excerpt, the words "social network" appeared twice very close to each other, making "network social" become a valid candidate. We propose that in real world situations when millions of files are ingested, this type of error will be masked by the real concept sequence.

Another interesting point is that beyond 3 tokens, there are only a few interesting things found. In the table below of keyword sets greater than 3, most of the concepts identified were due to noise, such as the words "vol." and "pp." Overall, there does not appear to be many real concepts identified beyond 3 keywords. This was also the result after subsequent runs with other data sets. Perhaps this reflects reality as most concepts can be described in 2-3 words.

Human concepts (stemmed)	Keyword count	Freq	Doc Freq	Original Text
newman commun structur network	4	10	5	Newman community structure network
applic cambridg univers press	4	7	5	Applications, Cambridge University Press,
dept comput scienc univers	4	12	5	Dept. Computer Science University
network cambridg univers press	4	10	5	Network Cambridge University Press,
network method applic cambridg press	5	9	4	Network Methods Applications,

				Cambridge Press,
network method applic cambridg univers press	6	9	4	Network Methods Applications, Cambridge University Press,
network method applic press	4	6	4	Network Methods Applications, Press,
network method applic univers press	5	6	4	Network Methods Applications, University Press,
network method cambridg press	4	9	4	Network Methods Cambridge Press,
network method cambridg univers press	5	9	4	Network Methods Cambridge University Press,
network method univers press	4	6	4	Network Methods University Press,
network nation academi scienc vol	5	4	4	networks, National Academy Sciences, vol.
network nation academi vol	4	4	4	networks, National Academy vol.
network nation scienc vol	4	4	4	networks, National Sciences, vol.
network natl acad sci	4	5	4	networks Natl Acad Sci
network scienc vol pp	4	5	4	networks, Sciences, vol. pp.

Figure 8: Human concepts longer than 3

4.3 Google Data

In order to study the behavior of the algorithm with more data, Google search was used to collect some current real world web pages to run the algorithm against. The data collection was done by using Google to search a topic and retrieving the top 40-50 links. The four topics were searched using the following search strings:

- mobile computing
- sound investment
- greenhouse effect
- court ruling

In total, there were over 200 files of Google results. All of the files were manually preprocessed to remove the noise (such as navigation menus, advertisements, HTML tags, etc) and leave only the relevant textual content.

These groups of files become the controlled data input. Since the main subject matter of each group is well-known, the test is to see if the algorithm is able to learn and

discover the expected human concepts from each group and classifies them as the top most frequent.

4.3.1 Search for "mobile computing"

The top 40 files were retrieved from the search result. This was then fed to the algorithm to see if it will discover mobile computing as a top human concept. The results are in the table below. As expected, the top results did turn up mobile computing as the number one identified human concept, among many related mobile computing concepts. There are a few undesirable concepts that turned up such as "mobile developed," "computing new," and "mobile new."

Human concepts (stemmed)	Keyword count	Freq	Doc Freq	Original Text
mobil comput	2	212	29	mobile computing
comput mobil	2	62	22	Computing mobile
mobil devic	2	87	17	Mobile devices
mobil new	2	25	13	Mobile new
mobil develop	2	27	12	Mobile developed
mobil technolog	2	41	12	mobile technology
tablet devic	2	17	12	Tablets devices
new comput	2	16	11	new computing
comput new	2	15	11	Computing new
develop comput	2	15	11	developed computing

Figure 9: Results for mobile computing

4.3.2 Search for "sound investment"

The top 62 files were retrieved from the search result. This was then fed to the algorithm to see if it will discover sound investment as a top human concept. This was a real surprise. At the onset, the search string was intended to find links related to advice

on financial investment and making good money decisions. Unexpectedly, what turned up from Google were both musical entertainment services and financial links, with the former topic being an overwhelming majority of the links. This is the reason why a little more files were retrieved for this search string since it actually returned two different subject matters.

As is normal for Google, the links were interleaved with no regard to grouping related ones together. Not surprisingly, our software was able to capture both subject matters as its top human concepts.

However, this group of data files revealed a potential problem in the algorithm. The words "sound investment" was discovered as a top rank concept but the phrase itself is ambiguous in the human language. Ideally, "sound investment" should not be identified as a human concept but rather its superset should be, such as "sound investment music" and "sound investment money." As it is, anything classified into this group is not differentiated by the software. We recognize this can be a problem but it can also be attributed to not having enough data for processing. The superset versions of the concept probably never materialized because there was not enough document frequency. It is an area that needs to be looked at for further study.

Human concepts (stemmed)	Keyword count	Freq	Doc Freq	Original Text
sound invest	2	155	41	Sound investment,
sound servic	2	36	18	Sound service
sound busi	2	24	16	Sound Business
sound compani	2	28	16	Sound Company
fund invest	2	16	16	funds invested
monei invest	2	24	16	money invested

great invest	2	18	14	great investment
financi invest	2	24	14	financial invest
compani invest	2	20	14	Company investments
creat invest	2	16	14	create invests
sound entertain	2	16	14	Sound entertainment
sound music	2	18	14	Sound musical
music invest	2	20	14	musical Investment
sound year	2	20	14	Sound years
stock invest	2	14	14	stocks investment
stock market	2	20	14	stocks markets

Figure 10: Results for sound investment

4.3.3 Search for "greenhouse effect"

A data set of files was collected by typing "greenhouse effect" in Google and retrieving the top 46 files returned from it. It was manually preprocessed and fed to the algorithm and the results are below. This example illustrates that whatever was typed as a search string does not necessarily end up being the most frequent keyword occurrence. The results suggest to the user that the software was able to correlate greenhouse effort with all the related hot topics such as "climate change" and "gas emissions."

Human concepts (stemmed)	Keyword count	Freq	Doc Freq	Original Text
climat chang	2	117	22	climate change
ga emiss	2	128	22	Gas Emissions
global warm	2	146	21	global warm.
effect greenhous	2	88	21	effect greenhouse
gase greenhous	2	77	19	gases greenhouse
heat atmospher	2	35	18	heat atmosphere.
effect atmospher	2	41	17	effect atmosphere
ga greenhous	2	41	16	Gas Greenhouse
dioxid atmospher	2	36	16	dioxide, atmosphere.
carbon atmospher	2	48	16	carbon atmosphere.
carbon dioxid atmospher	3	39	16	carbon dioxide. atmosphere,
carbon emiss	2	54	15	carbon emissions.

Figure 11: Results for greenhouse effect

4.3.4 Search for “court ruling”

A data set of files was collected by typing “court ruling” in Google and retrieving the top 52 files returned from it. It was manually preprocessed and fed to the algorithm and here is the result. There was no surprise from this run and it validates that the algorithm is able to capture the relevant concepts.

Human concepts (stemmed)	Keyword count	Freq	Doc Freq	Original Text
suprem court rule	3	62	33	Supreme Court ruling
suprem rule	2	68	33	Supreme ruled
suprem decis	2	53	27	Supreme decision
suprem law	2	44	25	Supreme law,
suprem court decis	3	41	25	Supreme Court decision
law court	2	57	25	law, Court
said rule	2	44	23	said. ruling
suprem court law	3	42	22	Supreme Court law
said court	2	40	21	said court
law said	2	36	21	law said
care law	2	71	21	care law,
health law	2	82	21	health law,
law rule	2	34	20	law ruling
mitt romnei	2	28	20	Mitt Romney
state law	2	60	20	state law
said law	2	38	19	said. law.
presid obama	2	34	19	President Obama
health care law	3	71	18	health care law
health insur	2	65	17	health insurance

Figure 12: Results for court ruling

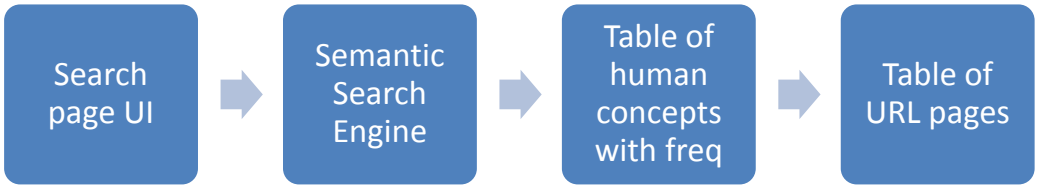
5 Applications

With the knowledge base built, there are many important applications for which this technology can be utilized to make them semantic aware. In this paper, we illustrate two applications: semantic search and examiner utility.

5.1 Semantic Search

By using the knowledge base of human concepts built, we build a simple search engine to demonstrate how a semantically aware search would work. It will feature ability to divide ambiguous search string by subject matter and it will use document frequency as a rudimentary way to rank page results.

The knowledge base built for this project contains the three essential things needed for the semantic search engine: human concepts, document frequency, and human concept-web address relationship. The idea is to implement a 2-tier search whenever there is ambiguity in the search string. The following block diagram illustrates the basic data flow.



When asked to search a topic, the search engine will look for the closest match against the table of human concepts first. If there is an exact match or there is a single partial match then it can proceed to retrieve all of the URL pages associated with the concept. If there are more than one partial match to the concepts table, then it will present the list of concepts to user and prompt for a selection of the intended one. After

this selection, the software will be able to return only the topic links desired. This is a much more friendly way of interacting with the user than the current way of unorganized results.

5.2 Examiner

This is a utility that is able to scan a previously unseen document and able to tell what its main topic is. It can do this by using the same knowledge base built for this project. The way it does this is by relying on what is already learned in the system. It regards it as the "goldmaster" of knowledge base. When given a file to analyze, it will scan through it and tokenize it to the candidate concepts list, very similar to the learning mode. The difference is that in this case, it won't be discovering new concepts or saving back anything new to the database. It will query the database to see if the concepts it found in the new file exists in the database. The concepts that are found in both the new file and exist in the database will point to the main topic or topics of the document. With a sizeable knowledge base built, such a utility will be able to dynamically classify web content as they come in. Obviously, the larger the knowledge base, the better it will be able to reveal the main topics of more documents. On the other hand, it will be limited by the topics that it has already seen. Therefore, for example, if it has never learned about mobile computing and a file about mobile computing is fed to it to analyze, it would report unknown subject matter.

6 Encouraging Results

The test results and the sample applications show the technology to be very encouraging. In the test runs against the Google search data, the software has shown it is able to capture the main subject matter using the proposed algorithm without human guidance or special data tagging. The resulting records in the database together form the knowledge base of the semantic engine. As it is a cumulative data collection engine, the more data it ingests the more knowledge it gains.

As noted, there are some undesirable results that turn up. Some errors are trivial such as those that indicate inadequate preprocessing or that there was not enough data files to build up the real human concept in the correct word sequence. Other problems are more serious when words with dual meanings are still discovered a single human concept. In learning software such as this, some error is to be expected. It is enough to be right most of the time. It is still suggested that most of these problems are due to this exercise being a small test run and not having enough data samples.

The important conclusion we set out to demonstrate how the algorithm can learn human concepts on its own and be able to classify web content. We have achieved that. We demonstrated it is able automatically parse and identify the key human concepts by processing existing web pages that don't have any special semantic formatting. We have shown this algorithm can run in finite time and able to return favorable results. The average speed of processing was at about 5000 words/second. Overall, we have demonstrated the technology has a good start, though with some artifacts, to take us further into more studies and eventually to solving the semantic awareness problem.

7 References

Ethem Alpaydin, Introduction to Machine Learning 2nd edition, Massachusetts Institute of Technology, 2010.

Google Search, http://www.google.com, 2012.

Info growth rate, http://www.emc.com/leadership/programs/digital-universe.htm

Martin Porter, The Porter Stemming Algorithm, http://tartarus.org/martin/PorterStemmer/, 2006.

RDF Working Group, Resource Description Framework (RDF), http://www.w3.org/RDF/, February 2004.

Roger E. Bohn and James E. Short, How much information, http://hmi.ucsd.edu/howmuchinfo_research_report_consum.php, January 2010.

Tsau Young Lin and Jean-David Hsu: Knowledge Based Search Engine: Granular Computing on the Web. Proceedings of the Web Intelligence Conference (2008) 9-18.

Tsau Young Lin and I-Jen Chiang, A simplicial complex, a hypergraph,structure in the latent semantic space of document clustering, Int. J. Approx. Reasoning 40(1-2): 55-80 (2005).