Musag: an agent that learns what you mean

Claudia V. Goldman and Amir Langer and Jeffrey S. Rosenschein

Institute of Computer Science

Hebrew University

Givat Ram, Jerusalem, Israel

ph: 011-972-2-658-5353

fax: 011-972-2-658-5439

email: clag@cs.huji.ac.il, langera@cs.huji.ac.il, jeff@cs.huji.ac.il

url:http://www.cs.huji.ac.il/$\sim$ $\{clag|langera\}$

Abstract

This paper presents a system that carries out highly effective searches over collections of textual information, such as those found on the Internet. The system is comprised of two major parts. The first part consists of an agent, MUSAG, that learns to relate concepts that are semantically "similar" to one another. In other words, this agent dynamically builds a dictionary of expressions for a given concept that captures the words people have in mind when mentioning the specific concept. The second part consists of another agent, SAg, who is responsible for retrieving documents, given a set of keywords with relative weights. This retrieval makes use of the dictionary learned by MUSAG, in the sense that the documents to be retrieved for a query are semantically related to the concept given.

In this way, we overcome two main problems with current text search engines, which are largely based on syntactic methods. One problem is that the keyword given in the

query might have ambiguous meaning, leading to the retrieval of documents not related to the topic requested. The second problem concerns relevant documents that will not be recommended to the user, since they did not include the specific keyword mentioned in the query. Using semantic methods we will be able to retrieve such documents if they include other words, learned by MUSAG, that are related to the main concept. The system also includes two other agents, USAg and Ag. USAg is concerned with the interface to the user, and directs the user's requests to other relevant agents. Ag is a low level agent that is responsible for the output of the search process.

We describe the agents' system architecture, along with the nature of their interactions. We also describe our learning and search algorithms, and present results from experiments performed on specific concepts.

1 Introduction

We propose the use of a system based on four stand-alone agents whose domain of action consists of huge sources of information that change continuously (e.g., the World Wide Web). We are investigating two paradigms. One concerns learning the meaning of concepts, namely which words are semantically similar. In this case, the semantic connections among words are learned from textual files. The second paradigm we are interested in is to use this semantic information we have learned to make the search for information more efficient, with regard both to time required for the search and the quality of returned results.

There are two main problems with current search engines, which are largely based on syntactic methods. Usually, the user's query consists of a keyword or of a logical query. The syntax-based search engines will retrieve a set of documents that include the keyword given in the query when this keyword has appeared exactly in the same way as in the query. On the one hand, many irrelevant documents might have the same keyword if this word has multiple meanings and will be retrieved for the user. On the other hand, many other documents which might be relevant will not be retrieved if they didn't include the keyword, but did include other words semantically "similar" to the given query.

We want to use the knowledge the user has in mind when he submits a query by specifying a keyword. This knowledge will be learned by an agent, called MUSAg. Musag is the Hebrew word for concept. This agent learns about concepts from documents he "reads". Musag builds a dictionary for every keyword he is asked to learn about. The dictionary consists of other words or combination of words that are related to the keyword. Thus, Musag will capture the implicit and common knowledge existing about concepts in different communities. Whenever an agent searches for documents using the information in the dictionaries for concepts, he will behave as one of the group, as one that already knows the vocabulary used for the domain he is looking for.

Musag is the top agent, whose task is to learn about concepts related to a given keyword. Usag is the second agent in the system hierarchy. Usag (User Search Agent) is the agent that interacts with the user, and directs the user's requests to the other agents in the system. The third agent is SAg, the Search Agent. This agent looks for a page in the Web, given a list of keywords with their relative weights. The lowest level agent is Ag who is responsible for presenting the results of the search and also keeps a data base of related Web pages and their relevant keywords.

In the following sections, we will explain further each one of our agents and how they interact with one another. Before concluding, we present several representative search engines and agents programmed for information retrieval.

2 Musag: The Learning Agent

Musag is an agent programmed to build dictionaries of words, or combinations of words, that are semantically related to a given keyword. These dictionaries capture the semantic knowledge that exists when a user mentions the keyword. Intuitively, if the input keyword is *dai*, we'd like to have a dictionary, represented as a list, composed of words such as {cooperation, coordination, agent, software agents, distributed problem solving}.

Given a page from the Web, we apply several heuristics to extract the *important* words

from this page. We consider that different tags in an HTML document and the text attached to them induce an importance relation over the words in the document. Given a single page, the words we will consider to be inserted in the dictionary for a particular keyword will be those words that appeared in the HTML document next to the following tags: $< title >$, $< head >$, $< h1 >$, $< h2 >$, $< h3 >$, $< blink >$, $< b >$, $< strong >$, $< em >$, $< i >$, $< u >$. Moreover, we also take into account the words that appear as headings for lists, and the first words of sentences that start new paragraphs. In any case we disregard connective words like and, or, for, the, etc. We also regard combinations of words up to three words. Thus, we could consider as concepts in the dictionary expressions like *artificial intelligence*, *software agents*, *distributed artificial intelligence*, and so on.

Whenever Musag is faced with K pages, the words that will be inserted into the dictionary will be those that are considered important and that appear in at least $\frac{K}{2}$ pages

Each word in the dictionary has a learning weight and a search weight. The learning weight for a word i at time t is given by $\frac{\Sigma_t k_i(t)}{\Sigma_i \Sigma_t k_i(t)}$, where $k_i(t)$ is the number of pages in which the word i has appeared out of the K_t (the number of pages given at time t). t is the index that runs over the iterations of the learning program and i is the index for the words in the dictionary. After each iteration, we multiply $\Sigma_t k_i(t)$ by γ, a decay parameter to weaken the weight of words which appear much less in current pages.

The search weight is used to direct the search, hence it should consider the keyword, too. For that purpose, we compute the search weight for a word i at time t as follows: $\frac{\Sigma_t k_i(t)}{\Sigma_i \Sigma_t k_i(t) + \Sigma_t K_t}$ where K_t is the number of documents visited until time t. The search weight for the keyword is given by $\frac{\Sigma_t K_t}{\Sigma k_i(t) + \Sigma_t K_t}$. This is equivalent to assuming that the keyword has appeared in all $\Sigma_t K_t$ documents. For the search weights we also take into consideration words that appear in the URL address. These words will not be part of the dictionary, but are only used to make the search process more efficient (e.g., words like *edu, ac* in the dai example will help the heuristic function identify a promising page, but having them as part of the dictionary will take the place of other words because a word like *edu* appears in many documents).

In the search process a document will be defined as semantically relevant to the keyword given by the user, if there is a combination of words from the dictionary built for the keyword such that the sum of their search weights is at least 0.5. The number of appearances of a dictionary word in the document and its position in it (according to the HTML tags) is also taken into account when evaluating the document's relevance to the keyword. More than one appearance of the word in the text, or the appearance of the word in the Title or Heading (for example) of the page, will result in more than the original word's search weight added to the sum. The search weight for the original keyword is kept always at least 0.5. Therefore, if a document includes only the keyword and does not include any other word from the dictionary, Musag will learn from this document. In addition, Musag might learn also from documents that don't include the keyword but do include a subset of the words in the dictionary. The learning algorithm we apply is based on a majority and intersection criteria.

The agent is an anytime learning program. At any point in time, he is able to output his current dictionary. Currently, Musag performs the first learning iteration after thirty good documents were found. A good document is one which contains any combination of words from the dictionary and the original keyword such that the sum of their search weights is at least 0.5. The second learning iteration is done after twenty more documents were found, starting from the previous thirty documents. From then on, Musag updates the dictionary after every ten documents. Note that the fact that Musag never ceases to learn and the decay factor γ, makes it possible for Musag to automatically update its dictionary when the concept has altered its meaning. K_t, the number of documents tested at each learning iteration is highly correlated with the cost of the learning process the agent (or its programmer) is willing to invest.

3 SAg: The Search Agent

Given a list of words and their relative weights, SAg (i.e, the Search Agent) will retrieve a page in which there is a combination of the words such that the sum of their weights is at least 0.5. In our current implementation, the list of words comprises the dictionary built so far for the keyword Musag is learning about, and also words taken from the path of the document's address. Their relative weights are given by the search weights. We have set a threshold for the search weight of the main keyword that is being learned such that its search weight never drops below 0.5.

Sag is an anytime search program. He runs continuously looking for pages that correspond to the current dictionary. Whenever he finds a document with a value of 0.5 or higher, he sends it to the agent who supplied his keywords (i.e., Musag). Sag implements the A* search algorithm on a spanning tree he creates from the internet. The root of the tree is a document Sag builds, which contains links to documents that are returned by existent search engines: Lycos [6], Yahoo [10], Inktomi [4], Infoseek [3], and Webcrawler [9]. Each page SAg intends to visit is evaluated by a heuristic function given by $hf = \alpha * g + (1-\alpha) * f$, where α represents the weight of the path up to the node being evaluated, $g = 0.7 * \frac{\Sigma(page\ values\ of\ the\ pages\ in\ the\ path)}{\#pages\ in\ the\ path} +$ 0.3 (* 0 if there was a decrease in the page value of the son relative to its father, otherwise 1). f represents the value of the node itself. It is computed from evaluating the words in the URL address, the words in the links' text (i.e, the words that are usually highlighted in blue in a HTML document), the locality of the page, and the document position in its own directory tree.

4 Usag (The User Search Agent) and Ag

USAg is the agent in charge of the interface between the user and the other agents. The first form USAg presents the user with (see Figure 1), enables him to see the list of the concepts that are currently being learned. In the future we will also enable the user to start running a

Musag agent, that is the user gives a keyword in order to build its correspondent dictionary, and eventually search for relevant documents.

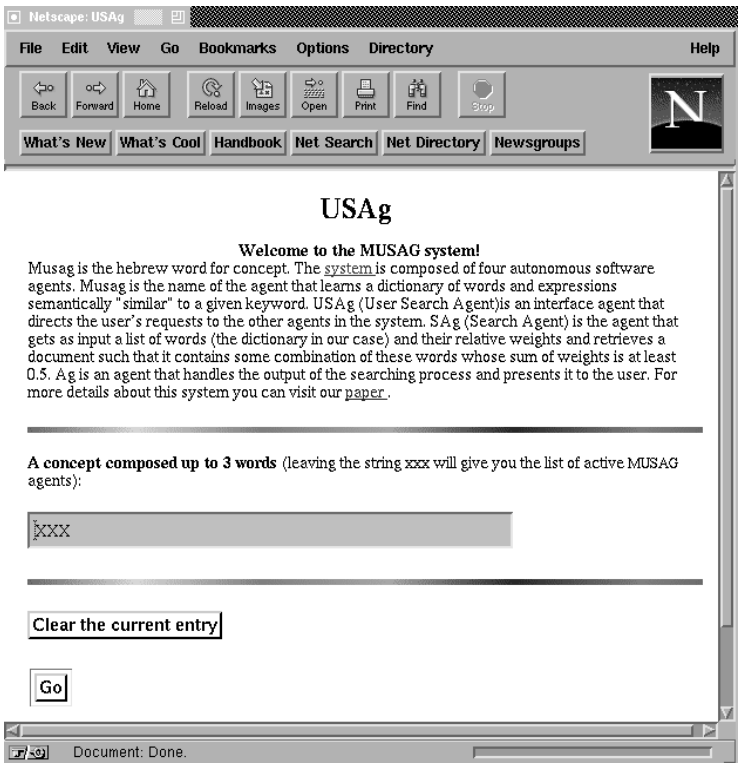

Figure 1: Usag's first form

After the user has chosen one of the concepts (see Figure 2), USAg presents him with another form. The user is asked to specify his requests to Musag, Ag, and Sag (see Figure 2).

Ag is the agent responsible for handling the output. Each time Musag receives a URL from Sag, he sends it together with the current dictionary and the current learning weights to Ag. Ag maintains a fixed table of the best ten documents he got so far. The quality of each page at time t is given by the sum of the learning weights of the words that are part of the current dictionary and appear in this page. In our current implementation Ag presents the user with a user defined amount of documents, sorted by their page values and all the words that appeared in the dictionary and in the documents are specified. In the future, we will also let the user choose another way of getting the output. The documents will be sorted by

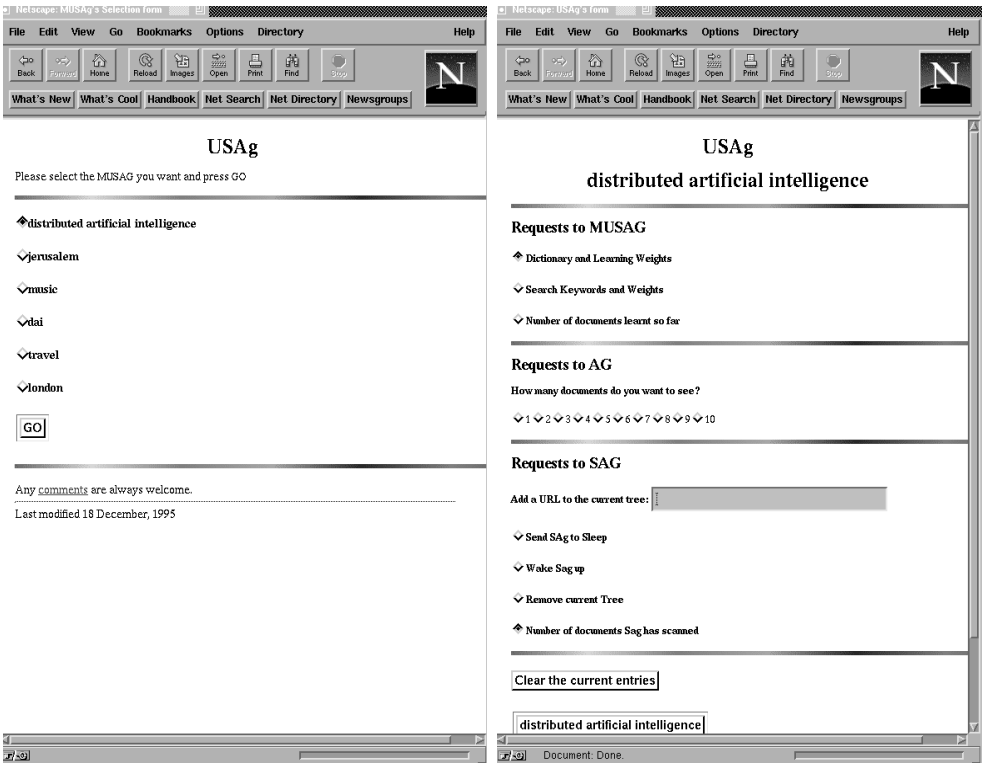

Figure 2: Left: The user is asked to choose one concept

Right:The user is asked to specify his requests

their type (i.e., personal, publications index, institutional home pages, commercial,etc ...). This item can be sophisticated by an agent that is also able to learn about these types of documents.

5 Experiments

We have run Musag on several concepts. We found out that during the first learning iterations, the dictionary included words which were related to the concept (i.e, the *good* words) but also it included words that appeared in the documents that were learned even though they weren't semantically related (i.e., the *bad* words). As Musag continued to learn, the *good* words' weight got stronger, while the *bad* words were gradually dropped from the dictionary. In all cases we have checked, the weights of words in the dictionary didn't prevent new words from being inserted to it. A particularly interesting example is when we tried to learn about the **dai** concept. The Search Agent reached a site which was dedicated to dai, but its documents were written in german. Since Musag was not programmed to ignore words such as *die* and *und*, they quickly became the strongest words in the dai dictionary. Nevertheless, as Musag kept learning from more documents, he overcame this problem, and eventually he got rid of all the non relevant words and enabled others to be inserted to the dictionary.

Figure 3 shows the output USAg presents the user with. The dictionary Musag has built after learning from 1060 documents is displayed on the left. On the right, we can see some of the documents Ag recommends to the user. As we explained in Section 3, Musag also learns about words that appear in the URL address in order to direct the search towards more promising sites. In this case, Musag learned that Jerusalem is highly connected to hosts located in il (i.e., Israel), huji (i.e., The Hebrew University in Jerusalem), and ac (i.e., academic domain).

Another example we ran regarded the concept *distributed artificial intelligence*. The dictionary at the first learning iteration included the following words: `formation, intelligence,`

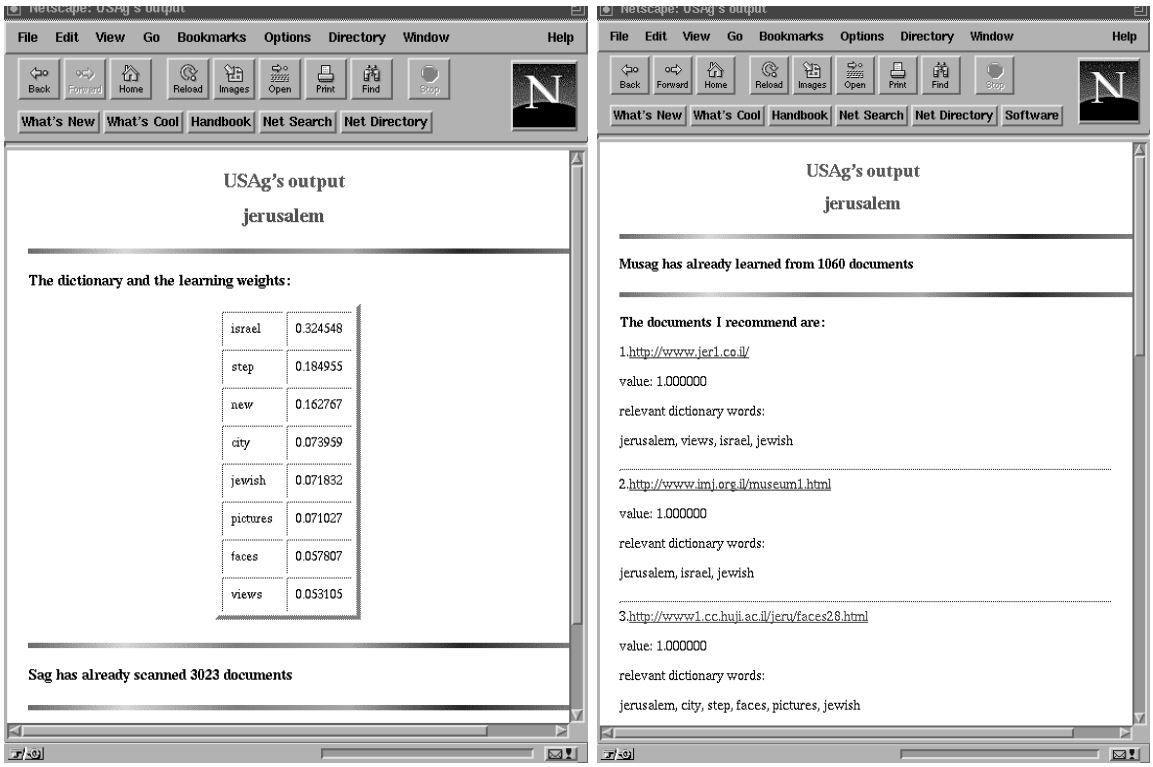

Figure 3: The Dictionary and the documents recommended for Jerusalem

durfee, workshop, framework, washington, integrated processing, university, update, knowledge, visit, laboratory personnel, learning, try, lesser, multi, negotiation, systems, special issue, modeling, seattle, new, papers, parallel, planning, processing, proceedings, real scheduling, agent, agents, ai, automated contracting, artificial intelligence, available, computer, blackboard systems, coalition formation, control issues, cooperative information gathering, design real scheduling, cs, cooperation, dai During the next ten iterations, the words integrated processing, visit, laboratory personnel, try, parallel, processing, real scheduling, automated contracting, available, blackboard systems, control issues, design real scheduling were dropped off the previous dictionary. During the next eight iterations the following words were deleted from the dictionary: framework, washington, special issue, seattle, coalition formation, cooperative information gathering. On the other hand, the words distributed, research, and group were added to the dictionary after the second and third iteration. distributed also became the word with the highest weight in the dictionary after twenty iterations.

A question that remains to be answered regards the evaluation of the quality of the dictionary at a given time. It is not yet clear whether the learning process is monotonic over time. Theoretically, the more we learn from documents related to the concept, the better we know about it. However, the order in which we visit these documents may influence on the quality of the dictionary. This might cause the words in the dictionary to deviate from the original meaning. During the tests we have performed, we found out that the search sometimes converges to a very promising site and stays there until all its pages are checked. This may harm the dictionary by inserting to it words which appeared many times in this specific site and thus presents a wrong image of the keyword meaning. One way to avoid this is to force the search to jump among several hosts under SAg's control. Currently, in most cases, we overcame this problem by waiting for more learning iterations. Since the agents are anytime programs, then eventually SAg will jump to other sites in the search tree.

It is obvious that syntax-based search engines could make mistakes when the concept they are asked for has multiple meanings. For example, we queried Yahoo [10] about the

concept *dai*. The results we got didn't have any relation to the research area of distributed artificial intelligence. They included:

- Dai Nippon Printing Co., Ltd (http://www.dnp.co.jp/)
- University Microfilms International (http://www.umi.com) which provide Dissertations Abstracts International database
- Baltimore Zen Meditation Society, Shorin-ji (http://www.gl.umbc.edu/~jcovey1/zendo.html)

When Musag encounters an ambiguous concept, he either returns a dictionary which reflects this ambiguity or he will let one of the meanings dominate the dictionary.

Musag will also retrieve documents that do not include the keyword explicitly, but have words that belong to this keyword's dictionary. For example, when Ag was asked to retrieve documents related to *distributed artificial intelligence*, one of his recommendations was a document located at http://louise.vfl.paramax.com/~mckay/pst-paper.html. This document, entitled *View-Concepts:Knowledge-Based Access to Databases* does not include the combination distributed artificial intelligence, but it did include *ai, artificial intelligence, and distributed AI* more than once.

When comparing Musag's results with the human conception of a certain keyword, we can see that Musag is not precise in the sense that it makes mistakes. The point to emphasize is that we don't learn about absolute meanings of words, but we aim at capturing their context in which most of the times they are mentioned, that means we aim at learning their meaning out of documents where they appear. For example, if we think of London as a keyword, we might say that it is a city, the capital of the United Kingdom. However, when Musag is asked about London, he came up with different aspects of the city like dining, and sightseeing in addition to the words city and uk that also were included in his dictionary.

6 Putting All the Agents Together — Summary

In this section, we present the complete algorithm for the system composed of the four aforementioned agents. The system is able to learn semantic knowledge related to a keyword given by a user, and it can also find documents that are conceptually connected to this keyword.

Schematically, our system is shown in Figure 4:

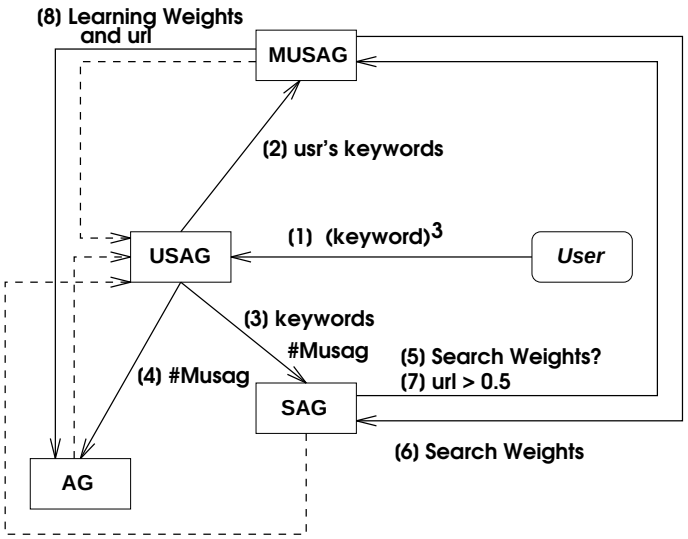

Figure 4: Musag, USAg, SAg, and Ag

The user can submit at most three keywords to USAg to run a Musag agent on this concept. Whenever Musag is started for this concept, also a SAg and Ag agents are initialized to cooperate with this specific Musag agent. Then (see step 5 in Figure4), SAg conitnuously asks Musag for the search weights of the words Musag has already learned about. Whenever SAg finds a document which includes any combination of words from the dictionary whose sum of search weights is at least 0.5, he sends this document address to Musag. Whenever, Musag gets a document from SAg, he sends it together with the current dictionary to Ag. All the responses from the agents to the user are presented by USAG (see the dashed lines in Figure 4). The number of communication messages is reduced by attaching a blackboard to each agent, from which the other agents can read. They read from it, only if it was changed.

The learning algorithm follows:

For a given keyword, `kw` :

1. Compute the search weights for `kw` and all the words in its current dictionary. The search weight for $word_i$ is given by $\frac{\Sigma_t k_i(t)}{\Sigma_i \Sigma_t k_i(t) + \Sigma_t K_t}$ (The initial search weight for the kw is 1.0).

2. Find K_t pages which include any combination of the words in the current dictionary together with `kw`, s.t. the weight of this combination is at least 0.5. This computation is done according to the search weights.

3. Get the *important* words from all K_t pages found in the former step. From the extracted words, find all those that appear in K_i pages among the K_t pages, s.t. $K_i > \frac{K_t}{2}$. Add these words to the dictionary. Add K_i to the sum of every word which is already in the dictionary.

4. Compute the learning weights for every $word_i$ in the dictionary, according to the following formula: $\frac{\Sigma_t k_i(t)}{\Sigma_i \Sigma_t k_i(t)}$

5. Goto step 1 ($t = t + 1$).

The agents implemented in this system are:

1. **Ag** — Given a list of pages and the current dictionary with the correspondent learning weights, this agent presents the user with the best documents he has so far. He also maintains the data base with a predefined number of these documents.

2. **SAg (The Search Agent)** — Given a tree, a list of words (i.e., the dictionary and the `kw`) and their search weights, this agent returns one page s.t. exists a combination of the given words in the page, whose sum of their search weights is larger than 0.5. This agent implements step 2 above.

3. **USAg (The User Search Agent)** — This agent is an interface agent which enables the user to control and see the output of various Musag, SAg and Ag agents.

4. **MUSAg (The Agent that learns concepts)** — Given K_t pages this agent finds the *important* words that appear in the intersection (or majority) of at least $\frac{K_t}{2}$ pages. These words will be inserted to the dictionary. This agent updates the learning weights for all the words in the current dictionary. This agent implements steps 1, 3 and 4 above.

7 Related Work

In recent years, there is an increasing amount of works on intelligent software agents on the web which deal with information retrieval and involves learning. M. Balabanovic and Y. Shoham [2] constructed an agent that learns about a user's preferences of web pages and suggests new pages to the user from a Best-First search it performs on the web. This may be similar to a case where we will let MUSAg learn from pages which appear in a person's bookmarks file (for example), with no initial concept, and build a dictionary of words which describes this person's preferences. The main difference lies in the fact that their learning system is based on explicit evaluations of the search results done by the user.

Other direction of research tried to assist the user when browsing the web by suggesting what link to choose or where to go from its current position. Letizia [5] is an interface agent which tries to model the user's browsing behavior by learning from the pages the user decided to view. WebWatcher [1] works in a similar vein although there, the recommendations are based on a goal you state before the search. Both agents supplies interactive assistance to the user who performs the search and do not handle any queries the user might have.

Queries are handled by all the indexes and search engines that can be found today on the internet. These indexes include agents which build and update a data-base such as Webcrawler [8] and Lycos [7]. None of those agents base their database on semantic methods.

Other index-based search engines as Yahoo [10], bypassed the semantic issue by presenting the user with a list of predefined domains. These domains enable the user to focus his search, although the user might be interested in a concept for which there is not any directory.

The fact that this indexing of the Internet is done by humans together with the vast and continuous growth of the Web, makes this solution non accurate and too static for such a dynamic world.

8 Conclusions and Future Work

We have implemented four stand-alone software agents.

USAg is an interface agent that enables users to make requests to the other agents in the system. Ag is an agent that maintains a data base of documents that will be eventually retrieved to the user if he asks for them. This agent qualifies each of these documents so that at each period of time he can retrieve a user defined number of them that are considered the best so far.

Musag is a learning agent, able of building dictionaries of expressions semantically close to keywords given to him. This agent learns from textual files, in our current system these are files written in HTML. The point in having a learning agent is an interesting research question, because having an agent that learns the semantics of a concept is like learning the language of a group of experts. That is, this agent will capture what these experts have in mind when they mention a concept, what is their common knowledge.

Another agent that has been programmed is called SAg. This is a search agent, that looks for documents which include some combination of words the agent is given as input together with their relative weights. In our current system this agent is given the dictionary built by Musag and returns to it a document that contains a combination of words such that their sum of search weights is at least 0.5. Therefore, the documents that will be retrieved to the user will be semantically connected to the keyword (or expression) he has given to the system.

By applying learning while searching for information, we intend to overcome two main problems in syntactic based search engines. One is that the documents retrieved include the keyword submitted in the query, but since this keyword might have ambiguous meanings,

not all the documents retrieved will be relevant. The second problem is that we will not get documents in which the keyword did not appear explicitly, but we nevertheless might be interested in them because they are about the topic we are looking for. These documents will contain other words that are semantically connected to the keyword.

Having expert agents for specific topics will make a search more specific, efficient, and fast. The agent already knows where he should look for, or what he is looking for; instead of searching a whole data base, he knows from where to start or what is the area in which he should find the answer.

8.1 Future Work

We have already begun to increase the sophistication of our current system. Issues that we are currently dealing with include a more interactive output presentation of the answers given to the user. This includes, for example, a refinement step in which a human expert will critique the dictionary, and add some information, or even the same user will focus the search process according to his preferences given the current dictionary.

We intend to interleave another learning process to learn about different types of documents, and use this information to guide the search and concept learning processes. We'd like to add to USAg learning capabilities so that he can learn about his user and then direct the other agents accordingly (e.g., learn from pages that are of a type preferred by the user. Search documents that are personal pages only). We thought also of letting the user submit the keyword and also a specific key field, for example author or institution. Then, Ag, when asked for the output, instead of retrieving the documents, he will only retrieve the information asked in the specific key field. Namely, the user will get the list of all the institutions that have research groups in dai, if for example the concept submitted was dai, and the key field was institution.

We are also extending the current system to deal with many agents that learn about the same concept and we are looking into ways in which these agents will learn from each other's dictionaries. There will be many SAg agents, so many interesting research questions arise

regarding the interactions among the SAg and Musag agents (i.e, how are they matched? When does the search agent know to ask for a "better" dictionary? How, when and at which cost should they address the other to get information?).

References

[1] Robert Armstrong, Dyne Freitag, Thorsten Joachims, and Tom Mitchell. Webwatcher: A learning apprentice for the world wode web. In *Working Notes of the AAAI Spring Symposium Series on Information Gathering from Distributed, Heterogeneous Environments*, 1995.

[2] Marko Balabanovic and Yoav Shoham. Learning information retrieval agents: Experiments with automated web browsing. In *Working Notes of the AAAI Spring Symposium Series on Information Gathering from Distributed, Heterogeneous Environments*, 1995.

[3] Infoseek. http://www.infoseek.com.

[4] Inktomi Web Services. http://inktomi.berkeley.edu/.

[5] Henry Lieberman. Letizia: An agent that assists web browsing. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, Montreal, Quebec, 1995.

[6] *lycos*^{TM}. http://twelve.srv.lycos.com/.

[7] Michael L. Mauldin and John R.R. Leavitt. Web agent related research at the center for machine translation, 1994. Presented at SIGNIDR.

[8] Brian Pinkerton. Finding what people want: Experiences with the webcrawler. In *Proceedings of the Second International WWW Conference*, 1994.

[9] *webcrawler*^{TM}. http://webcrawler.com/.

[10] Yahoo! http://www.yahoo.com.