

# ACE: Improving Search Engines via Automatic Concept Extraction

Paul M. Ramirez and Chris A. Mattmann  
Computer Science Department  
University of Southern California  
Los Angeles, CA 90098-0781 USA  
{pmramire,mattmann}@usc.edu

## Abstract

*The proliferation of the internet has caused the process of browsing and searching for information to become extremely cumbersome. While many search engines provide reasonable information, they still fall short by overwhelming users with a multitude of often irrelevant results. This problem has several causes but most notably is the inability for the user to be able to convey the context of her search. Unfortunately, search engines must assume a general context when looking for matching pages, causing users to visit each page in the result list to ultimately find or not find their desired result. We believe that the necessity of visiting each page could be removed if the concepts, i.e. over-arching ideas of the underlying page, could be revealed to the end user. This would require mining the concepts from each referenced page. It is our contention that this could be done automatically, rather than relying on the current convention of mandating that the searcher extract these concepts manually through examination of result links. This ability to mine concepts would not only be useful to finding the appropriate result but in further identifying relevant pages. In this paper, we present the Automatic Concept Extraction (ACE) algorithm, which can aid users performing searches using search engines. We discuss ACE both theoretically, and in the context of a Graphical User Interface and implementation which we have constructed in java to aid in qualitatively evaluating our algorithm. ACE is found to perform at least as well or better than 4 other related algorithms which we survey in the literature.*

## 1. Introduction

Today finding information on the web is an arduous task that will only get worse with the exponential increase of content [1]. To deal with this issue, *search engines*, web sites providing users with the ability to query for web pages of interest, have become the utility of choice. However, a simple search against any given engine can yield thousands of results. This presents a huge problem as one could spend a majority of her time just paging through the results. More often than not, one must visit the referenced page to see if the

result is actually what she was looking for. This can be attributed to two important facts. On the one hand, the result page's *link metadata*, i.e. the description of the page returned by the search engine, is not very descriptive. On the other hand, the result page itself is not guaranteed to be what the user is looking for.

Once a page is found that matches the user expectations, the search engine moves out of the picture and it is up to the user to continue mining the needed information. This can be a cumbersome task as pages on the web have a plethora of links and an abundance of content. Having to filter through thousands of references is not a very rewarding search process considering that there is no way to be certain that the search will be *complete*, or always yield the web page the user was originally interested in. These problems must be addressed in order to increase the effectiveness of search engines.

We believe that the aforementioned problems could be addressed if the number of links the user had to visit in order to find the desired web page could be reduced. First, we define the following term which will be used throughout the rest of the paper and in our assertion below.

**Definition:** A *concept* is one overarching idea or topic present in a web page.

Put simply, our assertion is: *if the concepts of underlying web page results being searched can be presented to the user automatically then the amount of links the user will need to visit to find her desired result will be lessened.*

Essentially this means automatically discovering the set of concepts that describe a web page. This paper describes and evaluates a method for automatically extracting the concepts from web pages returned via heterogeneous search engines including *Google*, *MSN Search*, *Yahoo Search*, *Altavista Search* and *Ask Jeeves Search*. Along with regular concepts, our method also extracts complex concepts. For instance, it not only extracts single string concepts such as "software" but it also extracts multi-string concepts such as "software architecture" by exploiting the spatial locality of concepts discovered from a web page. Our method for automatic concept extraction is dubbed *ACE*.

The rest of this paper is organized as follows. Section 2 presents the motivation for automatically extracting concepts from web pages. Section 3 describes the ACE algorithm in detail. Section 4 presents our implementation of ACE and highlights the development of a GUI front end to test and

evaluate ACE in both experiments and practice. Section 5 surveys related work in automatically extracting concepts from web pages and improving search engines. Finally, in Section 6, we evaluate our project and conclude the paper.

## 2. Motivation

Search engines index an enormous amount of pages (Table. 1). Although this is often used as a selling point it could be considered a drawback due to a simple fact: as the amount of pages indexed increases, the size of the result list returned from a search engine increases. The question becomes, does the amount of pages indexed, freshness of results, unique hits, number of dead links, or overlap of links measure the effectiveness of a search engine? Effectiveness could also be measured by *how easily one can find the results she was looking for* or *how accurately the results matched the user's expectations*. While both these measures are very subjective they would seemingly provide the user with greater satisfaction. Indeed, there is much room for improvement in conventional search engine process (shown in Figure 1 using a UML activity diagram).

	Estimate (millions)	Claim (millions)
Google	3,033	3,083
AlltheWeb	2,106	2,112
AltaVista	1,689	1,000
WiseNut	1,453	1,500
Hotbot	1,147	3,000
MSN Search	1,018	3,000
Teoma	1,015	500
NLResearch	733	125
Gigablast	275	150

Table 1. Amount of Pages indexed by search engines<sup>1</sup>

There are several interesting facets in this process that potentially could improve the effectiveness of a search engine. First, in the Search Engine Page state there is very little information given to the search engine to be able to discriminate against. The user only provides a set of keywords which must be taken in a general context (i.e. the context in which the term(s) are most frequently used). This can yield results which may be irrelevant and adversely affect the amount of time spent in the Filter through Results state. In addition, the whole scenario is seemingly bounded by the time spent in the first two aforementioned states. There are a few reasons why this is true. First the only metadata associated with each result from most search engines is the title, a brief excerpt of text (e.g. the page's description), and a link to the returned web page. While this sometimes is enough to make an educated guess that the result is indeed the correct page the user was looking for, it still often necessitates visiting the link. Second, the time it takes to visit a page to decide whether or not it provides

useful information can vary greatly depending on the design of the site, amount of content, and location of content on the linked to page.

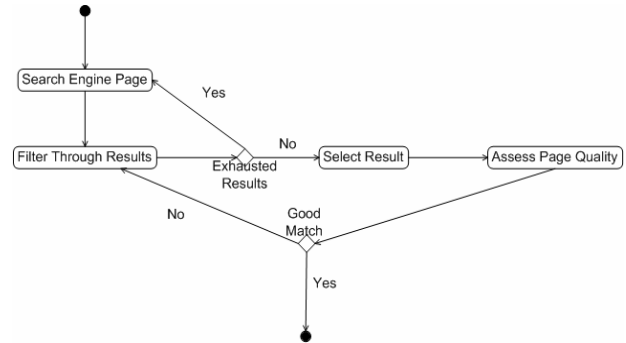


Figure 1. Search Engine Process

It is in these areas that we strive to enhance the user's experience. First, the initial time to find a matching result can be reduced by providing an accurate picture of the concepts presented in the results. This will allow the user to make an upfront discrimination and reduce the number of pages she will have to visit. If she still must visit a page then the approach we have taken will allow her to gain a better understanding of the page by visiting its referenced pages and making a quick assessment of their concepts. Lastly, feedback could be provided by the user by specifying which concepts she is interested in, thereby automatically discriminating ahead of time against the returned pages.

## 3. Automatic Concept Extraction

The problem that we are trying to solve is that of *Automatic Concept Extraction (ACE)* from web pages. It is stated formally as:

**Given:** a web page  $D$  treated as a set of ordered string tokens  $T = \{t_1, t_2, \dots, t_n\}$   
**Do:** Construct a software facility labeled  $CM$  to extract out a set of concepts  $C = \{c_1, c_2, \dots, c_n\}$  from  $D$ .

Initially we set out to formulate a method to generate  $T$ . In our algorithm,  $T$  is *extracted* from a web page which is returned by one of the 5 search engines (recall section 1) which we pose the user's initial keyword query to. Since search engines return back different content types (such as web pages, PDF files, Word Documents, etc.), we wanted to have the flexibility to define different extractors and associate them with content types. For simplicity and page limit constraints, we consider only the extractor which extracts tokens from HTML pages for now.

The HTML extractor begins by first stripping off the HTML tags from a web page, along with browser-specific HTML elements such as *style* and *javascript* tags. After the HTML tags are removed we next consider the issue of *stop words*. Stop Words are commonly used words in English sentences which convey no additional meaning other than the grammatical context they appear in. Common stop words include "the", "and" and "but". After stripping the HTML

<sup>1</sup> Taken from <http://www.searchengineshowdown.com>

tags and stop words, the HTML Extractor removes additional punctuation that we do not consider to be a potential concept. This punctuation includes symbols such as “@”, “|” and so on. At this point, we are ready to tokenize the web page into a set of tokens. Tokenization is a well known process of splitting a large string into a set of smaller strings based upon some *delimiter*. We use the *blank space* character as the delimiter by which to tokenize the web page after HTML tag stripping, stop word removal and extra punctuation removal. In summary, our HTML extractor performs the following process on D to generate the ordered string of tokens T:

**Given:**

1. a web page D
2. a set of punctuation characters  $P_c = \{p_1, p_2, \dots, p_n\}$
3. a set of stop words  $SWS = \{sw_1, sw_2, \dots, sw_n\}$

**Do:**

1. Define a function *strip\_html* which takes a document D and removes any HTML tags (e.g. begin tags and end tags), as well as browser-specific HTML tags (e.g. *style*, *javascript* and so on) from D producing D'
2. Define a function *remove\_stop\_words* which takes a document D and a set of stop words SWS and removes each  $sw \in SWS$  from D, generating a new document D', and returns the new document D'
3. Define a function *remove\_punctuation* which takes a document D and a set of punctuation characters  $P_c$  and removes each occurrence of  $p \in P_c$  from D, generating D', and completes by returning D'.
4. Define a function *tokenize* which takes a document D and a delimiter character c, and then splits D into smaller strings wherever c appears in D. The final list of smaller strings after applying the function is returned to the caller.
5. Perform the following:  
 $D := strip\_html(D)$   
 $D := remove\_stop\_words(D)$   
 $D := remove\_punctuation(D)$   
  
 $T := tokenize(D)$

### 3.1 Term Frequency

We began by investigating the *term frequency* algorithm [2, 3] that measures the amount of times that a term appears in a set of one or more documents. In our case, we have limited the term frequency calculation to a single document in order to solicit feedback from the user. The term frequency algorithm was used to assign a score to each term, and then accept a term t as a concept if it is greater than a threshold which we refer to as  $\theta$ . The set of terms that are above the threshold  $\theta$  becomes our candidate set of concepts C.

$$(1) TF(t_i) = \frac{Occurrence(t_i)}{\text{Max}\{Occurrence(t_1), Occurrence(t_2), \dots, Occurrence(t_n)\}}$$

Initially, the set of concepts C is initialized to the null set  $\{\}$ . Each term  $t \in T$  is then run through the TF function, and if  $TF(t) > \theta$ , t is unioned with C, e.g.  $C = C \cup t$ .

Some immediate observations that we made regarding term frequency was that it generally favors frequently occurring terms as opposed to other constraints on a web page. In particular, the term frequency score does not take into account the *stylistic constraints* that an HTML designer naturally imposes on a web page. For example, the HTML tags around text place an *implicit emphasis* on certain text on a web page that begs the consideration of the text as *candidate concepts*. Below, we detail a scoring method that takes into account such stylistic constraints.

### 3.2 HTML-based Emphasis

To determine stylistic constraints that an HTML designer placed on a web page, we need to modify our HTML extractor described above. We modify the *strip\_html* function defined above as follows:

1. Define a function *strip\_html* which takes a document D and removes only browser-specific HTML tags (e.g. *style*, *javascript* and so on) from D producing D'

In this manner, we can save the HTML-imposed stylistic emphasis that a designer explicitly uses on her web page. Given the new *strip\_html* function described above, the extraction process continues as normal. The token set T is then examined using the following algorithm.

**Given:**

1. A table,  $T_w$ , containing tuples of the form  $\{TAG, w\}$ , where TAG  $\in$  HTML\_TAGS, the set of all valid HTML tags considered (e.g. A, B, P, FONT and so on) and where w is the assigned *weight* of the particular html tag TAG e.g.  $\{B, 1.5\}$  or  $\{P, 0.5\}$
  2. A set of ordered tokens T which still contain HTML tags around each token  $t \in T$ .
- Do:**
1. Define a function *get\_tag*(token) which returns any HTML tag that is associated with a particular token t. Note that *get\_tag* may return 0 or more tags.
  2. Define a function *lookup*(table, tag(s)) which returns the weight (if any) in  $T_w$  for the HTML TAG “tag” or set of HTML TAGS “tags”. Note in the case of multiple tags, the function returns the summed weights (if any) of each tag passed in.
  3. For each  $t_i \in T$ , create a variable  $score_i$  initialized to 0
  4. For each  $t_i \in T$ ,  $score_i := score_i + lookup(T_w, get\_tag(t_i))$

Once the HTML scorer has scored each token t, then each t is examined in the same fashion as with the Term Frequency scoring algorithm. Initially a set of concepts C is initialized to the null set  $\{\}$ . Then, for each  $t \in T$ , if  $HTML\_Scorer(t) >$  a particular threshold  $\theta_1$ , it is accepted into C, e.g.  $C = C \cup t$ .

### 3.3 Exploiting area of locality around a concept

To review, our original problem with using a scoring algorithm such as term frequency to generate concepts was that it did not take into account the stylistic constraints on a web page that a designer may use to emphasize text. To remedy this, we constructed an HTML scoring algorithm to determine a set of possible concepts on a web page. One observation we made though at this point was that both scorers did not take into account the fact that concepts may be *multi-string* concepts, rather than the *single string* concepts that were present in the token set generated by the HTML extractor.

To attack this problem, in each scoring algorithm we decided to exploit the local area around each potential concept  $c \in C$  (referred to from here on as *candidate concept*) within a certain locality of tokens in the original token set. From experimentation we have observed that the tokens around the selected concepts generally could be combined, along with the concept itself to convey additional meaning. For example, if we selected the concept “software” from the web page in Figure 2, then assuming we examined the tokens around each occurrence of the token “software” in  $T$ , up to some bounding number e.g. 2 (which we will refer to from here on as  $A_L$ ), we could also potentially add the candidate concepts

```
<html><body>
  <h3>Chris's web site</h3>
  <p>Chris Mattmann is a research
assistant in the Software Architecture
Group at USC's Center for Software
Engineering.</p></body></html>
```

Figure 2. Exploiting area of locality to determine multi-string concepts

“software architecture” and “the software”, along with “for Software” and “Software Engineering” of the web page as well.

### 3.4 Putting it all together

Using both the term frequency scoring algorithm, along with the HTML emphasis scoring algorithm, we can handle two important types of concept emphasis which are present in web pages, (1) emphasizing a concept through the amount of times it appears on a web page and (2) emphasizing a concept using HTML tags. In practice, both methods of emphasis are used, so we decided that two additional parameters of importance should be *weights* which represent how much of each type of emphasis should be used to extract concepts from a particular web page. These weights could be learned from examining a web page, or class of web pages from a web site and modified as needed. If a web site contains 50% web pages which use term frequency to emphasize concepts and 50% web pages that use HTML emphasis to emphasize concepts, then the weights for each scoring algorithm would be assigned as .5 and .5 respectively.

Using this method of combining the scoring algorithms, we are forced to save the final weights that each scoring algorithm (TF and HTML scorer) assigned to its respective concepts. We can then combine the concepts returned from each scoring algorithm using the following algorithm which rounds out our algorithm for automatic concept extraction.

**Given:**

1. A set of concepts  $C_{TF}$  returned from the term frequency scoring algorithm
2. A set of concepts  $C_H$  returned from the HTML scoring algorithm
3. 2 weight values,  $W_{TF}$  and  $W_H$  which correspond to the amount of each type of emphasis present in a web page
4. A final threshold  $\theta_f$  corresponding to the cutoff score for final concepts

**Do:**

1. Define a final set of concepts  $C_f$  initialized to the null set  $\{\}$
2. For each  $c \in C_{TF}$  if  $c * W_{TF} > \theta_f$  then accept  $c$  into  $C_f$ , e.g.  $C_f = C_f \cup c$
3. For each  $c \in C_H$  if  $c * W_H > \theta_f$  then accept  $c$  into  $C_f$ , e.g.  $C_f = C_f \cup c$
4. Return  $C_f$

## 4. ACE Implementation

### 4.1 Architecture and Use-Case

Initially a set of keywords, e.g. a query, is sent by the user to the *Concept Search GUI* component. The Concept Search GUI proceeds to forward the user's query to the *SearchEngine* component, which asynchronously forwards the user's query to 5 different WrapperClient components which communicate with 5 respective *wrapper* [4] components using the Theseus [5, 6] plan execution system.

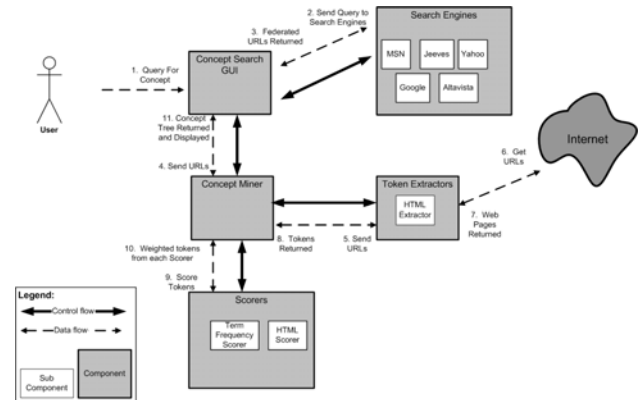


Figure 3. The Architecture of the ACE system

Each wrapper exposes one of the 5 search engines which our system retrieves web pages from: *MSN Search*, *Ask Jeeves Search*, *Yahoo Search*, *Altavista Web Search* and *Google Search*. Each wrapper returns tuples of the form  $\{Description, Link, Title\}$ , where Description is a short description of the web page link returned, link is the actual HTTP link to the web page, and Title is the returned web page link's title. After retrieving the tuples from the search engines, the Concept Search GUI component waits for a user to select a particular tuple, and then sends the selected tuple

information to the *ConceptMiner* component which transfers control to the *TokenExtractors* component, also sending it the tuple data it received from the Concept Search GUI. The *TokenExtractors* component then calls the *HTML Extractor* sub-component, passing it the tuple information. The *HTML Extractor* component then sends a request to the tuple's *link* field to retrieve the web page that the link refers to. Once the web page is retrieved, the web page is *tokenized* (recall Section 3) and the list of tokens is returned to the *Concept Miner* component. The *ConceptMiner* then passes that list of tokens to the *Scorers* component, which then asynchronously sends the token list to both the *Term Frequency Scorer* and *HTML Scorer* components respectively. Each scorer component returns back its own token list, with an additional data field, *score*, attached to each token in the original list. The *Concept Miner* component then examines each returned token list, and then applies the final weighting calculation (recall Section 3) to determine the final concept list. Finally, the token lists returned from the *Term Frequency Scorer* and *HTML Scorer* components, along with the final concept list generated by the *Concept Miner* component are all sent back to the *Concept Search GUI* for display in the left most window. Figure 4 displays a screenshot of the *Concept Search GUI* component.

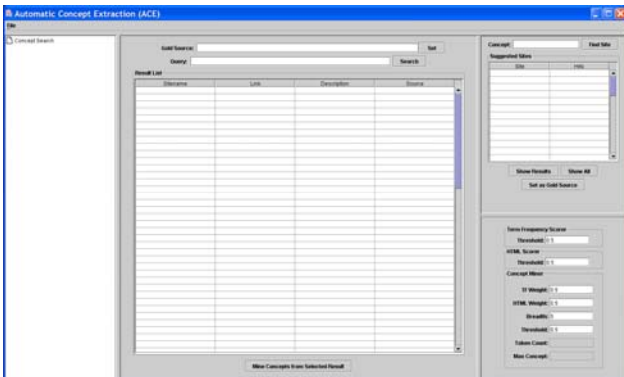


Figure 4. The Concept Search GUI front end

## 5. Related Work

Lin [7, 8] presents work on the subject of automated topic identification. His focus is that of examining a corpus of text and using the frequency of a concept's appearance in the text as a measure of its importance. Concepts are generated using the classification of words by WordNet [9], a lexical database of 120, 400 words organized into 96, 760 concepts, located at Princeton University. Lin's work also uses the  $tf * idf$  calculation to acquire keywords from texts which are classified using the concepts generated from WordNet. Our work differs from Lin's on two fronts. First and foremost, our work is focused on extracting concepts from *web pages* as opposed to from a corpus of text (although a subset of our work is able to deal with a corpus of text given as a set of ordered tokens). Second, and of equal importance is the use of the concepts that we extract: the focus of our work is using concepts to reduce the amount of links that a user must visit while using a search engine. Further, we mine concepts from the *referenced* pages along a user's web page search

path, along with the concepts that we mine directly from the current web page that the user is examining. Lin's corpus of text is pre-fetched and static, and the goal of his work is to pre-classify static documents using the concepts he extracts from WordNet.

Yamamoto et al. [10] and Fuentes et al. [11] highlight two different methods for extraction of classifiers, or *concepts* that describe a set of one or more texts. Yamamoto et al. consider *multi-term* classifications by grouping terms within a text into sets of terms or *classes* using a thesaurus. Classes are used to calculate a score, which is then discriminated against (similar to ACE) to determine a candidate set of concepts for a text. The weights are generated by a calculation based on the  $tf * idf$  algorithm, which is applied to each class in the corpus. Instead of  $tf * idf$ ,  $cf * idf$  (class frequency product with inverse document frequency of classes, e.g. multi-term classification) is used. Fuentes et al. present a system AS, which uses a text's *cohesive properties* to briefly summarize a large body of text. Cohesive properties of a text include the relation of terms to each other (e.g. a similarity score), their lexical classifications and their grammatical classifications. Firstly, our work differs from Fuentes et al. in the sense that we are not focused on extracting summaries of large bodies of text as a whole – in fact our goal is that of summarizing single texts (in our case web pages), *iteratively* throughout the user's search engine experience, allowing the user to provide direct feedback (i.e. through her selection of which links to follow after being presented with a set of concepts). Secondly, our work differs from Yamamoto et al. as we do not use a thesaurus to formulate our multi-term concepts – in fact, we exploit the *area of locality* of concepts within a web page to generate multi-term concepts. Also, because of the nature of an HTML page and the stylistic meaning that certain tags convey, our algorithm can also account for other forms of emphasis *besides the TFIDF-like scoring calculations* that both of these related works present.

Cohen et al. [12] present the *Secondstring* library of approximate string matching techniques, which uses algorithms from the database, AI, and information integration communities to calculate a score which represents the similarity between two compared strings. We originally thought of using *Secondstring* to generate *multi-term* concepts, and indeed we point to this as one potential avenue for future work. Our goal also was to use the *Secondstring* *TFIDF* java class (or the *BagOfTokens* java class) to calculate term frequency for our Term Frequency scoring algorithm; however, we experienced difficulty using the classes due to their explicit *private* declarations (e.g. the Java classes were declared private so they could not be instantiated by the end user of the API).

Landauer et al. [13, 14] introduce the theory of *Latent Semantic Analysis* (LSA). The LSA theory and method is a fully-automated method for determining the context, or set of contexts in which a word can be used by examining a large corpus of text represented as a unique set of words appearing in distinct samples (such as sentences, paragraphs or passages). Given the corpus as input, LSA uses singular

value decomposition (SVD) (a particular type of matrix decomposition) to weight and score words and concepts based upon their appearance within the distinct samples. LSA can be used to classify and categorize concepts appearing within the given corpus of text, much the same way that ACE can *implicitly* classify a web page by automatically extracting out its concepts. Our work differs from LSA in the sense that LSA requires a large corpus of text to classify words (e.g. concepts) whereas ACE is applied iteratively to a set of web page documents. Feedback can potentially be provided during each iteration of ACE, whereas LSA initially must examine large corpuses of text in order to generate concepts from bodies of text.

## 6. Evaluation

We evaluate both the ACE algorithm and the ACE implementation *qualitatively* using the following dimensions, against similar extraction algorithms found in the literature, and surveyed in the Related Work section:

- **Automated Extraction** – the ability of the algorithm to automatically extract out valid concepts (concepts are deemed valid according to a simple hit/miss test in which web pages are assessed by humans, following with manual human concept extraction, then followed by a comparison of the human results to the results extracted out by the machine algorithm)
- **Ability to extract single-term concepts** – the ability of the algorithm to identify concepts containing just a single string term, or word
- **Ability to extract multi-term concepts** – the ability of the algorithm to identify concepts containing more than one term.
- **Ability to extract multi-term concepts automatically** – this refers to the ability of the algorithm to generate or extract multi-term concepts without the use of an outside data source such as a thesaurus.

The evaluation results are presented in Table 2 below. √ implies that the algorithm fully supports the desired properties. — implies the negation.

	ACE	Lin	CW <sup>2</sup>	AS	LSA
<b>Automated Extraction</b>	√	√	√	√	√
<b>Single-term concept extraction</b>	√	√	—	—	√
<b>Multi-term concept extraction</b>	√	√	√	—	√
<b>Automatic multi-term extraction</b>	√	—	—	—	√

**Table 2. Qualitative Evaluation of ACE against other similar concept extraction algorithms**

## 7. References

- [1] S. Brin and L. Page, "The anatomy of a large-scale hypertextual search engine," *Computer Networks and ISDN Systems*, vol. 30, pp. 107-117, 1998.
- [2] S. E. Robertson, "Term frequency and term value," presented at 4th International ACM SIGIR conference on Information storage and retrieval: theoretical issues in information retrieval, Oakland, CA, 1981.
- [3] L. Guthrie, E. Walker, and J. Guthrie, "Document classification by machine: Theory and practice," presented at 15th International Conference on Computational Linguistics (COLING), 1994.
- [4] I. Muslea, S. Minton, and C. Knoblock, "Hierarchical Wrapper Induction for Semistructured Information Sources," *Autonomous Agents and Multi-Agent Systems*, vol. 4, pp. 93-114, 2001.
- [5] G. Barish and C. A. Knoblock, "An Expressive and Efficient Language for Information Gathering on the Web," presented at 6th International Conference on AI Planning and Scheduling (AIPS-2002) Workshop: Is There Life Beyond Operator Sequencing? - Exploring Real-World Planning, Toulouse, France, 2002.
- [6] G. Barish, "Speculative Plan Execution for Information Agents," in *Computer Science Department*. Los Angeles: University of Southern California, 2003, pp. 153.
- [7] C.-Y. Lin, "Robust Automated Topic Identification," in *Computer Science Department*. Los Angeles: University of Southern California, 1997, pp. 243.
- [8] C.-Y. Lin, "Knowledge Based Automatic Topic Identification," presented at 33rd Annual Meeting of the Association for Computational Linguistics, Massachusetts, USA, 1995.
- [9] G. Miller, R. Beckwith, C. Felbaum, D. Gross, and K. Miller, "Five papers on WordNet," Cognitive Science Laboratory, Princeton University, Princeton, NJ, CSL Report 43, July 1990 1990.
- [10] K. Yamamoto, S. Masuyama, and S. Naito, "Automatic Text Classification Method with a Simple Class-Weighting Approach," presented at Natural Language Processing Pacific Rim Symposium, Seoul, Korea, 1995.
- [11] M. Fuentes and H. Rodriguez, "Using cohesive properties of text for Automatic Summarization," presented at Workshop on Processing and Information Retrieval (JOTRI), 2002.
- [12] W. W. Cohen, P. Ravikumar, and S. E. Fienberg, "A Comparison of String Distance Metrics for Name-Matching Tasks," presented at IJCAI-03 Workshop on Information Integration on the Web, Acapulco, Mexico, 2003.
- [13] T. K. Landauer, P. W. Foltz, and D. Laham, "An Introduction to Latent Semantic Analysis," *Discourse Processes*, vol. 25, pp. 259-284, 1998.
- [14] T. K. Landauer and S. T. Dumais, "Latent semantic analysis and the measurement of knowledge," presented at Educational Testing Service Conference on Natural Language Processing Techniques and Technology in Assessment and Education, Princeton, NJ, 1994.

<sup>2</sup> CW stands for "Class Weighting", the approach used by Yamamoto et al.