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, 91904 Jerusalem, Israel

ph: 011-972-2-658-5188

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/~ $\{clag|langera\}$

Abbreviated Title: Context Learning on the Web

*An earlier version of the paper was published in the Proceedings of PAAM96. This version of the paper has not been published elsewhere and it has not been submitted simultaneously for publication elsewhere.

†Supported by the Eshkol Fellowship, Israeli Ministry of Science

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. We aim at achieving this by learning from the context in which these words appear.

The second part consists of another agent, SAg, which 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 related to the concept given according to the context of previously scanned documents.

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 context learning methods we will be able to retrieve such documents if they include other words, learned by Musag, that are related to the main concept.

We describe the agents' system architecture, along with the nature of their interactions. We describe our learning and search algorithms, and present results from experiments performed on specific concepts. We also discuss the notion of "cost of learning," and how it influences the learning process and the quality of the dictionary at any given time.

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 in this research. One paradigm 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, i.e., they are learned from the words' context. The second paradigm we are interested in is to use the semantic information that 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 those irrelevant documents 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.[1] This agent learns about concepts from documents he "reads." Musag builds a dictionary for every keyword it 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, it 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

[1] "Musag" is the Hebrew word for concept.

for presenting the results of the search and also keeps a data base of related Web pages and their relevant keywords.

Each agent has been designed to be functionally independent. More capabilities can be added to each of the agents to enrich their performance. For example, Usag might learn from the user about his preferences and interact with him accordingly. Ag might also choose a specific way to present the output to the user based on a model it might have about its user. A schema showing the interactions among the agents is shown in Figure 4.

In the following sections, we will explain further about each of our agents and how they interact with one another. Then, we consider how the learning process might be affected if there is a cost associated with the resources used by the learning agent. We also examine the different types of documents that might be encountered on the Web, and how this classification can shed light on different classes of learning rules. 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}.

The main role of Musag is to *learn* what words should be included in the dictionary attached to a given keyword. These words do not necessarily express the meaning of the keyword, but they might be chosen because of the context in which they appear. These two characteristics make the dictionary built by Musag different from on-line dictionaries such as WordNet [wora]. We will further discuss these differences in Section 8.

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 for insertion in the dictionary for a particular keyword will be those words that appeared in the HTML document next to the tags such as: $< title >$, $< head >$, $< h1 >$, $< h2 >$, $< blink >$, $< b >$, $< strong >$, $< em >$, $< i >$, $< u >$. 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 dictionary concepts expressions like *artificial intelligence*, *software agents*, *distributed artificial intelligence*, and so on.

In the present work, Musag learns from pages on the Web. The agent is presented with a predefined number of pages (e.g., K) at a time, from which it is expected to learn about important words and their context. We have experimented with a learning criterion that tells the agent to consider a word relevant if it is important and it has appeared in a majority of the pages presented. Therefore, whenever Musag is faced with K pages, the words that will be inserted into the dictionary will be those important words 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 that appear much less frequently in current pages. Eventually, words with very small weights will be dropped out of the dictionary.

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).

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 as at least 0.5.

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. 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 also learn 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 majority and intersection criteria.

The agent is an anytime learning program. At any point in time, it is able to output its 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.

2.1 Experiments

The system MUSAg has been run on several concepts. It was found that during the first learning iterations, the dictionary included words which were related to the concept (i.e, the *good* words) but it also included words that appeared in the documents that were learned, even though those words were not semantically related (i.e., the *bad words*). As Musag continued to learn, the *good* words' weights got stronger, while the *bad* words were gradually dropped from the dictionary[2].

In all cases that have been checked, the weights of words in the dictionary didn't prevent new words from being inserted into it. A particularly interesting example is when we tried to learn about the **dai** concept. The Search Agent reached a site that 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, it overcame this problem, and eventually it got rid of all the irrelevant words, and enabled others to be inserted to the dictionary.

Figure 1 shows the output with which USAg presents the user. The dictionary Musag 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 was explained in Section 3, Musag also learns about words that appear in the URL address so as 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, 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,`

[2]As long as the dictionary gets focused, bad words will eventually appear less often and therefore their learning weights will become smaller. As explained in Section 2, words whose learning weight drops below a predefined threshold will be removed from the dictionary

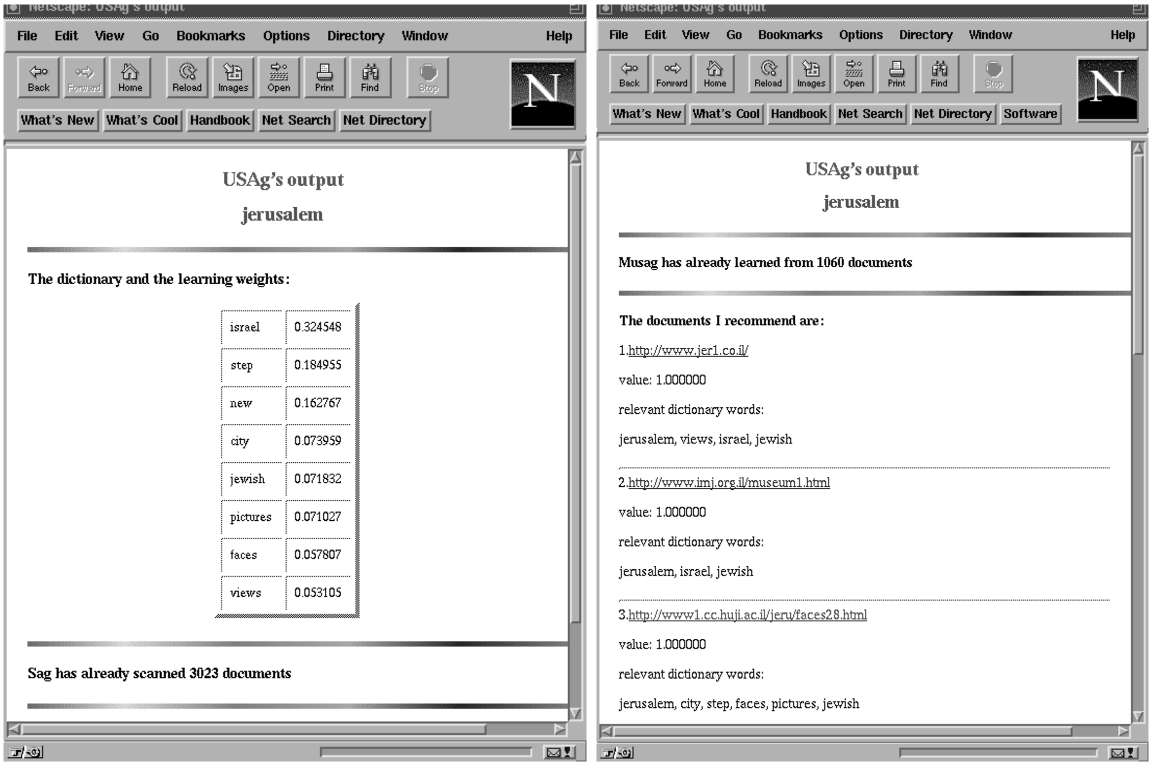

Figure 1: The Dictionary and the documents recommended for Jerusalem

`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.

Other experiments that were run considered the dictionaries for the concepts travel, Mozart, chess, and chemistry. The dictionary for travel consisted of the words `world, new, special, services, one, guide, information, center, contact, check and business`. After having learned from 290 documents the word `airline` was added to the dictionary, and after 420 documents the word `go` was added. Additional words that were included in the dictionary for some time are `directory, out, questions, and copy`.

The dictionary for Mozart included `wolfgang amadeus mozart, remarkable musician, weber, salzburg, musician, music, musical, composers, information, festival, classical, amadeus mozart, age`. Additional words, whose inclusion was due to their appearance in documents but which we might have liked not to have, are `out, new, search, see, after, toilette, topics, joshua, available`. After Musag has learned from 170 documents, new words were added: `wolfgang, wolfgang amadeus, amadeus, van beethoven, ludwig van beethoven, beethoven`. The dictionary after 210 documents included also `orchestra, symphony, classics`. Looking at the documents from which Musag has learned, we found sites that publish cosmetics and software tools with the name Mozart (e.g., http://alpin.or.at/gemr http://www.mozart.com).

The dictionary built for chess included `games, search, uscf, new and contents` (uscf stands for The United States Chess Federation). The dictionary remained unchanged after learning from 50 documents until 1130 documents.

The dictionary for chemistry remained rather static between the 50 documents and 260 documents from which Musag learned. It included words such as `computational chemistry, molecular, chemistry resources, chemical, american chemical society, analytical`. In addition it also included words mostly related to academics, like `university, science, course, department, information, electronic, software, scientific, search, institute, project, world`.

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 more we know about it. However, the order in which we visit these documents may influence the quality of the dictionary. This might cause the words in the dictionary to deviate from their original meaning. We further discuss this question regarding the evaluation of the quality of the dictionary at a given time in Section 6.

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, it came up with different aspects of the city like dining and sightseeing, in addition to the words city and uk that were also included in its dictionary.

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. It runs continuously looking for pages that correspond to the current dictionary. Whenever it finds a document with a value of 0.5 or higher, it sends it to the agent that supplied its keywords (i.e., Musag). SAg implements the A* search algorithm on a spanning tree it 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 [Lyc], Yahoo [yah], Inktomi [ink], Infoseek [inf], and Webcrawler [web].

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, The path's value is computed by evaluating the average page value from all the pages in the path, taking into account a decrease in the last node in that path (which may point to us that this path is no longer promising as it was in the upper levels of the tree). f represents the estimated 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 an HTML document), the locality of the page, and the document position in its own directory tree.

3.1 Experiments

During the tests that were performed, we found 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 into it words that appeared many times in this specific site, and thus presents an incorrect image of the keyword's 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 were able to overcome this problem by waiting for more learning iterations. Since the agents are anytime programs, SAg will eventually jump to other sites in the search tree.

It is obvious that syntax-based search engines can make mistakes when the concept they are asked for has multiple meanings. For example, we queried Yahoo [yah] 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, it either returns a dictionary that reflects this ambiguity, or it 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.

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 2), 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.

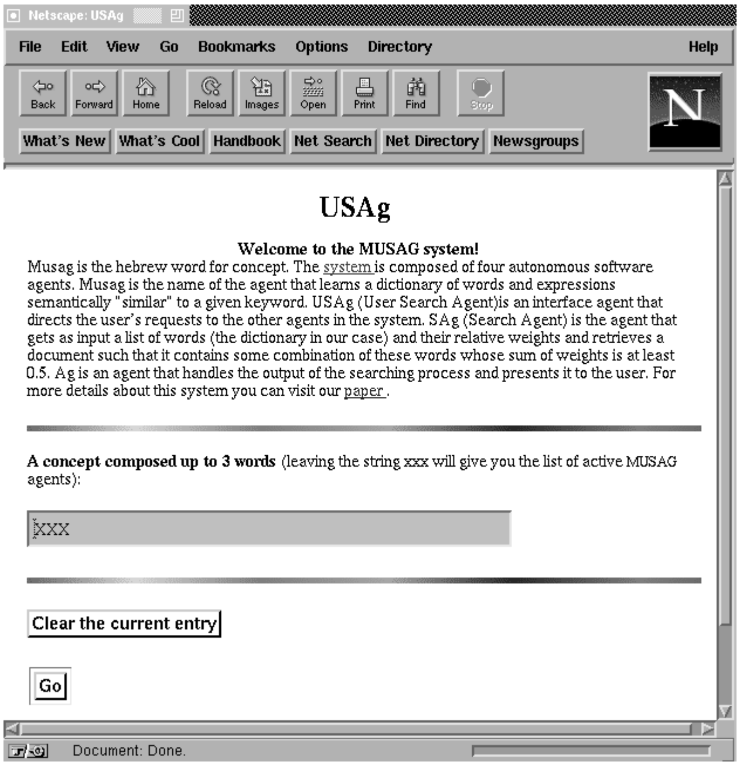

Figure 2: USAg's first form

After the user has chosen one of the concepts (see Figure 3), USAg presents him with another form. The user is asked to specify his requests to Musag, Ag, and SAg (see Figure 3).

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 it has gotten so far. The quality of each page at time t is given by the sum of the search weights of the words that are part of the current dictionary and appear in this page. In our current implementation, Ag presents

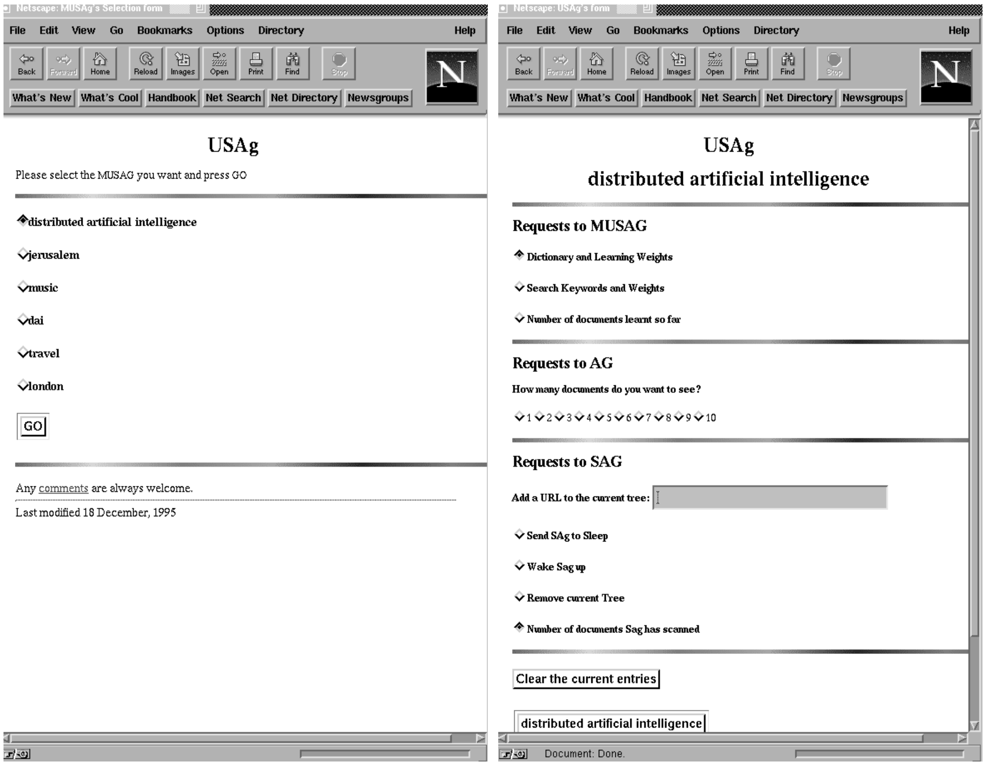

Figure 3: Left: The user is asked to choose one concept

Right: The user is asked to specify his requests 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 future versions, we the user will be able to choose another way of getting output. The documents will be sorted by their type (i.e., personal, publications index, institutional home pages, commercial, etc ...). This technique's sophistication can be increased through the use of an agent that is also able to learn about these types of documents.

5 Putting All the Agents Together — Summary

In this section, we present the complete algorithm for the system MUSAg 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:

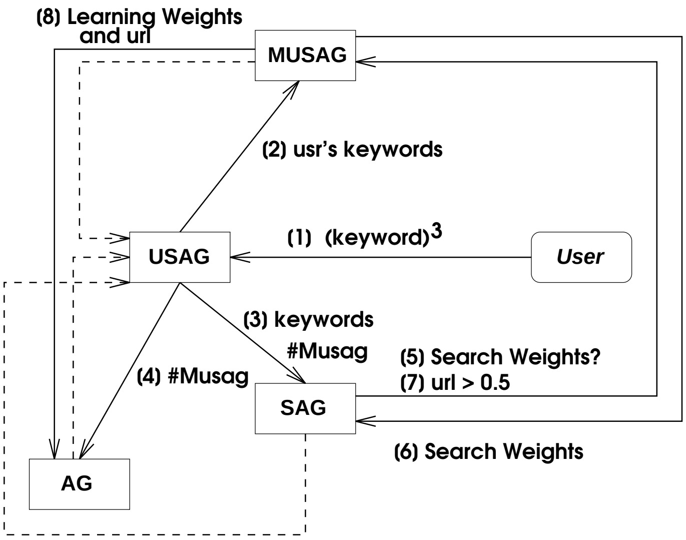

Figure 4: Musag, USAg, SAg, and Ag

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, SAg and Ag agents are also initialized to cooperate with this specific Musag agent. Then (see step 5 in Figure4), SAg continuously asks Musag for the search weights of the words Musag has already learned about. Whenever

SAg finds a document that includes any combination of words from the dictionary whose sum of search weights is at least 0.5, it sends this document address to Musag. Whenever, Musag gets a document from SAg, it 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 it has so far. It 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.

6 Evaluating Musag and the Cost of Learning

Currently Musag is an anytime learning agent. It keeps learning as long as it runs in the system. During the learning process, Musag changes the words in its dictionary. In this way, Musag can become more focused on a specific topic related to the original keyword it is learning about. On the other hand, it might diverge to other topics. The main purpose of the system we have presented in this paper, was to learn a small language for a group of people that work together, or interact on a certain subject. To achieve this, we need Musag to *understand* that the dictionary it has built so far is *good enough* for describing the language it was asked to learn. It is not necessarily the case, that the more Musag learns the more it knows; the knowledge Musag has (expressed in the words in the dictionary), is not necessarily monotonic with the user needs. Hence, we want to have Musag evaluate its current dictionary at each time, and based on this value decide whether it should stop learning or not. This is a point that has not been dealt in most other agent systems that learn about their users.

Another important point to consider is the effort the learning agent invests in learning. This effort can be translated into the resources the agent uses when learning. The user that has a learning agent running on his computer will be interested in reducing the amount of CPU time that his agent uses. Moreover, the cost that an agent might incur when learning might be taken into consideration when a foreign computer decides to host a mobile agent that will run on it to learn from its files. For example, a learning agent might pay with electronic cash to be able to learn from a foreign host.

These two issues lead us to add stop conditions to the learning algorithm for Musag (or any other learning agent). The learning agent will stop learning when the agent *decides* that what he has learned so far is *good enough* for his user, or when the agent has invested *enough effort* in the learning process. We will denote by ϵ the maximal amount of learning units it is allowed to use to learn about a concept. For example, if we assume that each learning iteration costs one learning unit, then Musag will run for ϵ iterations. Other examples could be to define a learning unit as a function of the CPU time, or as a function of the time the agent is running on a specific computer. In this way, agents that learn from busier computers could be charged with larger amount of learning units, than agents that learn on idle machines. We will also refer to a function `Eval(Dictionary,t)`. This function will give us a numerical value that represents the quality of the dictionary at time t. As we will show below, this value could be computed for example, from a function that evaluates the quality of the documents that are retrieved given the dictionary at time t.

When considering the cost of learning, the agent could return to the user, (and keep in its memory for further searches performed by SAg), the last dictionary built after having learned for ϵ learning units. Another possibility is to return the best dictionary, according to the function `Eval`, gotten during the learning period. For this case, we will add another parameter, $\delta \leq \epsilon$, that will denote the price the agent is willing to invest to look for a better dictionary than the one he already has.

We suggest three different approaches to the function `Eval`. In the first case, the value of the function will be computed from a more active interaction with the user. The second approach is heuristic. `Eval` will be computed from the quality of the documents retrieved by SAg at the time of evaluation. The third approach is a more formal approach and is based on information theory [CT91].

6.1 The Interactive Approach

Usually, the effectiveness of many information retrieval systems is measured by using two parameters, recall and precision [SM83]. Recall measures the relevance of the information retrieved relative to the items in the system, by dividing the number of relevant items actually obtained by the total number of relevant items in the collection of the system. Precision measures the relevance of the items retrieved by dividing the number of relevant items actually retrieved by the total number of items retrieved. In our case, Musag runs on the Internet, and therefore we do not know in advance what is the number of relevant documents and we cannot compute the recall of our system. This is also true for other information retrieval systems that base their search on the whole Web. On the other hand, precision is a subjective measure, that has to take into consideration what is relevant for the user.

Hence, our first approach to evaluate Musag is based on the interaction with the user. The user can ask Musag to see the current dictionary and the weights the agent has given to the words so far. Then, the user by clicking on the words could sort them in a new way, giving more importance to the words the user considers more relevant to him. Musag will hold a dictionary that it is learning as an average dictionary for the people whose language it is trying to learn. Each specific user, could ask to adjust the dictionary learned so far for his private use. So, for example, Musag can have been learning for a while and can hold a dictionary for the concept *dai*. In the same group of dai researchers there are those that are more interested in planning and there are others that are more interested in learning. Then, given the dictionary so far each one could change it so that the search would continue in a more personal way. Whenever the user is satisfied with the dictionary Musag presents him with, the user can ask Musag to stop learning. From then on, any search that will be performed by SAg will be based on the dictionary the user has chosen. Every new user, could ask to adapt the average dictionary to his use. So different users can ask Musag stop learning at different points.

There are concepts that might have several meanings. The intervention of the user can guide Musag to specific meanings he is interested in.

Our aim is to present the dictionary in a graphic way (e.g., a pie chart). Then the user could change the words and their weights dynamically by enlarging the area of the words he is interested in more, and automatically reducing the other areas.

6.2 The Heuristic Approach

In order to have Musag decide autonomously when it can stop learning about a given concept, it can evaluate the quality of the documents it retrieves for its current dictionary. A dictionary would be considered "good" whenever a new relevant document (retrieved by SAg) would get a high evaluation score. Repetitive high evaluated new relevant documents, would suggest that the dictionary built so far captured the context of the concept. This evaluation measures the matching between the documents and the dictionary. This is the main difference between learning semantics and learning the context of words. In extreme cases, this approach may give a high score to a bad dictionary from a semantic point of view.

The optimal test for the dictionary would be to evaluate it on a set of known "good" documents. This would be a case for supervised learning, where an expert or a user could supply these documents. Since we cannot assume this in the general case, we must suffice with the documents SAg can return. If there is a close matching between the words in those documents and the words in the dictionary, then we can conclude that the dictionary is close enough to what SAg can find on the web and there is no need to continue learning.

This measure does not take into account the actual meaning of the words in the dictionary. The former approach, that lets the user directly interfere in the learning process, is based on the semantics as the user would like to have. In this case, however, Musag can compare between dictionaries, by comparing the quality of the documents retrieved for each of them. Thus, this approach will induce a measure of distinction. In other words, a dictionary will be considered good if it distinguishes between documents related to the concept being learned and documents that are not related to this concept. Therefore, this is an appropriate measure for guiding the search in the Web, a *good* dictionary will cause the retrieval of *good* documents.

An example of qualifying the documents is to sum up the search weights (i.e., the weights learned taking also the weight of the original concept into consideration) weighting them by their position in the document (according to the HTML fields) and by their number of appearances in the document. Musag can sum up the quality of N documents retrieved by SAg, and then give this value as a result of `Eval(Dict,t)`. Then as explained before, Musag can invest ϵ steps to find a better dictionary, namely one that will increase the quality of the new documents that could be retrieved.

We will use this heuristic to test the performance of the system. We are also interested in testing whether the learning process is monotonic in the learning iterations. We assume that it can be the case that the dictionary can be spoiled with time, and we expect to stop the learning when the dictionary is useful for us using the heuristic explained in this section.

6.3 The Information Theory Based Approach

Another approach we analyze is based on information theory tools. We will define p_i as the probability that word i in the current dictionary appears in a *good* document. A *good* document is a document which is retrieved by SAg as a relevant document to the concept being learned. p_i, is therefore given by the number of documents from which Musag has been learning from and in which word i appears, divided by the number of documents from which Musag has been learning. We will also define q_i. This is the probability that word i that appears in the current dictionary, appears in a random document. Then, we can compute the cross entropy, that is the distance from one distribution to the other. In this way we can estimate the distance between the dictionary Musag has learned so far and a random dictionary. A random dictionary is a dictionary created using the same rules Musag uses, on random pages from the web.

The distance from p_i to q_i is given by $\Sigma_i p_i log\frac{p_i}{q_i}$. i are the words in Musag's dictionary. We will denote this distance by D_{pq}.

An important point to remark on here is that D_{pq} expresses the distance between Musag's dictionary and a random dictionary. So, this distance measures how far or close is the dictionary Musag has learned so far from a random dictionary. In this case, the `Eval(Dict,t)` has to do with the randomness of the words in the dictionary. We cannot analyze the dictionary as being *good* for the user, but as being more or less random.

The problem with this approach is that we are interested in capturing the notion of a *good dictionary* for a user, and this notion can not be described by the probability p_i mentioned above. The measure D_{pq} will distinguish between random dictionaries (denote them by $Dict_R$) from non random dictionaries (denote them by $Dict_{NR}$). According to this measure, dictionaries in $Dict_R$ are not as good as dictionaries in $Dict_{NR}$. However, according to what Musag is learning about, we encounter two kinds of problems.

The first type concerns dictionaries that will belong to $Dict_R$, and as users we'd like to evaluate as *good* dictionaries. There are two cases. One consists of dictionaries that Musag learns about concepts that are very common in Web pages, and thus p_i of the related words will be almost random, namely p_i will be very close to q_i leading to a small D_{pq} and a small value for `Eval(Dict,t)`. An example of such dictionary is the dictionary learned for words like Internet, Web, WWW, etc... The second case of dictionaries that will be underestimated is the one considering words which p_is are very close to their q_is, although they do not have to be necessarily very common words. This will happen with words that are not necessarily very common, but have several meanings, leading to small values of p_i. Since the words are not common their q_i values will also be small. Examples of such words include Jordan (i.e., the country, the basketball player, the Jazz musician etc.), process (distributed processing, peace process, etc.).

The second type of problem regards those dictionaries that belong to $Dict_{NR}$ and thus will be considered *good* according with the measured discussed in this section, although as users we'd not prefer them. This case includes dictionaries with words with p_is different from their q_is. This difference does not say anything about whether the words are relevant to the concept learned or to the user.

7 Learn First How to Learn

Currently, Musag uses some heuristic rules on the HTML tags to extract the important words from the pages it receives. We believe that this part of the agent's learning algorithm can be improved by learning the web style of writing and having a better understanding of where to look for the important words given a web page.

The WWW is an environment which is shared by many people from different cultures and different backgrounds. In order to easily understand one another, the WWW community have developed certain codes, customs and unwritten rules which are accepted and known to everyone who is in the web. The huge growth of the web meant that the rules/customs are simple, and involve common sense in order for everybody to figure them out. This growth also meant that it is getting harder to change these rules since there's a good chance that no one will notice the new style and this forces a lot of the pages all over the web to use the same rules. Only very powerful companies in the WWW can introduce a change in the web style codes such as Frames which were introduced by Netscape (http://home.netscape.com) and became very popular in many sites' home pages. The codes or rules we are referring to may be specific keywords which are used all over the web (such as "homepage", a page called "index.html" or a link titled "back"). They can also be the look and content of a web page. (A homepage of a site will generally include links to other pages on the same host, and will rarely include a lot of text.)

These style-rules also apply to the whole organization of a site and specific custom types of pages evolved to serve specific purposes. (For example, a page which is actually a list of links, an archive of mail messages, an index of images (using thumbnails), etc.) By looking at the whole structure of an HTML document and using these rules, Musag may acquire new knowledge about the page it tries to learn from. More specifically, we would like to learn the pages' type and use this knowledge to direct our extraction of important words. The types we define should be such that suggest different places for important words. For example, in a mailing-list or newsgroup archive, we may give more emphasis to words which are repeated a lot in the text, while in a list of links we may ignore the text and concentrate only on the link anchors text. By finding a page's type we may also learn to ignore completely pages of error or "page has moved" messages. We are currently working on a page's type classifier and examining two approaches: The rule-based approach, where we define certain rules to describe a type. The information theory approach, where by defining a n-dimensional space according to measurable parameters of the document structure and style-codes implementation, we will be able to use a classification algorithm such as the k-Nearest Neighbors algorithm to classify pages. According to our experiments so far, the rule-based approach seems more promising. The types of pages we defined:

1. a site's front page.
2. mailing-list/newsgroup archive
3. list of links
4. personal page
5. mainly text - article, text file etc.
6. mainly image/s
7. page of data/tables/lists
8. interactive page - applet/query/game/etc.
9. message of moving to another location
10. brief text message - disclaimer/about etc.
11. error message
12. multimedia page - links to video/audio files etc.
13. a page which includes both an opening section and the whole text itself (A whole site inside one document).

We defined dozens of rules to capture the current web-styles. There are rules which look at the semantic contents of the page, or its URL. For example, we are looking if there's a link titled "Home" in the page. Obviously, this may suggest that this is not a home-page. There are rules which examine the appearance of certain HTML tags or the amount of appearances compared to the page's total number of tags. For example, we will look how many different images are in the page, or if there is a FRAMESET tag in the page. We also look at the links from the page. Are they all references to the same page ? Are there any references to special types of documents such as ".wav",".au" files ? etc.

Musag can focus its learning of a specific page if it already knows what type of page it is. It will need first to characterize the page according to several heuristics, and then extract the important words from that page, based on the page's type. For example, if we need to learn from a newsgroup archive then probably most links will be links to the email addresses of the authors, but to learn from the page we will try to focus much more on words which appear in the text. When trying to learn from a list of links, we would try to give more importance to the links and their titles instead of the document text. A page that is dominated by images can be better learned by also trying to get words of the images' names or `ALT` attributes. Moved and error messages can be ignored by the learning agent.

8 Related Work

This work deals with two main research questions. One is to learn about information according to the context in which it appears. The second is to build a sophisticated tool for searching the Web based on the knowledge acquired in the learning process.

Different formal approaches have been studied with respect to context (for a survey, see [AS96]). One important and useful task, in which context needs to be regarded, is related to information retrieval tasks. A query will be better defined when considering the context of the query. Therefore the performance of the information retrieval system will increase.

In our system, we developed an agent whose task is to learn the context of keywords submitted by a user, on the Web. Musag, the learning agent, learns dictionaries for different keywords. These dictionaries might differ from regular ones in the sense that they might include words that were found to be contextually related to the given keyword, but they are not necessarily the keyword's meaning. For example, people might often use abbreviations, commercial names and combinations of words to refer to a subject that they are acquainted with. With our system, we will be able to search for material about this kind of information after we have learned about their context. Many on-line dictionaries can be found on the Web; for an index of on-line dictionaries see [onl].

WordNet [wora, worb] is an on-line thesaurus built on psycholinguistic theories of human lexical memory. Therefore it gives comprehensive information about words in a lexicon. Nevertheless, this differs from our approach in that the source knowledge is not learned but written by lexicographers. An example of a search engine that uses WordNet is EchoSearch [ech]. EchoSearch uses syntactic search engines to get files for a specific concept and then performs some semantic parsing of the pages' text using WordNet. Being a commercial product, EchoSearch does not explain its ideas and methods but this is also an attempt to attack the Web search problem by more than just comparing strings. EchoSearch main problem is that it bases its results on regular search engines and so it cannot add results to what is already returned by those search engines. Instead, it focuses on a more semantically logical display of the results.

In recent years, there has been an increasing amount of work on intelligent software agents on the web that deal with information retrieval and involve learning. M. Balabanovic and Y. Shoham [BS95] 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 directions of research have tried to assist the user when browsing the web by suggesting what link to choose or where to go from its current position. Letizia [Lie95] is an interface agent which tries to model the user's browsing behavior by learning from the pages the user decided to view. WebWatcher [AFJM95] 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 [Pin94] and Lycos [ML94]. None of those agents base their database on semantic methods.

Other index-based search engines as Yahoo [yah], 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.

9 Conclusions and Future Work

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. Experiments with MUSAg showed that there is an improvement over syntactic based search engines when dealing with retrieving relevant documents that do not necessarily include the keyword. This is caused by the way Musag's dictionary is built and held. The dictionary includes words related contextually to the initial keyword. Musag's dictionary also aids us when tackling the problem of ambiguity to a certain keyword. If the dictionary is not ambiguous then the relevant documents will have greater weight. The main problem Musag still faces is how to avoid holding an ambiguous dictionary, and how to converge to the meaning which was the user's intention in the first place. An extension to deal with this problem can be implemented by adding to USAg learning capabilities of the user preferences and interests.

We have also discussed the notion of cost of learning. We showed three approaches to evaluate the dictionary at a given time. Based on a criterion that Musag can apply on the quality of its current dictionary and the price it was designed to invest in the learning process, it can decide to stop learning. Any further search will be based on the static dictionary Musag holds at the time it stopped learning. The basic idea is to get to a dictionary that is representative *enough* for the *user* and the given concept.

We believe that the HTML pages on the WWW store more knowledge than just their tags' meaning. We believe that most pages on the internet obey certain style-guidelines and rules which we can learn and use. This will enable us to classify web pages to specific types and in the case of Musag, may improve the learning process of extracting the important words from the documents. This classification can also be used when retrieving the answers to the user, by enabling him to specify the type of pages it wants as answers. This classification process requires a new agent that will interact with the current system.

The system we have presented is composed of four stand-alone agents. Although in this paper all the agents act together in the same environment, the Web, we can run the agents autonomously. Musag might learn about a concept from a specific data base or archive of documents, and even from a user's bookmarks. In such cases, Musag might apply more sophisticated learning rules to choose important words and to weight them. Each one of these different sources of information have different features and might have different sizes leading to more specific learning rules. The notion of cost of learning changes also when the source of information is finite. Then, we might want Musag to check the whole collection of documents to build its dictionary. After having learned the dictionary, SAg can search another collection of documents (e.g., the Web), using the dictionary learned by Musag.

Ag, the output agent, has also been designed as a separate agent because it has its own functionalities. These include being customized for a specific user, presenting the output in different ways appropriate for its user, storing the documents retrieved so far in a data base. 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.

9.1 Future Work

One of our future projects is to implement a more customized version of Musag. This includes a more interactive interface, in which a specific user could interfere and guide the learning process for his personal use as we explained in Subsection 6.1. The presentation of the output can also be designed more user oriented. USAg and Ag can learn about the types of documents the user prefers (e.g., from previously learning about their user). Then, the agents could guide the search and present the results accordingly (see Section 7). For example, the user might prefer to get documents of a certain type such as commercial or personal homepages.

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, 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.

Currently, the dictionary is presented as a set of words, ordered linearly by their learn weights. We are interested in presenting the dictionary in a structured way. For example, the words in the dictionary might be clustered in a hierarchy according to their co-occurrence. From a semantic point of view we'd like to learn about meta topics that are represented by subsets of words in the dictionary. This involves the definition of a metric among the words in the dictionary.

For example, when Musag was given the concept Jerusalem, it learns that city and step were two words in the correspondent dictionary. The word step was inserted into the dictionary because there were many documents describing a trip in Jerusalem composed of steps. In such case both words, city and step are related to Jerusalem, but in two different layers. We'd like to say that Jerusalem is a city and it is related to something else that is related to steps. It is not necessary that the name of the meta topic should be known in the learning process, but it is more important to distinguish among the layers, as in city and step. In a later step, the user could give names to these clusters of words.

We also expect to refine the learning rules that Musag applies, by adapting the rules to the types of documents (see Section 7).

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

[AFJM95] 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.

[AS96] Varol Akman and Mehmet Surav. Steps toward formalizing context. *AI Magazine*, 17(3):55–72, Fall 1996.

[BS95] 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.

[CT91] T. M. Cover and J. A. Thomas. *Elements of Information Theory.* John Wiley and Sons, Inc., 1991.

[ech] Echosearch. http://www.iconovex.com/ECHO/ECHOS.HTM.

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

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

[Lie95] Henry Lieberman. Letizia: An agent that assists web browsing. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, Montreal, Quebec, 1995.

[Lyc] $lycos^{TM}$. http://twelve.srv.lycos.com/.

[ML94] Michael L. Mauldin and John R.R. Leavitt. Web agent related research at the center for machine translation, 1994. Presented at SIGNIDR.

[onl] Index of on-line dictionaries. http://www.bucknell.edu/ rbeard/diction.html.

[Pin94] Brian Pinkerton. Finding what people want: Experiences with the webcrawler. In *Proceedings of the Second International WWW Conference*, 1994.

[SM83] G. Salton and M. McGill. *Introduction to Modern Information Retrieval.* McGraw Hill International Books, 1983.

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

[wora] Wordnet. http://www.cogsci.princeton.edu/ wn.

[worb] Introduction to wordnet:an on-line lexical database. ftp://clarity.princeeton.edu/pub/wordnet/5papers.ps.

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