ProtoSpider: Using Semantic Networks for Keyword Analysis and Document Classification
(Under the direction of Antonie Stam)

A large part of electronic commerce is dealing with information, particularly finding the right information amoungst the gigabytes of information available. This paper addresses this need by outlining a prototype of an information gathering agent for the world wide web. Put simply, an information gathering agent is a type of web spider which roots around on the web for information relating to a specific topic. This paper mainly deals with an implemented prototype for this sort of agent, focusing on how to specialize the spider to the topic. It also deals with agent architectures, multi-agent systems, agent communication and other issues involved in agent based programming.

Index words: semantic network, software agent, web spider

ProtoSpider: Using Semantic Networks for Keyword Analysis and Document Classification

by

John Eikenberry

B.A., The University of Toledo, 1992

M.Phil., University of Toledo, 1995

A thesis Submitted to the Graduate Faculty

of The University of Georgia in Partial Fulfillment

of the

Requirements for the Degree

Master of Science

Athens, Georgia

1998

ProtoSpider: Using Semantic Networks for Keyword Analysis and Document Classification

by

John Eikenberry

Approved:

Major Professor

Date

Approved:

Graduate Dean

Date

Contents

Chapter 1

Motivation

An integral part of all information gathering agents is information filtering, or the ability to determine if the information in a certain document is the type of information the agent is intended to gather. The implementations currently available seem to fall into two broad camps in the context of this project, the overly simple and the overly complex. The methods used by simple web gathering filters consist primarily of doing keyword searches on various web indexes and editing out duplicates. Some try to supplement these simple methods with various techniques for combining keywords, such as using 'AND' and 'NOT', to get a more targeted set of documents [1]. Though this helps a little, it still leaves the system relying entirely on the web index as its primary filtering process and offers no access to other sources of information (eg. email, usenet). At the other extreme are very sophisticated and complex systems. These systems use such things as natural language processing techniques, constraint-based logic, and pre-built, full-dictionary semantic networks [2]. These techniques, usually used in some combination, can provide a powerful mechanism for 'understanding' the content of a document and filtering correctly at the cost of being non-portable, computationally expensive, and hard to maintain. In addition to theses costs they tend to be overly specialized and brittle; being good at the particular target application, but not very extensible.

My goal is to strike something of a middle ground between these two camps. That is to build a filtering technique that will be more discerning and independent than the simple systems while not becoming overly complex and allowing for a variety of uses. This design builds upon the simple filtering techniques by retaining the possibility to use outside sources to start the filtering process while not being dependent on them. It consists of a set of agents that parse out the document, analyzes its content and return a report on its relevance to the topic. The process of parsing out the document is simple and well understood, it consists only of parsing out the html, noting the title, headers, meta-tags and the other text. This infomation is then ready to be analyzed.

[1] http://www.agentsoft.com/
[2] http://www.excalib.com/products/rw/rw.html

The analyzer then looks for notable keywords, words that could be important for capturing the subject of the document. Usually this is determined by looking for uncommon words that appear often in the document, words that are common in that document while being uncommon in other documents. These keywords are then fed into a simple semantic network. The semantic network slowly develops an 'understanding' of these words by their relation to the other words in the network. The relations in the network are simple syntactic associations like 'next-to' and 'same-sentence'. The relations are also scored with either a positive or negative weight. The higher the score the stronger the relation, with positive relations meaning that the two words having that relation on that document raises the likelihood that the document contains the information being sought, with negative relations having the opposite effect. As new documents are analyzed and the user gives some feedback to the system as to its effectiveness, the network will add or remove keywords and adjust the score weightings of the relations.

An agent using this system might work as follows: first the agent sends a query to a search index on the internet. This provides a good way of getting a set of pages without requiring a predefined set of starting points or having to just search the whole web. The agent would then fetch each of the documents that the search index has returned links to and analyze them. If the web page scores well, the agent could feasibly traverse the links on these pages as well. After a set amount of time or having gathered a certain number of pages, the agent could then present the user with a set of links, or even a custom newspaper containing the information gathered.

However, because the filtering system is independent of the search indexes, it can be used on other information mediums like the more sophisticated or complex systems. It can be used in the same way to analyze usenet postings, email, or even live IRC chat sessions. This is of key importance, because while the web is the most popular source of information, it is not always the best. Many times a mailing list can provide much earlier reports of news events or newsgroups can provide a more variety of outlooks on the subject. The agent may even find out about new web pages not in the search indexes from the content of these alternative forms of information. The modular design of the system also lends itself well to new extensions of its abilities, say when XML starts getting used.

Chapter 2

Literature Review

This section will presents synopsys of various papers related to software agents in general and information gathering agents in particular.

2.1 PAWS: An Agent for WWW-Retrieval and Filtering[1]

This paper presents an overview of the theory and technology behind this agent implementation. PAWS is an agent designed to gather web documents which fit a 'profile' and keep them in a database. A profile for this system refers to (usually) subject based classification, examples of profiles might include: 'linux', 'astronomy', etc. Once the agent has the documents in the database the user has the option of scoring them as good or bad, not scoring is an option. The agent then feeds the scores to a genetic algorithm which is in control of adapting the filter, which is used to determine if a document fits a profile or not, to better capture what the user wants. The PAWS agent system is also designed to work collaboratively with other PAWS, enabling the exchange of different people's profiles.

2.2 AUTOMATING THE INTERNET: Agents as User Surrogates[2]

The author of this papers offers an introduction to the usefulness of agents for automating the users routine tasks on the internet. He discusses four examples and from these examples pulls five desiderata for Web automation. The four examples are:

- gathering articles of interest to the user from multiple newspaper sites each day;
- collecting the information for a stock profile, including stock performance, company information, recent articles of interest;

[1] Falk, Anders. and Jasson, I. *PAWS: an agent for WWW-retrieval and filtering.* http://www.fek.su.se/forskar/program/imorg/dok/1996-08-28_1/erimedlab/paws/NewPAAM.doc.html

[2] Krulwich, Bruce. *AUTOMATING THE INTERNET: Agents as User surrogates. http://computer.org/internet/*

- local network configuration;
- continuously scanning online news sources for articles that mention the companies clients or competitors.

From these examples the author draws the following conclusions:

- automating arbitary Web activity, including filling out forms;
- gather information from various Web pages into a single report;
- act on sceduled bases or on demand;
- integrate information from a variety of sources, both private and public;
- smoothly operate with custom program modules.

2.3 PATTIE MAES ON SOFTWARE AGENTS: Humanizing the Global Computer[3]

In this interview Pattie Maes discusses many general principles dealing with agents, particularly agents dealing in information. Discussed are ideas dealing with what is known as 'collaborative filtering', when the each user's agent collaborates with everyone elses agents to determine similar interests. For example, her original system dealt with music tastes. Each user would give their agent a profile of the types of music they liked. Then their agent would consult its fellows to see if any of their users liked similar music. The agent would then make recommendations based on what other users with similar tastes liked and that agents user hasn't heard of yet. She also made some interesting points about the interfaces to the agents. She pointed out that it is important for the agents not to promote a narrowing of the users information by giving the user only what they asked for. Users tend to like the systems better when they receive unrelated information occasionally, so they have the opportunity to expand their areas of interest.

2.4 Intelligent Assistance for Navigating the Web[4]

The author of this article, Christopher Welty, discusses the use of a deep ontology of the information available on the WWW. A deep ontology is a form of knowledge representation, basically

[3] Maes, Pattie. *PATTIE MAES ON SOFTWARE AGENTS: Humanizing the Global Computer. http://computer.org/internet/*

[4] Welty, Christopher A. *Intelligent Assistance for Navigating the Web.* http://www.cs.vassar.edu/faculty/welty/papers/untangle

a knowledge classification and relational system. He proposes a modification of a certain form of deep ontology that was originally created for intelligent email distribution. He then used his new ontology to show how the information contained on the YAHOO! pages could be searched in many more useful ways. It would be made deeper, since the information would be able to contain more knowledge. For example, unlike current search engines, one could query for and 'find all the home pages of people interested in AI' and receive a list of all the links to AI people's home pages. He doesn't deal with the parsing of the pages for the information, natural language processing or anything like that, he deals strictly with the representation of the knowledge contained in the pages.

2.5 WAG: Web-At-a-Glance[5]

The authors of this paper give an overview of the design for a database system for caching web pages for easy and more thorough querying. The majority of the paper was quite technical in its discussion of this design, but there was an interesting discussion of web page classification. The web page was classified into one of four (4) catagories: an organizations home page; an index page; a personal home page; a document. The classification of the page was determined by statistical methods working over numbers gained from the following information: page size, number of local incoming links, number of outgoing links, frequency of access. With the statistical method and some general heuristics (eg. like a persons home page might have the word 'home' on it), they achieved 96% success in classifying the pages.

2.6 Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents[6]

This paper gives a nice overview and definition of autonomous agents and a nice breakdown of their main characteristics. It lists many of the definitions proposed by major people or organizations in the field, and ends with his own formalized definition:

> An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what is senses in the future.

[5] Catarci,Tiziana, Chang, Shi-Kuo, Nardi, Daniele, Santucci, Giuseppe and Lenzerini, Maurizio. *WAG: Web-At-a-Glance.* http://www.dis.uniroma1.it/~nardi/Ricerca/pubbl/

[6] Franklin, Stan. and Graesser, Art. *Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents.* Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag, 1996.

The paper then proceeds to discuss the main characteristics of agents, though not every agent must have all of these:

Property	Meaning
reactive	responds in a timely fashion to changes in the environment
autonomous	exercises control over its own actions
goal-oriented	does not simply act in response to the environment
temporally continuous	is a continuously running process
communicative	communicates with other agents, perhaps including people
learning	changes its behavior based on its previous experience
mobile	able to transport itself from one machine to another
flexible	actions are not scripted
character	believable "personality" and emotional state

2.7 A Softbot-Based Interface to the Internet[7]

This papers goes over the design points that came up in the development of an agent based interface for finding information on the internet. There were four major aspects of the interface and agent design: the agent should be goal oriented; it should be charitable, or try to complete its goal without demanding all the relevant information from the user; it should try to balance the cost of the finding the information on its own against hassling the user with more questions; it should have an integrated interface so the user doesn't have to learn a new interface for each of the agents abilities. The paper also stressed one additional design factor that came up after the initial implementation. That is that the agent needs to be safe. The agent, by necessity of its functionality, has access to powerful tools which could cause problems if used incorrectly. Thus the author came up with four secondary design principles that underlie the others: the agent should refuse to make destructive changes (reformat a disk drive); the agent should be tidy, or clean up after itself; the agent should be thrifty with its resources (cpu/disk/network usage); the agent should be vigilant and make sure that the users actions don't have unintended consequences.

[7] Etzioni, Oren and Weld, Daniel. *A Softbot-Based Interface to the Internet.* CACM 37(7):72-76.

Chapter 3

Related Agent Projects

This section presents an overview of other agent projects that are available or under development. There are actually several dozen such systems and it is beyond the scope of this paper to cover them all. Thus I will focus on the systems that deal specifically with information gathering and electronic commerce. For each entry I will give a basic overview of the system and then list its strengths and weaknesses.

3.1 Virtual Secretary Project[1]

The Virtual Secretary Project is an academic project's goal is to construct user-model-based intelligent software agents, based on the ideas of mobile computing and computer networks. The agents are meant as replacements for humans for typical secretarial tasks. The project consists of two phases which represent the levels of complexity of the agents. The first phase will be concerned with the low-level details like information retrieval and agent mobility (the ability of the agents to move their process over networks). The second phase consists of developing the agents capable of the higher order tasks. These agents would be the ones that would serve secretarial like tasks and would include limited domain knowledge and problem solving abilities.

- Strengths: implemented in a VHLL[2] ; open source; multi-agent cooperation; access to email; news and WWW; knowledge-based agency; cross platform.
- Weaknesses: experimental platform only; limited developer support; limited code reuse; required programming to develop or customize agents; no user profiling.

---
[1] < http://www.cs.uit.no/DOS/Virt_Sec/ >
[2] Very High Level Language

3.2 Webmate[3]

Webmate is an WWW browsing helper agent. It 'accompanies' you while you surf the web and provides various features to aid in surfing. It provides such things as: parallel searching using multiple search engines, profiling your interests and recommending links based on those interests, checking pages for bad links, offline browsing.

- Strengths: cross platform; open source; user interest profiling.
- Weaknesses: only useful while user is actively surfing the web (not autonomous); only accesses WWW; unable to cooperate with other agents; implemented in a low level language; not easily customized or extended.

3.3 Agentware Suite[4]

Agentware is an end user system for developing intelligent agents. The agents gather text and images from the web and put them together into a personalized newpaper. Agents gather the information based on an initial set of preferences which it adjusts to the user with use to get more specific results. Each agent is specific to a subject of interest and it adapts to that users interests in that subject by adding additional constraints to its search parameters. The agents can be sent out to do general research on a subject or can put together a personalized newspaper by looking at the online news sources for stories relating to the subject the user is interested in.

- Strengths: profiles users interests; multiple profiles based on subject; the agent is able to transfer tasks to a counterpart located on a dedicated server so the agent can continue to work while the user is offline; friendly interface for end-user.
- Weaknesses: proprietary; platform specific (not cross platform); not customizable or extendable; limited to WWW.

3.4 Retrievalware[5]

Retrievalware is a knowledge retrieval and indexing system. It can access the WWW, internal databases, email, pdf files, and lotus notes repositories. It combines agent based technology with

[3] < http://www.cs.cmu.edu/˜softagents/webmate/ >
[4] < http://www.agentware.com/agent/index.html >
[5] < http://www.excalib.com/products/rw/rw.html >

an indexer and front end. It uses agents for information retrieval, and includes natural language processing and semantic networks. The language processing abilities come from a large dictionary of word meanings and language idioms. It contains a pre-built semantic network of word associations relating to the dictionary entries.

- Strengths: word association based semantic network and language processing; end-user friendly front end; access to email, web documents, images, databases, and various other formats of documents.
- Weaknesses: proprietary; no profiling or adaptation; fixed, pre-built semantic network; limited platform support; not customizable or extensible; no access to usenet or other online information sources; very limited agent collaboration.

3.5 VIA[6]

Versatile Intelligent Agents (VIA) is a robust development system for creating agent based applications or integrating agents into other systems. The agents in this system are designed to act like and represent their user on the internet. The agents are very flexible in how they can be used. Each agent has five potential areas for customization: actions - include emailing, phoning, faxing, paging, and a number of other options; environment - runtime infomation resources such as the WWW, databases, local files; reasoning - agent can be built to reason with knowledge bases or stimulus responce; senses - agents can monitor the various environmental sources for changes or certain qualities; stimuli - the specific events that trigger the agents actions. Once developed, the agents can get the specific parameters from the end-user in two ways; a natural language-like text interface and a 'wizard' (a program that walks the user through the process).

- Strengths: fair flexibility (as good as it can get without source code); nice end-user interface; access to WWW, email, databases, and a few other proprietary forms.
- Weaknesses: proprietary; no interest/subject profiling; no scripting interface (bad for developers).

[6] < HTTP://WWW.KINETOSCOPE.COM/VIA/DEFAULT.HTM >

3.6 Inquisit[7]

Inquisit is an internet based service which you pay for the use of their agents. The agents will gather information about specific topics for the user. Users assign tasks to agents via the web and receive reports via email. User can specify topics, delivery times and places (various email addresses). Agents gather information from news wires, newpapers, press releases and magazines. Some of these are dedicated services (paid for, dedicated feeds) which is possible since it is a paid service. This provides only a limited advantage as most of this information is already online.

- Strengths: unique information sources; easy end-user interface; well targeted; doesn't require user to be online.
- Weaknesses: not flexible; proprietary; limited information sources; not customizable or extendible; useless to developers; no profiling; privacy/security issues.

3.7 Bargainbot[8]

Bargainbot is a multi-agent system that finds the best price on a book for the user. It is a good example of a very specific or targeted agent application. The web site based cgi script starts up multiple agents, each checking a different bookstore then reporting back the price it found. The information is then presented to the user from best deal to least.

- Strengths: ease of use; good at what it does; agents report separately, so no one agent will keep the system from reporting; quick responce time.
- Weaknesses: limited usefulness/can only do one thing; resides on server (which means booksellers can easily block it out); only of use to end-users.

3.8 Firefly[9]

Firefly is a server residing agent system for profiling clients. It profiles a client of a website by observing their activity on the website and provides customized web content. Each client is assigned an agent which keeps their profile and works with other agents whose clients have similar profiles. In this manner each agent can adapt to their client while still being able to offer new material based on similar profiles. It is a good example of how to apply agent techniques to the server side.

[7] < http://www.inquisit.com/ >
[8] < http://www.ece.curtin.edu.au/~saounb/bargainbot/ >
[9] < http://www.firefly.net/ >

- Strengths: well targeted; profiles users/clients interests; offers new information based on knowledge; collaborates with other agents.
- Weaknesses: doesn't allow for user control or input in a direct fashion; proprietary; privacy/security issues.

3.9 WebSprite[10]

Websprite is an agent program that gathers specific data automatically from the internet and displays it in a ticker on the users desktop. The information it retrieves must be highly defined like stock quotes, sports scores, horoscopes, etc. The ticker display is kept separate from the agent, and different agents can be plugged into the ticker to provide different information.

- Strengths: unobtrusive/simple interface; well defined role and user expectations.
- Weaknesses: limited usefulness; proprietary; platform specific; new agents provided via proprietary source; not customizable or extendible.

3.10 LiveAgent[11]

LiveAgent is a commercial agent system with a unique 'agent' construction method, aimed toward the non-programmer, developer. The agent creation systems consists of a proxy server sitting between the user and the WWW which records every interaction between the two. The user controls how the proxy server records the agent scripts via a GUI control panel that allows that user to set up conditional events, like downloading an article if it contains certain keywords or only grabbing the titles of each article. The system records everything from text entered into forms, to a javascript onClick() events. After the agent-scripts are recorded, the user can use the agents collevtively to retrieve the information for a 'masteragent tool' which will combine it into a report.

- Strengths: very simple agent construction allowing an end-user to build agents; agents can be used in union to perform a more complex task.
- Weaknesses: proprietary; platform specific; no profiling; no adaptation; little flexibilityl; not as useful for a developer.

---

[10] < http://www.websprite.com/ >
[11] < http://www.agentsoft.com/ >

Chapter 4

Architecture

4.1 Introduction

An information gathering agent requires a technique to determine whether a page is relevant to the users expressed interests or not. This aspect of the information agent works like a filter, weeding out the uninteresting pages and ranking the remainder in relation to the users interests. The design of such a filter should reflect the agents interfacing with it to ease integration issues and provide as robust a filtering process as possible without overburdening the system or the developer. To this end, the design will stress these criteria:

- it should be implemented in an agent oriented fashion
- it should be kept lightweight (ie. easily understandable and not processor intensive)
- it should use as many existing technologies as possible
- it should adapt itself to the user's interests
- it should separate the user's interests into tractable categories

4.2 Agent-Based Infrastructure

As the filter is to be a part of a larger information gathering agent system, it only makes sense that the filter should be structured as an agent as well. To facilitate the description of the filter as an agent I will first briefly describe the agent system that the filter will be playing a role in, this will also expose certain advantages of the design. The filter-as-agent (referred to as filter-agent from now on) will be a part of a larger agent development kit that will provide the building blocks for many types of agents, including information gathering agents. The most important part of the development kit in relation to the filter-agent is how it allows a collection of simple agents to be dealt with as a single agent by the rest of the system. This is referred to as a multi-agent system in much of the agent literature [1]. The Firefly agent system mentioned in Section 11 is a good example of a multi-agent system.

Basically the development environment will consist of a library of simple agents (referred to as agent-objects). These agents will each be capable of performing one fairly specific task, for instance one agent could be responsible for fetching text/html documents while another is responsible for parsing out the URLs from that document. Because each agent does just one thing, and does it well, the system is quite flexible in that you can put together many different combinations of these simple agents to accomplish a wide variety of tasks. The real power of this environment comes into play when you want to group together a collection of these simple agents-objects to form a more complex agent. The kit allows for this collection of agent-objects to be treated by the rest of the system, ie. the other agents, just as if it were a single agent-object (these collections are referred to as menski agents). Using this encapsulation the developer can build complex agents by making collections of agents and collections of collections of agents (menski agents can encapsulate groups of other menski agents) [2].

The filter-agent will be constructed as a collection of agent-objects and dealt with by the rest of the system as a menski agent. With the foundation of the agent-objects, many different types of filtering agents will be able to be constructed. From topical web spiders to newsgroup and email filters. To limit the breadth of this paper, the discussion will be contained to the design of a topical web spider, though the reader should think about how the agent-objects that make up this spider could be used in other contexts.

4.3 Design of a Topical Web Spider

A topical web spider is an application which searches the web for information related to the topic of interest to the user(s) of the system. The filter-agent plays a key role in this application by screening out irrelevant information and only letting the system to present the appropriate items to the user. For this type of application the filter-agent needs to not only screen out irrelevant documents, but it must teach itself about the interests of a particular user so it can achieve this with the best results. Below is an outline of an information gathering agent. The whole agent is

[1] Franklin, Stan. and Graesser, Art. *Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents*. Proceedings of the Third International Workshop on Agent Theories, Architectures , and Languages, Springer-Verlag, 1996.

[2] For more details on this development environment, please refer to appendix A.

outlined here to provide a context for the next section which will go into the details of the filter-agent, or agents as is actually the case. In the outline below, agents marked with * are dealt with in more detail later in the paper and for more information dealing with how the agents system works, see Appendix A.

4.3.1 The URL Finder (Menski Agent)

The URL Finder agent is responsible for finding the appropriate documents on the web. To do this it will use standard web indexes so that it doesn't have to crawl the whole web and duplicate already available information. It should use information from the interest profile (the semantic network) to build the query used for the search engine to limit the starting list of URL's as much as possible. The URL Finder is a menski agent with the agents given below in its collective. The agents are presented in the order they would act in as much as possible, with exceptions noted.

Query-Builder (Agent-Object)*

This agent provide the query for the web index. It either queries the semantic network (details below) for important keywords and builds a string out of them or excepts a query string as provided by the user.

Search Engine Post Builder (Agent-Object)

Takes the query-string, structures it into a http post for a specific search engine as provided by the user (who select from a list of options).

URL Fetcher(Agent-Object)

Fetcher posts to web engine's form specified in the request. It posts the query and gets the resulting web page.

HTML Parser (Agent-Object)

Parses out the web page returned by the search engine and returns list of links to be sent on the the URL Hound.

The URL Hound (Menski Agent)

The URL Hound is a menski agent made up of two agents. The hound's job is to fetch the URLs while avoiding fetching the same page twice.

Target URL Tracker (Agent-Object)

This agents job is to keep track of the web sites that have been visited already, to avoid duplication of effort. It takes the information provided by the URL Finder and the HTML Parser and maintains a list of the sites to be visited and a list of sites that have already been visited. It handles relative links as well.

URL Fetcher (Agent-Object)

This agent is responsible for the actual fetching of the web page. It's job is pretty straight forward, but it does need to be able to handle http errors and network hangs. The initial version of this agent will use asynchronous IO to handle multiple connections, though a threaded model is possible and easily integrated. The results are sent on to the HTML Processor, and the list of URL's visited is also forwarded on to the Profile Builder so it can know when its received all the information for that run.

4.3.2 The HTML Processor (Menski Agent)*

The HTML Parser

This agent parses out the html document. It must parse out all of the following: links to other pages; the full text without html markup; each paragraph; emphasized words (eg. marked with <title>); keywords listed in within <meta> tags. Each of these pieces of information are kept separate to be used by the extractor.

Extractor (Agent-Object)

Parses out keywords from text for each web page; ie. emphasized words, rare words, prominent words, words used many times, etc. It also figures out which keywords exist either next to each other or near each other. All this information is then sent on the the Profile Builder.

4.3.3 The Profile Builder (Menski Agent)*

The Profiler Builder's job is to take the current interest profile, the current set of documents and the users feedback on those documents and update the interest profile with the new information.

Network Builder(Agent-Object)

Builds a bare semantic network for the keywords for each document analyzed. The semantic network being bare means that all the relationships between the keywords are neutral (not scored).

The Report Generator (Agent-Object)

Scores the pages, determines the relevant ones, and sends this information on to the Display Agent to present to the user. It then stores the semantic networds as constructed by the network builder agent along with a list of the URL's while waiting for user feedback which is provided either through GUI or (for the demo) through a web interface. Once the feedback is obtained from the user about the quality of the pages it sends the feedback along with the already built networks to Network Maintainer.

Display Agent (Agent-Object)

Generates the report to be shown to the user based on the relative scores and URL's passed to it by the reporter. Once the user feedback has been given, it sends this information back to the Report Generator.

Network Maintainer(Agent-Object)

Adds user's feedback to the just built semantic networks and merges the now scored semantic network back into main interest profile semantic network.

4.4 The Filter-Agent and its Components

The term 'filter-agent' as use above actually refers to a set of agents and their underlaying components. In this section I will lay out in more detail these components, how the agents use these components and how the agents interact to provide a simple, yet robust, information filter. There are two non-agent components that these agents utilize in their tasks; a semantic keyword network and a global keyword database, and three agent components; the query builder, the content analyzer and the profile builder (covered in the next section).

4.4.1 The Semantic Keyword Network

This is really the core technology behind the filtering system. It allows for the dynamic development of interest profiles in a straightforward and easily understandable way. Most semantic networks are usually prebuilt system consisting of real-world based connectors [3] like 'is-a' or 'has-a', eg. Linux is-a operating system', which are used to represent the meaning of the words in the network. In addition to systems that use the traditional semantic networks [4] to catagorize web pages, there has been work with ontological networks [5] to classify documents or to do concept clustering [6]. Unlike these types of semantic networks, the semantic keyword network looks for patters in the use of words and their structural layout. So instead of connectors like 'is-a' it uses connectors such as 'next-to' and 'near', representing the meaning of the words by capturing the patterns in word structure that coincide with certain topics. See below for a table of the connector types and the relationship they describe.

Connector	Relation
next-to	The words appear next to each other, and are in the same sentence
near	The words are in the same paragraph
base	The words are on the same document (keywords)

In addition to this difference from traditional semantic networks, in the basic types of connectors it utilizes, the connectors also express whether the relationship between the words is a positive one, indicating the document contains pertinent information, or a negative one, indicating the document is not pertinent. It does this by assigning a normalized positive or negative score to each connector. Thus each connector consists of a relation type (eg. 'next-to') and a score ranging from -1 to 1. See the figure below for a simple example.

[3] In semantic networks, the nodes of the network are the words and the connectors are the 'lines' drawn between those words.

[4] http://www.excalib.com/products/rw/rw.html

[5] Eichmann, David A. *Ontology-Based Information Fustion.* http://www.botspot.com/faqs/article5.htm.

[6] M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam & S. Slattery (1998). *Learning to Extract Symbolic Knowledge from the World Wide Web.* Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98).

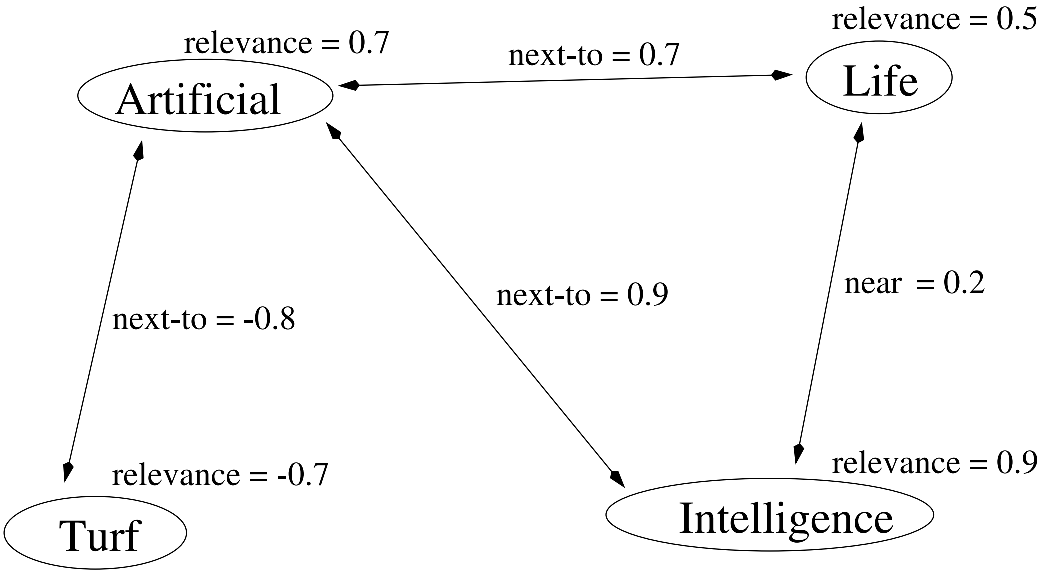


Fig. 1 - a simplified example of a semantic keyword network

For each area of interest of the users, there will exist one such semantic network. The network will start out blank except for the initial keywords fed to it by the user for the index search. It then add in the new information from each document it processes based on user feedback. It will also remove keywords that turn out to be irrelevant, ie. keywords whose connections all fall in the range of 0.1 to -0.1. Thus the information gathered off the net will at least have the appropriateness of the index search, it will grow more specific with time and change with the users interest in a subject matter.

4.4.2 The Keyword Database

The keyword database will store all the keywords from all the semantic networks along with their frequency of occurrence in the networks. Technically this database is where the semantic networks are stored as well, just with a different interface for extracting this information. This is required for the parsing of the keywords as one very good way of telling whether a keyword is representative of a certain topic is how often it is used in relation to that topic compared to how often it is used in other topics [7]. This allows for the semantic networks to become better at their tasks as the amount of information analyzed becomes broader, ie. covers more topics, in addition to becoming deeper, ie. more information on a single topic. The details of how the keyword database is used will be discussed below in the section on the parsing agent or the extractor (mentioned above).

[7] Boone, Gary. *Concept Features in Re:Agent, an Intelligent Email Agent.* The Second International Conference on Autonomous Agents (Agents '98), Minneapolis/St. Paul, May 10-13, 1998.

4.4.3 The Query Builder and the Content Analyzer

These agents use the data collected in the semantic network as an information source only, ie. they don't effect changes on the semantic networks or the keyword database. The query builder agent simply extracts the three most significant keywords from the database whether they be positive or negative, though at least one of the keywords must be positive. Which keywords are the 'most significant' is decided by their relevance score as kept in the database. These keywords are then returned, with negation noted on the negatively scored keywords, and used to build the specific string that each search engine requires for its use.

The content analyzer agent takes the parsed text of a document, as returned by the html parsing agent, and uses the semantic network and keyword database to analyze the text and return a score for it. It matches the keywords in the semantic network against the text of the document, looks for any of the relationships captured in the semantic network, then returns a normalized score of the combined activated connectors and the keyword relevance scores.

$$S = \frac{\sum keyword + \sum near + \sum nextto}{Nkeyword + Nnear + Nnextto}$$

Where *keyword* is the score for each keyword, *near* and *nextto* are the score on each activated connector and N is the relative number of scores for each type.

4.5 The Profile Builder

The profile builder menski agent is a collection of agents responsible for building and adapting the interest profiles. There are four agents in the collective, three of which are worth further examining; the extractor, the network constructor and the feedback enfolder. The fourth, the librarian, simply is an interface to a persistent storage mechanism that stores the semantic networks while waiting for user feedback.

4.5.1 The Extractor

The extractor agent takes the parsed out text from the html parsing agent and pulls out words which would either make a good keyword or are already keywords in the profile's semantic network. To determine whether a word makes a good keyword the extractor scores the word on a number of criteria that measure their possible relevance to the topic (this becomes their relevance score). The process of applying these criteria and how they function is described below, first with an English explanation of the criteria and the equation, then the equation in mathematical notation, and finally a rationale for it. Note that all scores are signed normalized numbers, between -1 and 1.

- If the word contains 3 or fewer characters, eliminate it. Words of 3 or fewer characters will either be articles or very common words, and therefore will be of little use in identifying topical web pages.

- The most frequently used words in the document get a starting score based on the equation below: with N being the number of times the word appears on the page, $Tmean$ being the trimmed mean, the number of times each word appears on the page minus the largest and smallest values, and Max being the maximum any word appears on the page. The $Tmean$ is used because of the way web pages tend to be structured. Web pages tend to be short, consisting only of a few printed pages, and concise. This results in a word freqency spectrum that is heavily weighted toward 1, and doesn't result in a very useful number. By trimming off the lowest frequency values (usually 1) and the highest (to help balance things out), a much better estimate of the freqency of meaningful words is returned.

  $$S' = \frac{N - Tmean}{Max - Tmean}$$

  The most commonly used words tend to be relevant to the topic. Those words which are not relevant will be weeded out later, primarily by the next criteria.

- If the word occurs in other semantic networks, take the current score and multiply it by 1 - the number of other nets it occurs in divided by the total number of nets minus one (don't count its net), the equation below lays this out explicitly: with S' being the currently determined score, N being the number of other networks the word appears in, and T being the total number of nets.

  $$S'' = S' \left(1 - N / \left(T - 1\right)\right)$$

  This equation is used to weed out those words which are common across multiple topics. This equation and the previous are popular for this type of keyword identification [8] , and the two together is commonly referred to as TFIDF [term frequency times inverse document frequency] [9] .

- If the word is more than 6 characters long, take the square root of the current score.

$$S''' = \sqrt{S''}$$

Just as words with 3 or fewer characters are very common in the English language, long words (words with more than 6 characters) tend to be more rare and more specific in their use. Thus long words get their score increased slightly.

- If the word occurs in a 'secondary' heading (h3,h4 in html) in the document, take the square root of the current score.

$$S''' = \sqrt{S''}$$

Html headers are printed in a larger bold-faced font, making them obvious places for section titles and the like. Thus deserving a slight increase in score which squaring provides. It's worth noting here that the combination of two squarings raises the score high enough to be considered significant. Thus if a word is both over 6 characters long and is a part of a secondary heading it will probably be figured as a relevant keyword.

- If the word occurs in the title of the html document or is in one of the primary headings (h1,h2), it gets set to 1.

$$S''' = 1.0$$

Obviously, if the word occurs in the title of the document, it is likely to be relevant to whether that page is topical or not, and is thus automatically considered a significant keyword. Also the primary headings are printed in very long bold-faced fonts, which is used to denote major sections and is nearly as important as titles. Thus making them prime candidates to be relevant to how topical the page is.

- If the word occurs in the meta keyword listing in the html document, it gets set to 1.

[8] M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam & S. Slattery (1998). *Learning to Extract Symbolic Knowledge from the World Wide Web.* Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98).

[9] Lieberman, Henry. *Autonomous Interface Agents.* http://lieber.www.media.mit.edu/people/lieber/Lieberary/Letizia/AIA/AIA.html.

$$S''' = 1.0$$

In html, the meta keywords are used as a way to list the relevant keywords to the document on hand, so it is obvious to consider these significant.

- If the word meets none of the above criteria, it gets set to 0.1.

$$S''' = 0.1$$

If the word has not met any of the above criteria (which most words on the page shouldn't have), they are given a score of 0.1. This score is considered insignificant and these words will not be considered as keywords. But by having a non-0 score, it is possible for them to be raised through squaring to a high enough score to be considered significant.

For the keywords that already exist in the semantic network, if they exist in the document then the above calculations are applied to them as well, if they don't appear they are not scored at this point. Once all the words have been scored, the extractor returns any that are significant. Significant is defined as scores above 0.5.

4.5.2 The Network Builder

This agent takes the keywords as returned by the extractor and builds a skeleton semantic network out of them, ie. the connectors are all in place, but they have no values as yet. All the extractor's information and the parsed document are then discarded at this point. The skeletal semantic network, along with the URL denoting the location of the page it represents, are then stored away to wait for the users feedback.

4.5.3 The Feedback Enfolder

Before this agent acts, the user scores the pages. The scoring is not obvious, hiding behind functionality. For instance, if the user adds the link to that site to their bookmarks this would entail a high positive score, if they choose to have the agents check the links off that page this would also be a positive score, but if they deleted URL to the page this without even reading it this would indicate a high negative score. Once the score is received by the feedback enfolder agent, the scores are applied to the connections in the semantic network. Each connector receives the same score based on the users feedback.

After receiving and recording the scores the user returns, the agent then folds both the new relevance score and the user feedback into the profile's semantic network. This will cause the existing scores to be moved toward the new score by an amount dependent on the the difference between the two scores and the maturity (Age) of the already existing keyword or connection. The exact formula is as follows:

$$St = \left(So + \left((Sn - So)\left((10 * Age)^{1.5}\right)^{-Age}\right)\right)$$

With St being the new total score for that connection or relevance score, So being the old score, Sn being the new score. The Age variable is determined by counting each time a new page is scored and enfolded into the network and dividing this number by a fixed number, with a minimum value of 0.2. The fixed number is doubled any time the count equals it, until a maximum is reached. For instance, the initial fixed number is set at 30, but once the count reaches 30 the fixed value is set to 60. This continues until the fixed number reaches 480, after this point the count is set back to 220 each time it reaches 400.

By starting with low valued Age variables, this allows the network to be very flexible early on. Yet as it matures and becomes more focused on its topic, its scores change less dramatically. But to allow it to branch out every so often, the fixed amount doubles which re-introduces a certain amount of flexibility back into the network. This works well until the fixed number starts getting to high, which is why the fixed number must have a maximum. While this is complicated to explain, it is a very simple and efficient algorithm to implement and does its job well.

It is also at this stage that the keywords and connections with low scores, less than 0.11, are removed from the network. But this only happens when the age of the network is above 0.75. It is done in this fashion to allow the network to periodically gain new keywords and relations that it might otherwise delete and keep them for a while to see if they become useful.

Chapter 5

Prototype and Results

The majority of this paper has focused on the backend components, the keyword scoring and semantic network. In this section I will discuss the prototype which was built to test these components and the results of testing done with the prototype. Since the rationale behind the creation of the prototype was strictly to test the backend components, many aspects of the interface and its functionality were not implemented. These will be implemented in a later phase of this project, after all the backend issues have been resolved.

The prototype was built with as a cgi program with a web based interface. While this is not the target interface, it has several advantages which make it appropriate for a prototype. Web interfaces can be quickly developed, offer a decent cross-platform interface and works well for demos by allowing for the interface to exist on a different machine from the application.

This section will start by explaining the requirements to install and set up the system for testing. A step-by-step sample run through will then be presented to explain how the system works. Finally there will be a discussion of the results derived from the sample run, with particular attention paid to the semantic network that has developed and how well it scores new documents.

5.1 Installation

The platform used for testing was standard PC running the RedHat 5.0 distribution of Linux with Python 1.5.1 installed as the development/running environment. Python 1.5 or newer is required for this system, so make sure you have an appropriate version of Python installed first. This can be done by simple starting up Python at the command prompt '$ python' and checking the version number listed in the startup headers. While the system itself will probably work quite well on many other platforms due to the cross-platform nature of Python, it has only been thoroughly tested on this system.

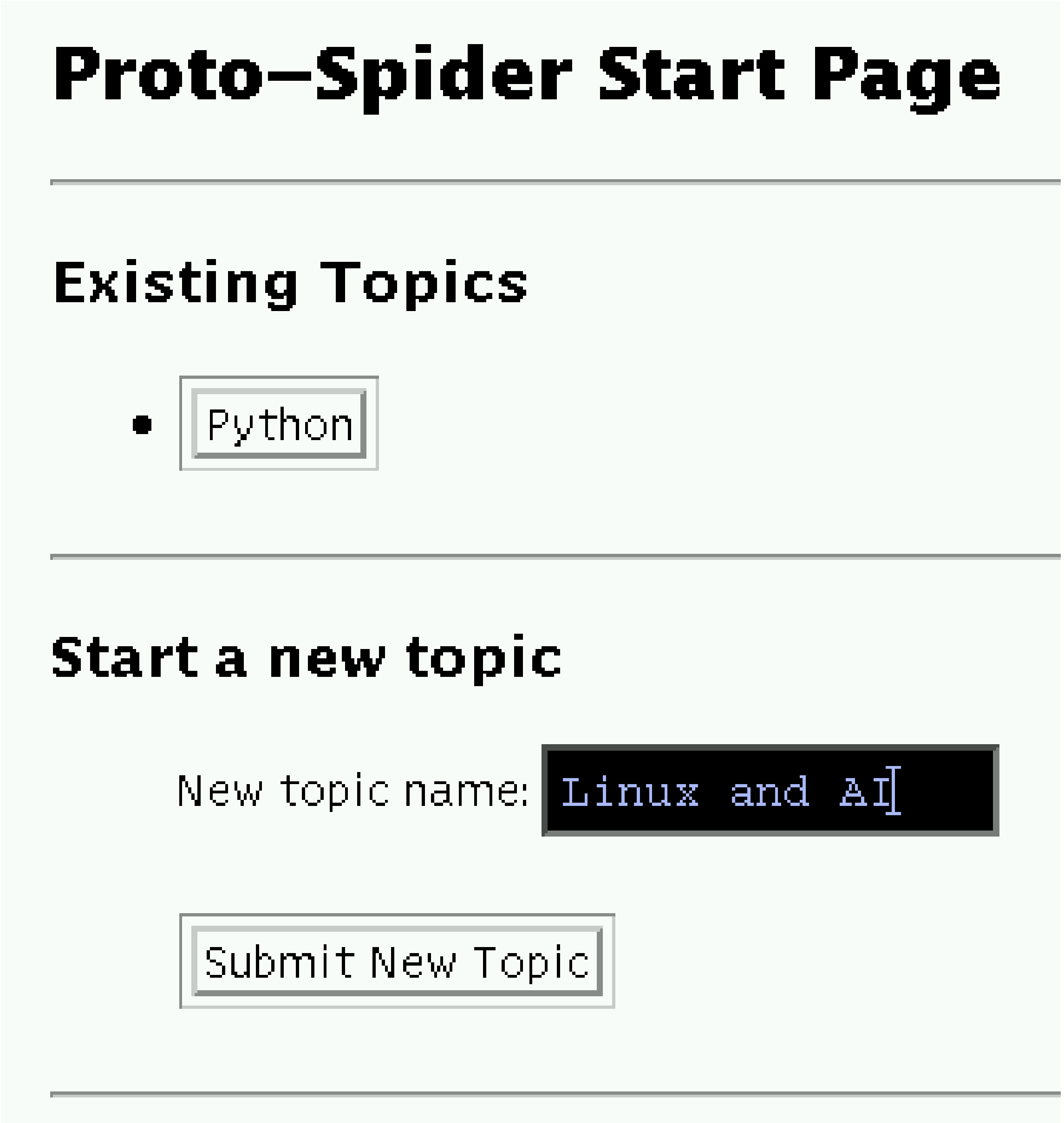
To install the system on a Linux system, simply uncompress and untar the package and place the files in a suitable location. To unpack the system, use the following command:

```
tar -zxvf protospider.tar.gz
```

After unpacking the system, move the protospider.py file to a cgi-bin directory, on Linux systems using Apache the most obvious place would be /home/httpd/cgi-bin. You will also need to change the 'path' variable at the top of the protospider.py file to point to the location of the rest of the distribution and make sure that the Apache user (nobody by default) has write permission in that directory. If you have problems with this procedure, please ask your system administrator for help.

5.2 Instructions and Sample Run

To start using the system for the first time, you simply go to the cgi-bin URL of the protospider.py file. This will present to you with the startup screen. This screen has two sections. The first, at the top of the page, lists any and all names used to refer to previously used profiles. If there are none yet, it says so. Just below this is a text box for entering new profile names. The name isn't used by the system in any way except for saving the profile. It must be unique and should be relevant to the target for which the topic is developed. The screenshot below depicts a session where the user has entered in a new profile name for 'Linux and AI' and just needs to hit the 'Submit' button. Once the system has been used, then you have the option of choosing a profile that has already been started. To do this simply click on the button marked with the name of the profile you wish to use. The button marked 'Python' in the screenshot below is an example of this button.



The next page presented is the querying page. From this page you can take two courses of action. You can enter a set of keywords and click on one of the buttons denoting one of the four search engines supported at this time. Your choices are: Webcrawler[1], Google[2], AltaVista[3] and Infoseek[4]. You can use a subset of the syntax that is supported by all these search engines. The table below outlines the operators you can use in conjunction with the keywords, along with their meaning.

[1] http://www.webcrawler.com/
[2] http://google.stanford.edu/
[3] http://www.altavista.digital.com
[4] http://www.infoseek.com

Operator	Meaning	Example
+	The page must contain this word.	+Linux
-	The page must not contain this word.	-Microsoft
“ ”	Denotes a phrase.	“Artificial Intelligence”

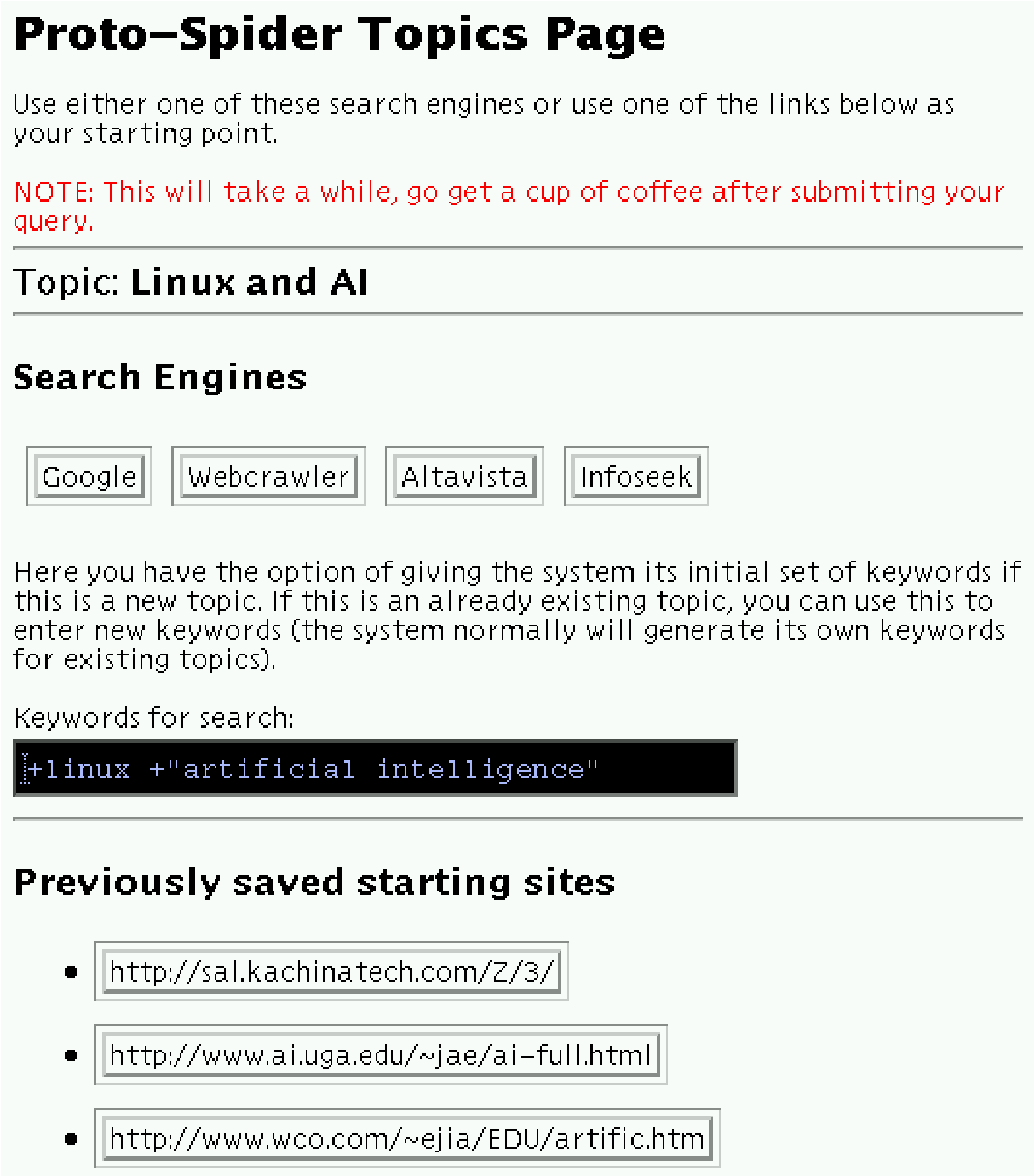
The quotes can be used in conjunction with the plus or minus signs. And any query can be made up of any combination of words and these operators (eg. +linux +”artificial intelligence”).

The second way to start a query can only be used with an previously developed profile. During the scoring step (below) one of the options is to save the page as a future starting point instead of using the search engines. If you have done this, these URLs will be listed below the search engine mechanism. To initiate a search using one of these saved URLs, simply click on it and the agents will use that page just as they use the page returned by the search engines.

Continuing the example session started above, the screenshot below shows the querying page. To better show all the parts of this page, this screenshot is taken from an already established version of the profile, where the user has already queried and scored a couple sets of pages. The topic selected by the user is shown near the top of the page, with the buttons and text box for the search engine queries below that. At the bottom of the page are listed the URLs saved from previous runs with this profile. If this were a freshly created profile, it would show a note saying that there aren’t any saved URLs saved yet instead of this list. In the example screenshot, the user has entered into the text box the keyword phrase

+linux +”artificial intelligence”

and only needs to select a search engine from those offered to start up the system.



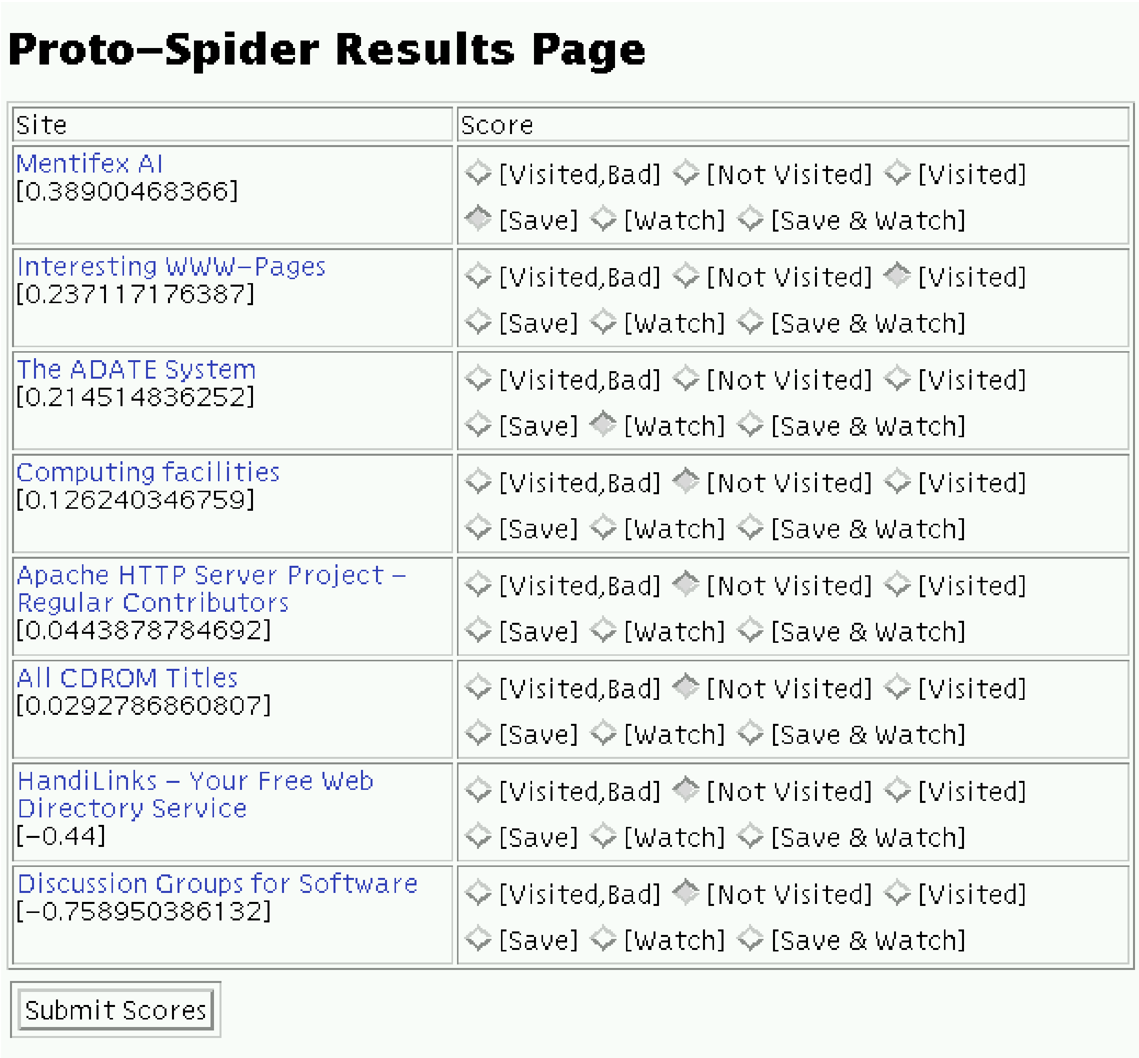
Once a query has been submitted, the agents go to work fetching and scoring five of the pages linked to either from the page returned by the search engine or those links on a saved page. If this is the first time using the profile the pages will not be scored (they'll be marked as 'Not Scored'), otherwise they'll be listed in the order of their scores, highest to lowest. Once the URLs have been analyzed and returned to the user, the user must then score the pages based on how appropriate they are to the topic. In this demo, the scoring is explicit. That is, the user is presented with five options and their responses have correlating scores. The options are not listed by the amount they would score the pages, instead they are listed by their eventual function[5]. For example, one of the users options is to save the page as a reference for further exploration (as mentioned above). This would tend to indicate that the page is related but that it doesn't contain a lot of information, it mainly consists of links, so it gets a low positive score. The options and their corresponding scores are listed in the table below.

Option	Score	Action*
Delete	-0.88	Deleted from list
Not Visited	-0.44	Nothing
Visited	0.0	Nothing
Save	0.33	Saved for future use as starting point
Watch	0.66	Site is monitored for updates
Save & Watch	0.99	Both Save and Watch

* Only the Save function is implemented in the demo. The others are only used for scoring.

In this last screenshot we see the results page containing the list of links selected by the agents off the search engines site. The link is referred to by either the page's title or the URL. As this screenshot is also taken from an already established profile, the links have been analyzed by the agents and each has a rating displayed just below the links text. You'll also notice that they have been sorted by rating and are listed from highest to lowest. To the right of each link is the list of the possible scores the user can give a page. In the screenshot, the user has already visited all the links and given them appropriate scores. All that is left to do is to hit the 'Submit' button which returns the user back to the starting page.

[5] Their function as it will be implemented in a later version of this system.



Proto–Spider Results Page

Site	Score
Mentifex AI [0.38900468366]	[Visited,Bad] [Not Visited] [Visited] [Save] [Watch] [Save & Watch]
Interesting WWW–Pages [0.237117176387]	[Visited,Bad] [Not Visited] [Visited] [Save] [Watch] [Save & Watch]
The ADATE System [0.214514836252]	[Visited,Bad] [Not Visited] [Visited] [Save] [Watch] [Save & Watch]
Computing facilities [0.126240346759]	[Visited,Bad] [Not Visited] [Visited] [Save] [Watch] [Save & Watch]
Apache HTTP Server Project – Regular Contributors [0.0443878784692]	[Visited,Bad] [Not Visited] [Visited] [Save] [Watch] [Save & Watch]
All CDROM Titles [0.0292786860807]	[Visited,Bad] [Not Visited] [Visited] [Save] [Watch] [Save & Watch]
HandiLinks – Your Free Web Directory Service [–0.44]	[Visited,Bad] [Not Visited] [Visited] [Save] [Watch] [Save & Watch]
Discussion Groups for Software [–0.758950386132]	[Visited,Bad] [Not Visited] [Visited] [Save] [Watch] [Save & Watch]

Submit Scores

5.3 Results

To generate the results, the system was used to query and score eighty seven web pages on two different topics. Fifty eight pages on the topic of 'Linux and Artificial Intelligence' and twenty nine on 'Python' (the programming language, not the snake). These numbers were used so that the Age of the profiles would be high enough to have an impact on the profile. The 'Python' profile is kept just long enough for it to get old enough for low scoring keywords and connections to be removed. The number of pages for the 'Linux and AI' topic is also based on this, only it is old enough to go through two such cycles, 0.75 is attained when the 23 web pages have been analyzed, and then again at 45.

At the end of this process the ratings for the analyzed pages, those returned on the results page, have gained a respectable accuracy. When using the 'Linux and AI' profile, pages that return a score above 0.2 seemed to be very close to the objective. Of course this is hard to quantify as it is inherently a subjective process, but I found that the profiles showed fairly regular improvement as they grew in age. The early ratings were very erratic, tending toward the extremes, and showed only a very superficial resemblance to the desired results. But as the amount of feedback increased, the ratings increased in accuracy fairly quickly. After around twenty pages the ratings reflected the general preferences expressed by the profile, with positively rated pages generally being of interest and negative rated pages not. Though there were still disparate ratings and the positively rated pages were not quite ranked correctly, ie. sometimes the pages that seemed most interesting were not rated the highest.

The straight numbers that were arrived at after the sample run are listed in the table below. The Age is the number of pages that were figured into the profile, the Keywords, Near Connections and Next_To Connections are the respective number of such elements that occurred within the network profile.

Profile Name	Age	Keywords	Near Connections	Next_To Connections
Python	26	99	355	107
Linux and AI	48	256	1529	384

These numbers do contain some interesting information, even if the matter of how well the profile works is a subjective measure. It would seem on the face that there is something of a pattern starting here, with keywords and next_to connections growing together and the near connections growing much faster. While this would seem to be a fairly good, and intuitive, general rule, it is doubtful that this is a hard and fast rule. The nature of web pages make it difficult to judge the results in this way. For instance, in going through the sample run for the 'Linux and AI' profile, one of the pages scored was a very large (over 140 kilobytes) file, while there were no such large files in the 'Python' profile's run. A large file like this can add an enormous number of keywords and connections, and this page alone is probably responsible for a good portion of the 'Linux and AI' profile's Near Connections.

This leads into the one major problem with this system. That is, it is possible for there to be many keywords and connections in a profile that have only shown up on one web page. If you will take a moment to look over the raw data, the actual keywords and connections, in Appendix B you will notice many instances where the scores are simply the score given to a page via the results page. Particularly you will see many instances with scores of 0.66 or -0.44, as these were the two most common scores given. This might denote a need for a more varied form of scoring, or other means of limiting the number of keywords and connections stored in the semantic network. Please refer to the Section #, Future Directions', for a discussion of possible ways of dealing with this problem.

Chapter 6

Conclusion and Future Directions

6.1 Conclusion

This system was designed to be an agent based, lightweight, non-redundant, adaptive topical web filter meant to expand upon the simple ideas of metacrawlers. The goal was to do this without becoming bogged down in the computationally expensive and complex technologies that have been applied to this domain. In this meeting this goal a multi-agent system was created that combined the ideas of the simple keyword based filtering system with a syntax based semantic network. This system performed quite well, adapting quickly and rating the pages in a regular and useful manner. There was only one significant problem that was discovered during the running of the prototype, and it proves to be quite valuable in pointing to new directions that not only address this problem, but expand the adaptive abilities of the system.

6.2 Future Directions

Score Degradation This idea is to slowly subtract from the scores of the keywords and connections in the semantic networks. This would help address the problem of the network getting overly bloated with words that have only a limited relation to the topic. It would allow the system to become better at rating the pages by eliminating these words and thus being more focused on the topic. It also allows for some interesting new ways of introducing other new ideas into the system, like genetic operators (below).

Genetic Operators If the scores of the system degrade over time, this allows for the introduction of some new, possibly useless, keywords or connections into the semantic network. One interesting way of introducing new connections, is to use genetic operators, specifically crossover and mutation. Basically how this would work would be that each time the network was updated, there would be a slight change of one of the genetic operators acting. Crossover would work by picking two connections at random, taking one word from each and creating a new connection with them (keeping the old connections). Mutation could work in a similar manner, randomly selecting a connection, making a copy of it, then replacing one of the words in that connection with one that has been randomly selected from the keyword database.

Content Analysis This is a simple idea for improving the rating system by analyzing a web page on a structural basis. For instance, if a web page has a total of 500 words on it, and 450 of those words occur in links. Then the page is probably a page of links without much real informational content. Conversely, a page with 500 words and 10 links has a high informational content level. This and other information like this can be used to help generate a more appropriate rating for a page than just keywords alone can provide.

Keyword Scores It might be useful to use the scores as generated by the html parser for something. At this point they are strictly used to determine if the word is an important keyword on the page, then they are discarded. These scores might be useful to be kept around as either an additional score or another number to calculate into the semantic networks. It could feasibly help get the network to maintain a smaller, more focus set of keywords and relations by highlighting the more important keywords.

PyBots It is also a continuing goal to beef up the underlaying agent system. To make it multi-threaded and give it better error handling. The implementation of the prototype also illuminated a few weak points in the design of PyBots that will be re-implemented in the future.

Appendix

A. Agent-Based System Architectural Issues

Introduction

The underlying infrastructure that this project is built upon is often referred to as an agent-based system This basically translates to a system built out of agents. Where the basic building block of the system is an agent, much like an object is in object-oriented systems. In this appendix I will briefly go over PyBots, a project to develop a complete agent-based development environment. The system is not complete, and the topical web spider discussed in this document is its first prototype application.

Why Python?

PyBots is implemented in the Python programming language. For a brief introduction to the language I swipe an excerpt from Python's web page [1]:

> Python is an interpreted, interactive, object-oriented programming language. It combines remarkable power with very clear syntax. It has modules, classes, exceptions, very high level dynamic data types, and dynamic typing.

Python boasts a lot of advanced features that make it a good choice for nearly any application, but I'll limit my comments to the reasons it worked so well for this project. There are two main reasons for my choice of Python. First is that it is both a very simple and a very powerful language. It allows for code to be written in an easily understood form, while allowing for programming on a large scale through modules and objects. It's simple syntax, high level of expression and interpreted nature allow for programs to be developed and written an order of magnitude faster than with most other traditional languages.

The second reason is because of its extensive package libraries. For just about any task, particularly in dealing with the internet, Python probably has a library package to help you implement that task. The library has packages for everything from database interfaces, to windowing systems,

[1] http://www.python.org/

to CGI controls. And since most of these modules are built around the concept of objects, they can be extended and customized in very powerful ways quite easily.

What is an Agent

The term 'agent' is not very well defined in the industry today. It has been used to reference to everything from spelling checkers to robotic control programs. From the users perspective, dealing with a software agent should seem more like dealing with a real world agent. Technically speaking this has certain ramifications. An agent will commonly have certain characteristics, having all of them is not necessary, but most agents will at least of a few of these.

- Autonomous - an agent should be much less dependent on direct manipulation. It should run much of the time with minimum supervision dealing with the user when it reports something, needs additional info, etc.
- Goal Oriented - an agent will have a goal or goals it is trying to achieve goals, and can deal with its environment to achieve those goals. It could be proactive, acting without user input.
- Proactive - the agent can take the initiative and act without user prompting.
- Environment Sensing - the agent can sense and interact with its environment. With software agents this translates into the networks, filesystem, OS, etc.
- Adaptive/Learning - the agent can adapt to it's role. For example, a mail filtering agent should adapt to its users reading habits.
- Social/Communication - the agent doesn't live in a vacuum. It can communicate with other agents and collaborate with them.
- Mobile or Static - and agent can either be mobile or static. A mobile agent can send it's code and data to another machine and resume its task on that machine. While a static agent stays on one machine and communicates to other machines.

Agent-Based Programming

For a development environment to be called agent-based is to say that its basic building blocks are agents. Much like an object-oriented environments' (eg. Python) basic building blocks are objects. Agents, at the code level, are implemented as objects, though they have certain differences in how they are used. The primary differences are in communication and hierarchy. These topics will be covered in more depth below.

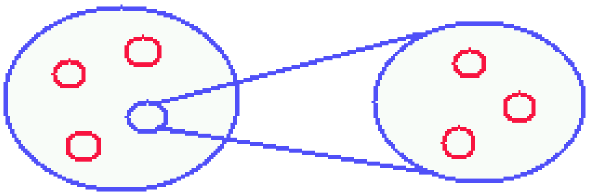
The real difference between agent-based development and object-oriented or, to a lesser extent, component based development, is not with the code however, it is in the metaphor. The metaphor in object-oriented development is an object. Objects have properties and uses (eg. scissors are metal and can be used to cut), so code objects have properties (members) and uses (methods). In a similar manner, the metaphor in agent-based development is an agent (surprise!). Agents in the real world act, have knowledge, communicates to other agents, etc. So an agent, in the code sense, should have these same characteristics.

Agent Types

While this project will use agents of many different types, they will always fall into one of two broad categories.

- Agent Object (AO) - An Agent Object is the term I will use for the simple agents that are the core of this system. They are the corollary to objects in object-oriented programming. They are agents that can only do one thing, and watch for information about when and how they are to do their thing. Their knowledge consists mainly of knowing what states they can transform, and remembering what they need to know about the environment to act (though this is mainly done in a distributed way, via the black board). AO's can respond to requests directed to them as well as make requests of other agents.

- Menski Agent - Menski agents are the term which is used for a sort of meta-agent. An agent who's primary purpose is to be a wrapper for other agents. The menski agent's main use will be to encapsulate the complex workings of many AO's (which I will refer to as the menski's collective). Thus a menski agent and its collective, which can include menski agents, can be treated just like any other AO. In this way they can be used to recursively develop more and more complex agents [see diagram below]. While basically looking like other AO's, menski agents have several specialized tasks. They maintain a black board to allow their encapsulated agents to communicate amongst themselves, handles passing this communication on to outside agents when appropriate, and passing outside communications in.



AGENT COMMUNICATION

Unlike object messaging, agents have a more generalized standard communication protocol. They don't send messages directly to one another, instead they use a special agent language and communicate via a set of black boards.

Black Boards (BB) The agents communicate via black boards. A black board is simply a data space which agents watch for messages and write messages. Each menski agent will manage its own BB and maintains a pointer to the BB of the menski agent above it. This allows for each menski agent to have it's own messaging space, while allowing it to maintain communication with the rest of the system.

Each BB will primarily be used for the communication of the agents comprising its collective. It will maintain state information between the agents of that menski agent (the menski agent's state), and hold the requests and answers to and from other agents. The BB's are structured so as to maintain a connection to the BB's above and below it, ie. the menski agent's collective it is a part of and the menski agent's in its collective respectively. A menski agent's collective, and thus it's BB, will usually serve one of two purposes. Its collective will either be comprised of a number of AO's in which case the BB will hold the requests relating to those agents or state information the collective needs to remember or it will be used as a communications conduit, when its collective is made up of other menski agents. Though it is likely that there will be shades between these two.

Vocabulary This system derives its vocabulary from a subset of speech act theory. It is purposefully as simple as possible at this point, with a mind for future expansion as needed. The vocabulary consists of the following words; request, answer, inform and state.

Semantics

- request - A query addressed to a particular agent by name. When an agent accepts a request, it is taken off the BB. Requests are sent to both other agents in that menski agent's collective and to other collectives above the encompassing menski agent.

- answer - A responce to a request. Addressed to a specific agent (the one that sent the request) by id. They are taken off the BB when accepted. It can range from just an 'ok' to a detailed report.

- inform - An message sent to a particular agent to pass on information of some sort. When accepted it is taken off the BB. Informs are sent to both other agents in that menski agent's collective and to other collectives above the encompassing menski agent.

- state -Descriptors of the current state of the agent. Used to maintain information internal to a particular menski agent and its collective. They will never be sent outside the menski agent from which their sender originates.

Syntax

- The syntax of messaging is also quite simple at this point. Each message consists of a simple dictionary (known as an associative array in perl or collection in visual basic) containing system information about the sender and recipient, along with the data content the message was initiated for.