Information Retrieval with a Knowledge Based System

A. Ultsch
ETH Zürich
Institut für Informatik
CH-8092 Zürich

Abstract

Information retrieval systems are used to search for specific information out of a great number of stored items. Such systems are usually designed for many users. With the advent of powerful workstations however, it is feasible to realize personal systems. A personal information retrieval system, that allows only online retrieval of documents, is unlikely to be used, since the effort to store the documents exceeds the benefit of having rapid access to one's own document collection.

Scientists working in particular areas get insights, they find worthwhile to keep track of. Some of these insights is knowledge about the objects of the working domain and can be represented as facts and rules that describe the object's properties. Knowledge based systems claim to be able to represent such knowledge and to draw conclusions from it.

In this paper the concepts of a knowledge based information retrieval system are presented. Facts in a knowledge base are used for the representation of (working domain dependent) knowledge as well as for retrieval. Rules are used to describe the retrieval of information and to infer new facts from the knowledge base.

Keywords: Knowledge representation, information retrieval, expert system shell.
1. Introduction

Information retrieval systems consist of a collection of documents and information structures used for the retrieval process. Such systems are usually meant for multiple users if not for public use in a library for example ([INSPEC]). But retrieval of documents is important in a personal framework too as Sauvain states in [SAU]: 'Almost everyone doing research, record keeping or analytical work builds up a collection of notes, records, letters, memoranda, papers, and the like. In time such collections get so large, that it is hard to retrieve information.' This strongly suggests the idea of a personal information system.

A personal information retrieval system however that does not more than online retrieval of documents (or references to documents) is unlikely to be used, since the effort to store the documents usually outweighs the benefit of having rapid access to one's own documents. Since working in a particular context leads to a certain way to organize one's domain and produces insights that are worthwhile to keep track of, a useful information retrieval system should be able to store this knowledge. It should be able to retrieve the built up knowledge and allow to use this information in the document retrieval process. In this paper we present an approach to a personal information retrieval system that is able to store general working domain dependant knowledge besides the classical information structures (thesauri) and present a way how to put the knowledge into use.

The knowledge based IR system (KIR) is a part of a single-user system for storing and retrieving full text documents and references, as well as doing inferences together with a comfortable user interface, a knowledge base and an update component. This system is called KOFIS (Knowledge Based Office Information System) and has been developed at the ETH [AEJU]. The system is written in MODULA-2 and Prolog and runs on the personal workstation Lilith [WIRT].

2. Knowledge Based Information Retrieval

In the sequel we explain our ideas about the knowledge based approach to information retrieval by presenting an exemplary work with a knowledge based system. As an illustrative example let's assume we are going to read and annotate parts of the article 'A Hash Code Method for Detecting and Correcting Spelling Errors' written by M.Mor and A.S.Fraenkel that
appeared in the Communications of the ACM ([MOR]). The full text is given in the appendix.

Documents in KIR are for example papers, notices, letters etc. which are stored in a document base. In order to describe the documents we add bibliographic facts (i.e. author, title etc.) into the system. We assign also an arbitrary identifier, the so called documentid, to this document in order to identify it uniquely. For the moment let 'ACM1282' be such a unique documentid for this document. In order to make the author, title and reference known to the system we write:

\[ \begin{align*}
\text{author}(\text{'Mor,M.'}, 'ACM1282'). \\
\text{author}(\text{'Fraenkel,A.S.'}, 'ACM1282'). \\
\text{title}(\text{'A Hash Code Method for Detecting and Correcting Spelling Errors'}, 'ACM1282'). \\
\text{in}(\text{'Communications of the ACM 12 (25) 1982 pp 935-937'}, 'ACM1282').
\end{align*} \]

These facts are stored in the knowledge base and can be used to retrieve the document directly. This retrieval is done using questions with variables (they begin with a capital letter) in Prolog notation. With such a question a Prolog interpreter, which is integrated in KIR, is asked to find a binding for the variables in the question. For example we retrieve the document via the authors' name by writing: \text{author}(\text{'Mor,M.'}, \text{WhatDocument}) and the system responds with: \text{WhatDocument} = \text{ACM1282}.

In order to retrieve the document via a description of it's content, we add descriptive facts to the system. This is the equivalent to indexing a document in classical information retrieval systems. From the title it is clear, that the paper deals with hash codes so we write: relevant('hash code', 'ACM1282').

With the meaning, that in this paper the phrase (term) 'hash code' is important. For simplicity reasons, we assume for the moment that the document base is empty besides just one other document, which is indexed by: relevant('binary search', 'KNUTH3'). Of course the document base is only in this state when a system is newly installed. Normally the document base is filled with documents and bibliographical facts are stored in the knowledge base.

We go on now by reading the first sentence of the first paragraph: "In a recent study on detecting and correcting spelling errors, Peterson[4] states that for locating words in a word list, '...a hash table approach would seem the most reasonable'."
A knowledge based system should allow to store the information (facts) gained from this sentence. For this purpose we use the knowledge base which is the place to store all facts from our area of discourse together with rules how to use this knowledge. Facts are formulated as relations among terms. The relation 'described' as mentioned above is an example for such a fact. For this sentence, the fact that a hash table is an implementation for the problem of locating word can be expressed like this: implements('hash code', 'locating of words'). This fact could also be represented as a named arc as shown in Figure 1.

As we read the first two paragraphs other facts could be extracted from the text (see the appendix for the original text):

- A spelling check uses somehow a method for locating words (concluded from the text)
- A spelling check should find single errors (concluded from the text)
- Single errors are either the insertion, deletion or transposition of one letter or the transposition of two adjacent letters (defined in the last sentences of the first paragraph)
- Significant words are defined as keywords in a query (second paragraph).
- Binary search implements the problem of location of words (last sentence of the first paragraph)

Figure 1: Graphical representation of a fact

In this way we could work with the whole article, extracting relevant knowledge, but let the above facts be enough for our example. In our notation we express these facts as shown below:

uses('spelling check', 'locating of words').
shouldfind('spelling check', 'single error').
isa('single error', 'insertion of one letter').
isa('single error', 'deletion of one letter').
isa('single error', 'substitution of one letter').
isa('single error', 'transposition of two adjacent letters').
isdefinedas('significant word', 'keyword in a query').
implements('binary search', 'locating of words').

Besides the facts we have stored in the knowledge base while reading the article, the knowledge base contains all facts we have gained before by our everyday work. Let's assume that our example knowledge base contains but the fact: implements('hash code', 'locating of words'). Figure 2 gives a graphical overview of the facts we have so far.

![Graphical representation of a part of a knowledge base](image)

**Figure 2:** Graphical representation of a part of a knowledge base

### 3. The retrieval of documents using retrieval rules

The retrieval process for documents is specified using if-then rules which state under what context what retrieval actions can be done. These rules-called retrieval rules - are stored also in the knowledge base and formulate the way how to find a document.
An exemplary set of retrieval rules is the following:

(1) relevant: for Term if relevant(Term,Document) then retrieve Document.

(2) implements: for Algorithm if implements(Algorithm,Problem) then use Problem.

(3) solution: for Problem if implements(Algorithm,Problem) then use Algorithm.

(4) shouldfind: for Errtype if shouldfind(Algorithm,Errtype) then use Algorithm.

(5) is_a: for Thing if isa(Thing,Otherthing) then use Otherthing.

Rule (1) means, that in order to find a document for a term, one has to see whether there are some documents that are relevant for this term. The rule named 'implements' (2) means, that if a term is the name of an algorithm to implement a specific problem, the name of the problem itself can contribute to the search for relevant documents. Rules (3), (4) and (5) are interpreted in the same way using the 'shouldfind' respectively the 'isa' facttypes.

Starting from the term 'binary search' in Figure 3, KIR proceeds as follows: By rule (1) 'KNUTH3' is a document directly relevant for the term. Rule (2) widens the query such that 'locating of words' is also used to answer the query. With this term, the rules (3) and (1) are applicable and 'ACM1282' is the next document found. For our small knowledge base this is also the last document for this query. The system works in such a way, that it always terminates and no duplicates for documents or terms are used in the derivation process. This is because sets for the query-terms and for the documents are used that are recursive enumerable in such a way, that each member contributes only once to the answer.

Queries can be not only single terms, but also expressions on terms using the boolean operators and, or and a binary negation called butnot. For efficiency reasons the queries are transformed to conjunctive normal form, simplified if possible, and the disjunctive factors are inserted in the query set.

4. Goals and termination

In certain applications it may be sufficient to get some instead of all documents, avoiding an exhaustive search over the whole knowledgebase.
For example a user might only be interested in documents that are immediately relevant to a given term. This means that only the rule named 'relevant' should be applied in order to answer a query.

To cope with this needs, so called control rules are added to KIR. Control rules have the syntax:

\[ \text{for } \text{<Goal>} \text{ select } \text{<Selection>} \text{ until } \text{<Termination>}. \]

Goals (in the example below: all, relevant, ten) determine the selection and termination process for the retrieval. For a given goal either all rules are applied if <Selection> is a variable. In the other case, only the rule with the given name is to be used when answering a query. The selected rules are applied until <Termination> holds or no more rule is applicable. Termination can be any predicate in Prolog syntax [CM] as well as certain builtin functions. Below are examples of control rules:

(1) toJind all select Rules until false.
(2) toJind relevant select relevant until false.
(3) to Jind ten select Rules until ndoc(N) and N = 10.

Control rule (3) for example counts the number of documents found so far and if this count yields 10 the retrieval process terminates.

5. Retrieval and inference of knowledge

The knowledgebase can also be used to retrieve working domain specific knowledge. Since the underlying unification and backtracking algorithm works with the same syntax and semantic as a Prolog interpreter it is possible to write special purpose predicates. Here is an example of such a predicate:

(1) whatis(X) :- (isa(X,Y);isa(Y,X)),writeln('A ',X,' is a ',Y).
(2) whatis(X) :- isdefinedas(X,Y),writeln('A',X,' is by definition a ',Y).

Rule (1) states, that a thing X is a thing Y, if they are connected by an 'isa' arc in our network, no matter in what direction. The second predicate treats definitions. With these rules KIR can derive all definitory facts we have stored:

- A single error is a insertion of one letter
- A single error is a deletion of one letter
- A single error is a substitution of one letter
- A single error is a transposition of two adjacent letters
- A insertion of one letter is a single error
A deletion of one letter is a single error
A substitution of one letter is a single error
A transposition of two adjacent letters is a single error
A significant word is by definition a keyword in a query.

Rules can not only be used to retrieve documents and facts, but also to gain new facts from the knowledge base via inference. Let's assume that for an area of discourse it is possible to state the following fact: 'If an Algorithm Y implements a problem Z and Z is used by a program X then one can conclude, that X could use Y' or in Prolog notation:

```
uses(X,Y):-implements(Y,Z),uses(X,Z).
```

Rules like this are called inference rules and are also stored in the knowledge base. Note that this definition is recursive and only terminates because the Prolog interpreter works in a depth first, left-to-right way. Let's now ask, what algorithms we can use for a spelling check:

```
uses('spelling check',ALGORITHM) and get:
ALGORITHM = 'locating of words';
ALGORITHM = 'binary search';
ALGORITHM = 'hash code'.
```

The first is a fact that we stored explicitly in the knowledge base (see Figure 1). The second and the third are inferred by the inference rule since a binary search algorithm implements the locating of words which is useful for the spelling check and analogous for 'hash code'. In this way KIR can be used to work with one's knowledge in a nontrivial way.

6. Conclusion

This paper describes a new approach to the design of personal information retrieval systems. We employed logic programming techniques used for the construction of expert systems and theorem provers in order to implement a personal expert system for storing and retrieving documents and domain dependent knowledge. The implementation of KIR is based on concepts of rule based systems including the declarative programming language Prolog. This made it possible to implement an information retrieval system with a deductive query language that allows to incorporate customizable retrieval strategies to reflect one's information needs. It made also possible to implement a consistent deduction mechanisms used for retrieval as well as
inference of conclusions.

Knowledge based information systems have several advantages over conventional systems: They express the users unique way of working with documents. They help directly with his literary work by making the information gained by indexing, abstracting, annotating the documents used in office work part of the information, that can be used in the retrieval process. Domain dependant knowledge, such as definitions and the like can be stored and retrieved. This knowledge can not only be put into use for locating one's documents, but also to derive new facts via inference rules. Since the strategies for retrieving documents are expessed as rules too, a user can tailor the information system to his own special needs.

References:


Appendix

The first two paragraphs taken from [MOR]:
(The numbers in brackets are references to literature of the paper)

In a recent study on detecting and correcting spelling errors, Peterson[4] states that for locating words in a word list, "...a hash table approach would seem the most reasonable." However, all the algorithms described do not use it only to restrict the search range (e.g. the cited DEC-10 SPELL program) in which case the hashing is followed by a sequential or binary search. Peterson then cites Damerau[2] who wrote that an inspection showed that over 80 percent of all spelling errors fall into one of the following categories of single errors: (1) one extra letter (insertion); (2) one missing letter (deletion); (3) one wrong letter (substitution); (4) transposition of two adjacent letters (transposition). We call these single errors. Peterson also adds that deletion and substitution are difficult to detect.

We have developed a method for overcoming single errors by using special operators, that operate on the so-called significant words, that is, the text words that are deemed sufficiently important to serve as keywords in queries. (In a full-text retrieval system, this significant words may be all text words or all but some 150 common words.) The method is implemented by means of hashing, and no other table lookups are needed. However, it may be implemented by other search techniques, such as binary search in a sorted table, instead of hashing.