IMPLEMENTATION OF AN INTELLIGENT AGENT TO COMPILE A CUSTOM ELECTRONIC PERIODICAL

A Senior Thesis

By

Michael Nichols

1996-97 University Undergraduate Research Fellow

Texas A&M University

Group: ELECTRICAL ENGINEERING/COMPUTER SCIENCE

Implementation of an Intelligent Agent to Compile a Custom Electronic Periodical

Michael Nichols
University Undergraduate Fellow, 1996-1997
Texas A&M University
Department of Electrical Engineering

Approved

Fellows Advisor A. L. Narasimha Reddy

Honors Director [illegible]

Implementation of an Intelligent Agent to Compile a Custom Electronic Periodical. Michael Nichols (Narasimha Reddy), Electrical Engineering, Texas A&M University

As the result of the creation of advanced information technology within the past two decades, we are entering into a new era of civilization. In this new day and age, where large industry was once the dominant force in our Western civilization, individuals are now finding themselves empowered through information technology in ways they never dreamed possible. Services such as telephony are now being replaced by software tools that make possible all types of information transmission and retrieval via the worldwide Internet network. One service in particular will change as the print media gives way to electronic publishing, and that is the newspaper. While news services will certainly continue to exist, the reader of electronic media will have a much greater selection of stories at his fingertips. Our program, described herein, is one such tool which will allow a user to locate and extract desired information from a wide variety of sources and create a personalized journal that could serve as the user's newspaper or perhaps as a professional or technical journal, depending on the user's preferences.

Introduction

As the worldwide Internet computer network expands to serve a greater and more diverse audience of users, information resources once limited to the medium of printing will begin a transition into the digital medium, much as our oral literary tradition of days past became a written one centuries ago. Already a part of this transition process are periodicals such as the *Wall Street Journal* and the *Houston Chronicle*. Their presence, and that of other periodicals appearing on the World Wide Web, gives a foretaste of the days to come, when all media forms will become available from our home computers, and the amount of information available to us from these machines will far surpass our human ability to sort through it.

With the speed and efficiency of computer technology, however, it is not only possible for us to archive such large amounts of information, but it is also possible to provide technology for locating and identifying information desired for human use. One such application of this information-retrieval technology in compiling a periodically-updated journal of information on desired subjects based solely upon the user's topical and Internet-locational preferences. The purpose of the project described in this paper is to implement a computer program that will perform just such a searching operation. Our project will take a list of starting locations on the World Wide Web and a list of topical keywords, and compile a journal from what it finds, extracting individual articles from web pages as necessary (a single page can have multiple articles), and saving them to the user's hard drive for later perusal. Throughout the past year, I have conducted research into various schemes for conducting such a search, and have begun implementing a rudimentary version of such a program.

System Architecture and Implementation Scheme

Languages

Currently, I am implementing my portion of the project in a combination of three languages to run on the Unix platform: C, Perl, and Java. The simultaneous use of these languages is made possible by Java's use of native class methods within its object-oriented framework. It is possible to compile C functions into a shared library that is then loaded by the Java compiler at runtime. Because Perl is written in C and has an interface for communicating with C (by linking the Perl interpreter into the C code), it is also possible to create a Perl shared library, which in turn becomes a set of native methods for a Java class, thereby fusing the three languages into one program.

The reason for this multilingual programming style becomes obvious when one examines the strengths of each of these languages in relation to the needs of a program such as this. Because our software deals primarily with text-based data, a good platform for text-processing is essential. Perl provides just such a platform—as does C for text-parsing, when combined with the Lex lexical analyzer generator (or the GNU flex utility provided with Linux, which has been the standard lexer generator for the project). C and Perl, however, are really poor choices for World Wide Web programming, however, because both of them require the programmer to write code at the “Socket” level—they have no built-in capability for accessing the World Wide Web. Java, however, provides this capability and the convenience of an object-oriented environment, so I have chosen it as the main language of implementation for the project.

Basic Program Architecture

Language issues aside, it is necessary to examine of the basic structure of the program itself. Although the implementation is object-oriented in nature, I will try here to describe the program in more procedural terms for clarity purposes, as there is a distinct series of steps involved in the searching process. At first, however, a brief description of the major data structures is in order.

Internal to the program, there are three primary permanent data structures. These consist of a queue, a hash table, and a vector (Java abstraction of a linked list or array). The queue is used to store a list of pages to be retrieved by the program, as well as some other information, including whether an entire page is to extracted or simply individual articles, and at what level of the page hierarchy the program is currently searching. The hash table contains a list of files retrieved from the network and a Boolean flag for each file, denoting whether the file should be disposed of at the end of the search process. The vector (called the "goodies") contains a list of the extracted articles and pages that will later be used to create a table of contents and to organize local links to material located on the hard drive.

External to the program is a cache directory of extracted files that doubles as a scratch area during program execution. Also external to the program is a file describing the contents of the cache (to be derived from the hash table) and a table preserved at the end of the search, and skip to step 8.

5. Run the Article Extraction Unit class on the retrieved file to extract the desired articles (this portion of the program will be described later in the document).

6. Store the filenames of the articles extracted in the goodies list. These filenames will all be prefixed with the letters "AEU."

7. Find all anchors (hyperlinks) located inside the extracted articles. Store the absolute net addresses of these links in the queue, setting them to the same level of search as the current retrieved page and also setting them for full-page extraction.

8. Find all anchors within the original copy of the current retrieved page (regardless of whether they are contained in extracted articles or not) and store their absolute net addresses in the queue, setting them to the next deepest level of search, and also setting them to article extraction mode. Skip this step if the next deepest level of search exceeds the maximum allowed level.

9. Repeat the process from step 1 until the queue is completely empty.

10. Take each file in the goodies list, and examine its anchors. If the anchors point to information known to be on the hard drive (from examining the hash), the anchors are set to point to the local data. If the anchors point to information not recorded on the hard drive, the anchors are set to point to the absolute net address of the desired data. If images are present in the document, then the hash table is first checked to see if the desired image(s)

are located on the hard drive. If so, the anchors are set accordingly. If not, the image(s) are downloaded from the network, the hash table updated, and the document anchors set accordingly. All images and documents pointed to by "goodies" are updated in the hash table to specify that these documents and images should be kept.

11. Find all "AEU" files in the goodies list and create an HTML table of contents file for these documents.
12. Expunge all documents marked for deletion in the hash table.
13. Catalog all remaining files and write to disk.

Article Extraction Unit

The Article Extraction Unit is the heart of the program, and is the main research focus in this project. The AEU performs two primary tasks simultaneously, namely, keyword searching and separation of a single page into multiple articles, extracting only the desired ones. The AEU follows a very basic sequence of instructions to locate and extract articles. The procedure is as follows:

1. The AEU's lexical analyzer locates and identifies the HTML tags (markups) within the document and generates a list of these tags to be used in other portions of the program.
2. The AEU reads (starting at the beginning of the document) a block of text set off by separator tags (headings, rule lines, tables, etc.) and performs a keyword search. If the block is a match, then it stores the block in memory

and goes on to the next block. If it is not, step 2 is repeated.

3. If the next block also matches, then this block is affixed to the end of the last block stored in memory, and step 3 is repeated. Otherwise, the block(s) stored in memory are written to disk as an article, with a unique filename starting with the letters "AEU."

This procedure continues until the AEU reaches the end of the file it is reading.

The Need for Advanced Searching

A careful consideration of the problem faced by such a program makes it clear that something more advanced than a standard keyword search in necessary to find all relevant documents corresponding to a particular query. Query constraints in a journal-type program are very vague. Users of such a program will find themselves asking for information about broad topics such as "basketball," or using language to formulate their queries which does not fully define a topic in the eyes of the computer, which sees only keywords and which normally cannot find one word to be as a perfect substitution for a topical concept, as it does not truly understand what it itself is doing.

There is a further complication in searching in that oftentimes, especially within news articles about standard topics, such as sports, the word which represents the topic itself (e.g., basketball) is nowhere within any of the desired articles to be found. It becomes necessary, therefore, to be able to replace a topic-defining word, such as basketball, with a set of query keywords, the closest thing to a conceptual unit within the computer's memory. A scheme for performing such a task should involve a machine acquisition of such words and should not be dependent on user knowledge for formulating that list, aside from the computer's perception of the user's likes and dislikes. In other words, the burden of defining keywords to associate with a given topic should be on the computer. However, to assist the computer in this task, user feedback to the computer about the relevancy of retrieved articles can and should be obtained. The basis for this sort of technology lies in the branch of computer science known as information retrieval. In the next section, I will describe in detail some of the important elements of this technology.

An Introduction to Information Retrieval

A careful examination of the information retrieval literature reveals that much of the important research in the field was conducted before 1980. Indeed, the definitive textbook on the subject is in its second edition, dated 1979.[3] The reason for this is that the early applications of this sort of technology were both impractical from a commercial standpoint and disappointing from a performance standpoint. Recent advances in hardware and software technology, however, make such techniques more practical, and there has been somewhat of a resurgence of interest in the field, especially due to the popularity of the World Wide Web.

The central idea behind IR theory is that documents can be located by matching sets of keywords or index terms. In the simplest mathematical form, the degree of match between a two sets of index terms is:

$$|X \cap Y|$$

where X is one set of index terms and Y is the other.[3] The level of document relevancy, therefore is a function of the number of matches of index terms between a document and the query to which it is compared. More sophisticated relevance matches include Dice's coefficient:

$$2\frac{|X \cap Y|}{|X| \cdot |Y|}$$

Jaccard's coefficient:

$$\frac{|X \cap Y|}{|X \cup Y|}$$

and the Cosine coefficient

$$\frac{|X \cap Y|}{\sqrt{|X|} \times \sqrt{|Y|}}$$

When these matching functions are used to rank search results in decreasing order of relevance, as with the Lycos search engine, it is called linear retrieval.[2,5]

Obtaining a list of index terms from a query is, of course, trivial, but the process of obtaining index terms from a document in another story. Not all words in a document pertain to the subject. Empirical studies show that "function words," or words that do not carry a meaning specific to the subject being discussed in a document tend to follow a Poisson distribution over all documents, whereas "specialty words" do not.[3] Another simpler, but still useful model of word relevance, called information content, comes from information theory and is given by the following equation:[2]

$$INFO(w) = -\log_2(P(w))$$

Where w is a word or other lexical entity, and P(w) is the probability that a given word or lexical entity in the particular textual corpus being examined is the lexical entity or word in question. Clearly, this is a more easily computed metric than the Poisson distribution function.

Taking this idea of document and query matching one step further, we can make similarity comparisons between documents as well. One can easily postulate that documents that are similar to each other in terms of index terms will also be similar in terms of content. By comparing documents to other documents, it is possible to cluster documents into different subject areas.

One standard algorithm for producing clusters is called Hierarchical Agglomerative Clustering. The algorithm is rather simple, and works as follows.[2] Given a set Ω of objects to be clustered, the algorithm starts with the subset of $\wp(\Omega)$, the power set of omega, consisting of only the singleton elements thereof. The next step is followed iteratively until there exists only one cluster in the set: the two clusters that are the most similar by one of the previous formulae are combined into a single cluster. The time complexity of this algorithm is asymptotically $O(n^2 \log n)$, where $n = |\Omega|$, not an impractical rate on today's machines.

GURU

GURU is an information retrieval program written by Dr. Yoelle Maarek. GURU uses automatic keyword classification techniques to categorize software components for reuse, a topic of much interest in the field of software engineering. GURU classifies software components by examining their documentation for keyword content, and employs the HAC algorithm as described above.

As document index terms, rather than using individual words, GURU uses groups of two words each known as lexical affinities. Lexical affinities are, theoretically, pairs of words functioning in a modifier-modified relationship. However, because the parsing required to determine a precise LA relationship is very involved, an approximation is used, based upon the fact that 98% of LA's relate words that are within a span of five words. GURU searches a sliding window over a document for pairs of open-class (high information-content) words to use as LA's.

GURU then determines the information content of each LA. This information content figure is multiplied by the number of occurrences of that LA in the particular document in question. This figure is known as the resolving power of the LA and is represented by the Greek letter ρ. Each resolving power is then normalized with the other ρ-values to obtain a "z-score." A threshold z-score value is predetermined and only those LA's whose z-scores exceed this threshold value are retained as index terms for the document.

GURU then uses this information to form clusters using the aforementioned

HAC algorithm. This hierarchy can then be searched. GURU's search algorithm works like this: The user formulates a search query according to the "authorized vocabulary" (GURU is restricted to particular search terms). Then GURU attempts a straight linear retrieval. If that is unsuccessful, GURU starts to retrieve documents based on cluster membership.

GURU, although it provides fully automatic classification, still has some disadvantages. The most noticeable is its ignorance of synonymity of terms, which limits the effectiveness of freely-constructed queries, since the exact choice of synonym can mean the difference between retrieving the desired information or not. A "simple" solution to this problem is to provide a thesaurus to make appropriate substitutions for keywords. Studies conducted in the late 1960s and early 1970s by Karen Sparck-Jones of Cambridge University and others show, however, that automatic thesaurus generation is impractical for a search engine such as GURU, where a query is formulated and executed only once..[4] In a World Wide Web periodical generation program, however, a thesaurus may be more practically generated using a repetitive machine learning technique.

Latent Semantic Indexing

In the late 1980s a new development in IR technology emerged. A group of researchers associated with Bell Labs developed what is known as latent semantic indexing.[6] LSI is a radical departure from previous information retrieval systems in that it dispenses with the traditional notion of searching a document index entirely. Instead, LSI maps documents and query terms into an n-dimension vector space (where n is usually around 200 or so). Documents are selected from queries by forming a query vector as the sum of vectors corresponding to the various index terms in the query. Inner products of this vector and document vectors within the vector space are then taken, and the relevant documents are the ones whose vectors form the smallest angles with the query.

This form of search technology is important in that, if enough documents are available to catalog, a thesaurus is less necessary to match index terms, since LSI implements a fuzzy search. LSI is also very fast. A variant of the algorithm (apparently not covered under Bell Laboratory's patent) is used in the Excite World Wide Web search engine.[7] LSI is important in the context of this project in that it provides a fuzzy search and does so by geometric means, but it is clear that the pure LSI algorithm, which maps words to documents is less applicable for the type of searching to be done by an automatic web periodical generation program, as it assumes prior knowledge of the documents to be searched from. This is not the case in periodical generation, as the idea is to search for new documents that have not yet been indexed.

Formal Concept Analysis

Formal concept analysis is a part of the mathematical subject known as lattice theory, which in turn is the study of certain types of partially-ordered relations.[10] Formal concept analysis, or *Begriffsanalyse* as it is known in German, was developed by the German mathematician Rudolf Wille at the Technische Hochschule Darmstadt. Formal concept analysis is concerned with the organization of a hierarchy of conceptual information.

A concept, from a philosophical and a mathematical point of view, is defined by its extent and its intent. The extent of a mathematical concept consists of the set of objects which belong to the concept, and the intent consists of the set of attributes attributable to those objects. So a concept, then, is an ordered pair (A,B), where A is the extent and B is the intent.

The example used in Davey and Priestley's book on lattice theory uses planets to demonstrate the functionality of a concept. One concept is that of the properties of planet Earth. The planet Earth has (under this abstraction) three attributes. Therefore the intent B={size-small, distance-near, moon-yes}. If we let A be the set of all planets that have the attributes of B, then A={Earth, Mars}. The complete concept is (A,B) or ({Earth,Mars}, {size-small, distance-near, moon-yes}).

Concepts are ordered according to the following formula:

$$(A_1, B_1) \leq (A_2, B_2) \leftrightarrow A_1 \subseteq A_2$$

where a "greater" concept is a more general one. It can easily be seen, then, that this is a convenient way of represent conceptual information, especially when it is necessary to distinguish various levels of conceptual generality.

A context is a triple of the form (O,A,R), where O is a set of objects, A a set of attributes, and R a binary relation between the two ($R \subseteq O \times A$). If an ordered pair $(o,a) \in R$, this phenomenon is usually notated oRa, and it means that the object o has the attribute a. The set of all concepts within the context (O,A,R) is notated $\mathcal{B}$(O,A,R), the letter B coming from the German word for concept, Begriff. This set can be calculated using Ganter's algorithm.[9] According to the

Fundamental Theorem of Concept Analysis, the partially ordered set $\langle \mathcal{B}(O,A,R);\leq \rangle$ is a lattice, and is therefore able to be analyzed with lattice theory.[10] This lattice, being a partially-ordered set, is a hierarchy of the concepts stored within it, and easily searchable as the next section describes.

Concept Analysis for Information Retrieval

To understand how component retrieval from a concept lattice is possible, it is helpful to establish some definitions.[9] Firstly, given $A \subseteq \mathcal{A}$ and $O \subseteq \mathcal{O}$ from context $(\mathcal{O},\mathcal{A},\mathcal{R})$, the common attributes $\omega(O) \equiv \{a \in \mathcal{A} \mid \forall a \in A: (o,a) \in \mathcal{R}\}$ and the common objects $\alpha(A) \equiv \{o \in \mathcal{O} \mid \forall a \in A: (o,a) \in \mathcal{R}\}$. Secondly, given a concept $c=(O,A)$ from a context $(\mathcal{O},\mathcal{A},\mathcal{R})$, the extent of c is $\pi_o(c) \equiv O$ and the intent is $\pi_a(c) \equiv A$. $\mu(a)$, then for some $a \in \mathcal{A}$ is the greatest concept c for which $a \in \pi_a(c)$.

Infimum and supremum for a concept lattice are given, respectively, by: $\bigwedge_{i \in I}(O_i,A_i)=(\bigcap_{i \in I} O_i, \omega(\alpha(\bigcup_{i \in I} A_i)))$ and $\bigvee_{i \in I}(O_i,A_i)=(\alpha(\omega(\bigcup_{i \in I} O_i)), \bigcap_{i \in I} A_i)$.

With these definitions, we can go on to define a query to a concept lattice $\langle \mathcal{B}(\mathcal{O},\mathcal{A},R);\leq \rangle$ as a set $A \subseteq \mathcal{A}$. An object $o \in \mathcal{O}$ satisfies query A, iff $A \subseteq \omega(o)$. The set of all objects which satisfy query A is called the result of A and is denoted $[A] \equiv \{o \mid o \in \mathcal{O}, A \subseteq \omega(o)\}$.

The result for any query can be found by applying the following theorem: the result of a query A to concept lattice $\langle \mathcal{B}(\mathcal{O},\mathcal{A},\mathcal{R});\leq \rangle$ is given by $[A] = \pi_o(\bigwedge_{a \in A} \mu(a))$. A set of significant keywords (that is, keywords which can narrow the search) is given by $\langle\langle A \rangle\rangle = (\bigcup_{o \in [A]} \omega(o)) \setminus \pi_a(\bigwedge_{a \in A} \mu(a))$. Any new narrower query can be processed incrementally from the results of the last query, thus saving execution time.

Like the clustering approach of Dr. Maarek, the concept analysis indexing scheme is hierarchical in nature. However, unlike the searching scheme employed in GURU, a concept analysis system, such as Christian Lindig's FOCS system uses the hierarchical nature of the concept lattice throughout the search process. To allow the user to narrow his search interactively; the computer uses the results of each query to determine a list of keywords which the user can use to narrow his search. Neither clustering nor latent semantic indexing allow this sort of interaction. Clustering is a last-resort search method, anyway, and cannot effectively take advantage of the interactive capabilities of concept-based searching.

Although a concept lattice is much larger data structure than a latent semantic indexing space, concept analysis does save memory over other indexing schemes. Because concepts are laid out in a hierarchy, it is not necessary to duplicate index terms for each concept within the lattice, as it is in clustering, where each document must have its own record of its index terms. In a concept lattice only information unique to a particular concept must be stored at a concept node, because the neighboring concepts provide the other pieces of information necessary to construct the full set of index terms (or to process a query). Also, both clustering and concept lattice construction are polynomial-time algorithms, so both are feasible from a time-perspective.

Information Retrieval Approach Taken

In examining the various schemes described above, I decided that a combination of the various approaches would lead to an effective solution to the problem. The traditional information retrieval technique of measuring information content of the words in a particular document gives a picture of which words might be useful for searching purposes. Concept analysis provides a method by which objects may be arranged in order of generality of content. A method combining these two approaches is in order.

My approach to the problem is as follows. Since the program is continually searching for given topics, it follows that continuous use of user feedback will eventually lead to computer t focus in on the exact preferences of the user. To employ this user feedback, it is necessary to examine previously retrieved documents to determine what distinguishes the relevant documents from the non-relevant ones. Since, intuitively, the relevant documents all show a preference for a particular concept, it follows that the concept analysis approach can be used to categorize the previously-retrieved documents.

Previously-retrieved documents are stored as the extents of concepts, and the keywords which the documents contain, as determined by information content analysis, are considered to be the intent. Relevancy information, as obtained from the user, is stored as well, although it is not used to order the lattice. The concept lattice thus obtained is a hierarchical representation of documents whose keyword sets define topics of various generalities.

At each concept in the lattice, a percentage of documents within that concept that are relevant is stored. The keyword set selection process, then, becomes the following: Find the most general concept in the lattice that has a relevancy percentage higher than some pre-determined threshold. The intent of that concept becomes the keyword set for the search. If the number of documents found to match the keyword set is too high, then narrow the search one level, by finding the next most specific concept in the lattice with the highest relevancy percentage. If the number of documents retrieved is too low, the query can be made more general by traversing up the lattice in the direction of highest relevancy percentage.

Conclusion

In the future, retrieval of information will be facilitated by the use of intelligent agents that will filter information for users. One application of such an agent is in generating custom periodical "publications" from information available on the Internet. In this paper, the development of one such program, one that is currently being implemented for the Unix platform, was discussed, and a historical context for its technology was provided.

Bibliography

1. Yoëlle S. Maarek. “Software library construction from an IR perspective.” *ACM SIGIR Forum,*Vol. 25, No. 2, Fall 1991, pp. 8-18
2. Yoëlle S. Maarek, Daniel M. Berry, and Gail E. Kaiser. “An Information Retrieval Approach for Automatically Constructing Software Libraries.” *IEEE Transactions on Software Engineering*, Vol. 17, No. 8, August 1991, pp.800-813
4. C. J. van Rijsbergen. *Information Retrieval*. 2nd ed. Stoneham, MA: Butterworths, 1979.
5. Karen Sparck-Jones. *Automatic Keyword Classification for Information Retrieval*. London: Butterworths, 1971.
6. http://www.lycos.com
7. Scott Deerwester, Susan T. Dumais, George T. Furnas, Thomas K. Landauer, and Richard Harshman. “Indexing by Latent Semantic Analysis.” *J. Amer. Soc. Inform. Sci.,* Vol. 41, No. 6, pp. 391-407, 1990.
8. http://www.excite.com

9. Christian Lindig. "Concept-Based Component Retrieval." Available from the author at the Technische Universität Brauschweig (lindig@ips.cs.tu-bs.de), 1995

10. B. A. Davey and H.A. Priestley. *Introduction to Lattices and Order*. Cambridge, UK: Cambridge University Press, 2nd. Edition, 1990, pp. 221-236.

11. Christian Lindig and Gregor Snelting. "Assessing Modular Structure of Legacy Code Based on Mathematical Concept Analysis." Available from the authors at the Technische Universität Braunschweig (lindig@ips.cs.tu-bs.de), 1996