A Plan Based Approach to Information Agents

Kristian J. Hammond
Robin D. Burke

Intelligent Information Laboratory
University of Chicago
1100 E. 58th St.
Chicago, IL 60637

Our aim in this work was to develop a set of active planning and information agents, that use a client's goals, plans, and preferences to determine what information to find, filter, and present to the user. These agents will be desgined to help users articulate their goals and develop the plans designed to meet them. This focus on plans and goals will provide our information agents with a touchstone of relevance that can be used to decide what information seek, where to look for it, and how to present it. Our initial work in this area was aimed at the development of a natural language (specifically question-based) front end to information sources and the design of a set of systems to aid users in preference browsing. This work was embodied in the **FAQ Finder** and **Find Me** systems. This report describes these two efforts in detail.

FAQ Finder

A large part of the work done under this grant went into **FAQ Finder**, a natural language question-answering system that uses files of frequently-asked questions as its knowledge base. Unlike AI question-answering systems that focus on the generation of new answers, **FAQ Finder** retrieves existing ones found in frequently-asked question files. Unlike information retrieval approaches that rely on a purely lexical metric of similarity between query and document, **FAQ Finder** uses a semantic knowledge base (WordNet) to improve its ability to match question and answer. We include results from an evaluation of the system's performance, and show that a combination of semantic and statistical techniques works better than any single approach

Introduction

In the vast information space of the Internet, individuals and groups have created small pockets of order, organized around their particular interests and hobbies. For the most part those who build these information oases have been happy to make their work freely available to the general public. One of the most outstanding examples of this phenomenon can be found in the wide assortment of frequently-asked question (FAQ) files, many associated with USENET newsgroups, which record the consensus of opinion among a group on some common question. So that newcomers do not ask the same questions again and again, most FAQs are periodically posted on the newsgroups to which they are relevant. This information distribution mechanism works well for individuals who are sufficiently interested in a topic to subscribe to its newsgroup, but not necessarily to those with a more transient interest. What is needed is a centralized means of access to these answers.

We believe that the most natural kind of interface to a database of answers is the question, stated in natural language. While the general problem of understanding questions stated in natural language remains open, we believe that the simpler task of matching questions to corresponding question/answer pairs is feasible and practical. The aim of the **FAQ Finder** project is to construct a question-answering system that extends further the aim and intent

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave Blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	14 Aug 97	Final Technical Report 1 Oct 95 - 7 Jun 96

4. TITLE AND SUBTITLE
Final Report on a Plan Based Approach to Information Agents

5. FUNDING NUMBERS
C N00014-96-1-0114

6. AUTHORS
Dr. Kristian J. Hammond
Dr. Robin Burke

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
The University of Chicago
Department of Computer Science
1100 East 58th Street
Chicago, IL 60637

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Office of Naval Research - ONR Code 311
Ballston Centre Tower One
800 North Quincy Street
Arlington, VA 22217-5660

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Available for general distribution

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

Our aim in the work set forth in this proposal is to develop a system architecture, composed of a set of active planning and information agents, that supports building systems that use a client's goals, plans, and preferences to determine what information to find, filter, and present to the user. This architecture will also support agents that help users articulate their goals and develop the plans designed to meet them. This focus on plans and goals will provide our information agents with a touchstone of relevance that can be used to decide what information seek, where to look for it, and how to present it.

This proposal draws on our work in both planning and information systems. In particular, it has its roots in case-based planning, information presentation and browsing systems, reactive execution, and multi-media information systems.

14. SUBJECT TERMS
Case Based Reasoning, Information Systems

15. NUMBER OF PAGES
27

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UL	UL	UL	UL

of the FAQ file phenomenon. The system is an information service, available on the World-Wide Web at http://faqfinder.cs.uchicago.edu/ . A user can pose a question and if it happens to be similar to one of the frequently-asked ones whose answer has been recorded in a FAQ file, the system will have a good chance of answering it.

FAQ Finder is built on four assumptions about FAQ files. (1) QA format: All of the information in a FAQ file is organized in question/answer format; (2) locality of information: All of the information needed to determine the relevance of a question/answer pair can be found within that question/answer pair; (3) question relevance: The question half of the question/answer pair is the most relevant for determining the match to a user's question; (4) General knowledge: Broad, shallow knowledge of language is sufficient for question matching.

We see assumptions 1-3 as leading to an essentially case-based (Kolodner, 1993) view of the FAQ retrieval problem. A question/answer pair might be loosely considered a kind of case: it is a piece of knowledge that has been considered useful enough to be codified for reuse. The question serves as an index to the knowledge contained in the answer. These assumptions do not hold for all FAQ files, as we discuss below. but, they hold often enough to form a good starting point for research.

Sample interaction

The following sequence of figures depicts a typical interaction with **FAQ Finder**. Figure 1 shows the World-Wide Web interface to the system. Suppose the user enters the following question:

Is downshifting a good way to slow down my car?

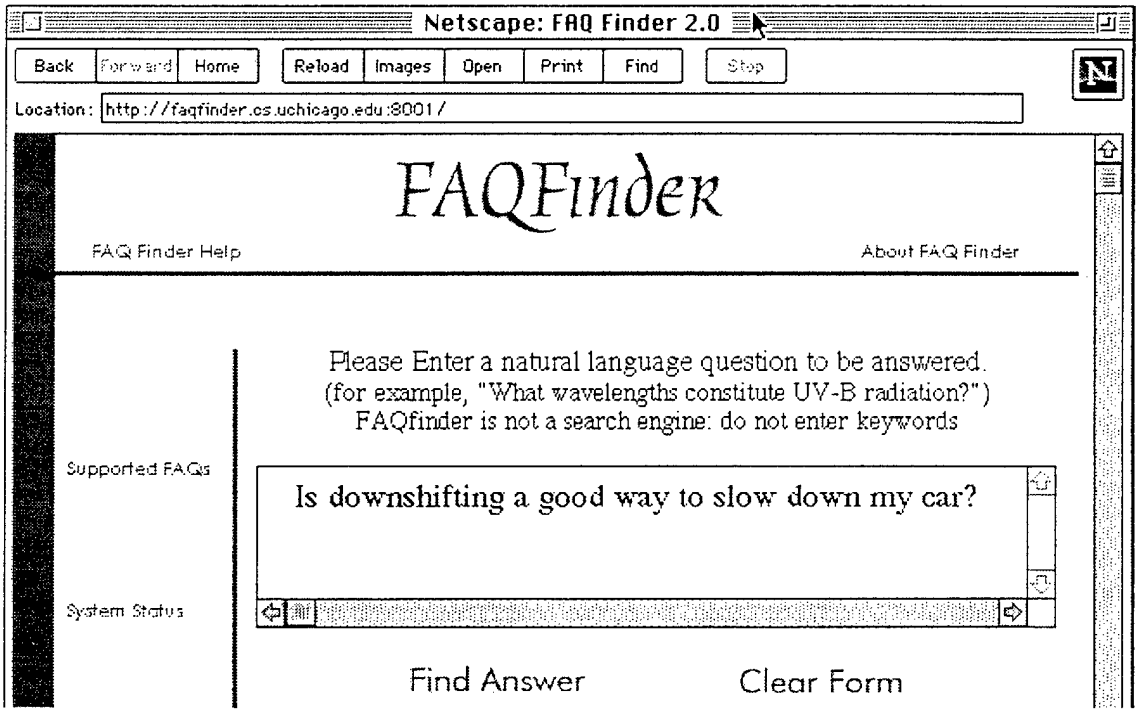

Figure 1: Asking a question of FAQ Finder.

FAQ Finder compares the question to its set of FAQ files, returning a list of files ranked by their relevance to the question (Figure 2), including in this case the file auto as the most relevant file. Some files of questionable relevance are also retrieved, a typical artifact of a keyword-based retrieval system. If the user chooses "Quick Match" when entering a question, the system will skip this first step of manual file selection and automatically choose the top-ranked file to match against.

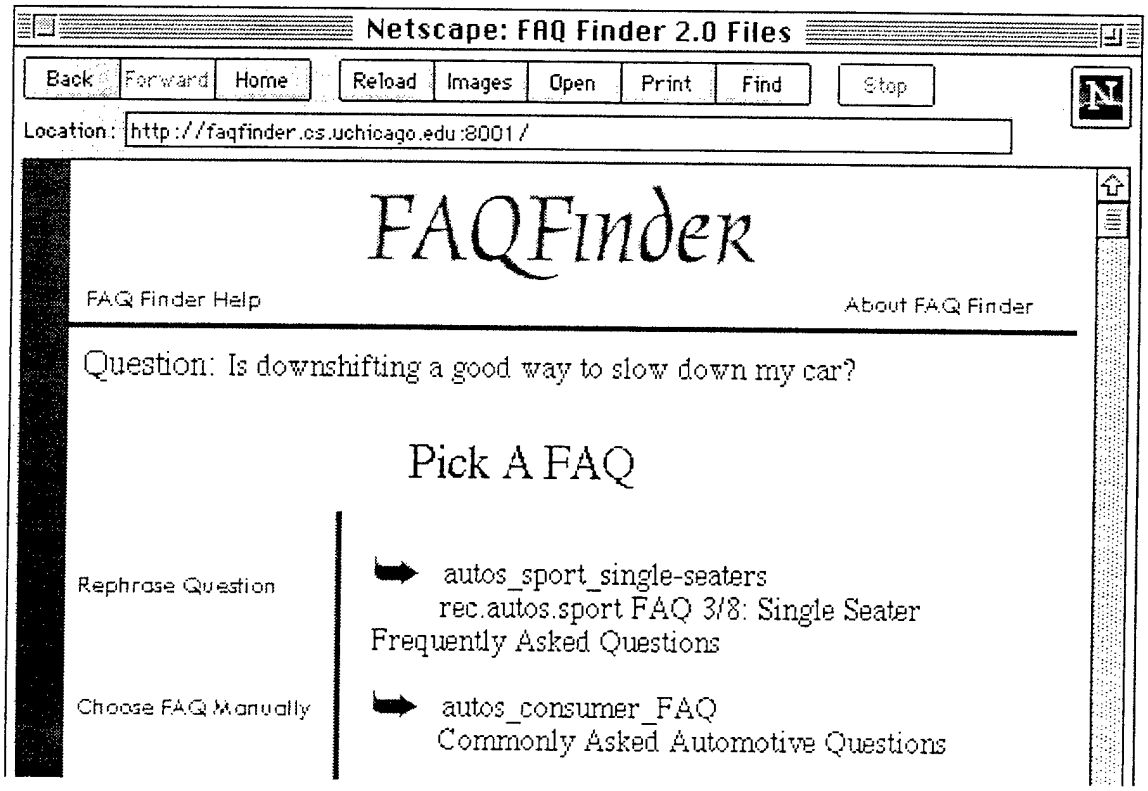

Figure 2: A Choice of FAQs

When a FAQ file is chosen, the system iterates through the QA pairs in that file, comparing each against the user's question and computing a score. The best five matches are returned to the user along with the first line of the answer, so the validity of the answer can be easily determined. Figure 3 shows the results of this matching step comparing our question about "downshifting" with the entries in the consumer_auto_FAQ. The right answer, a question about downshifting and braking is first, followed by two questions about brakes and two about tires. By selecting the appropriate link, the user can view the entire answer given in the FAQ file.

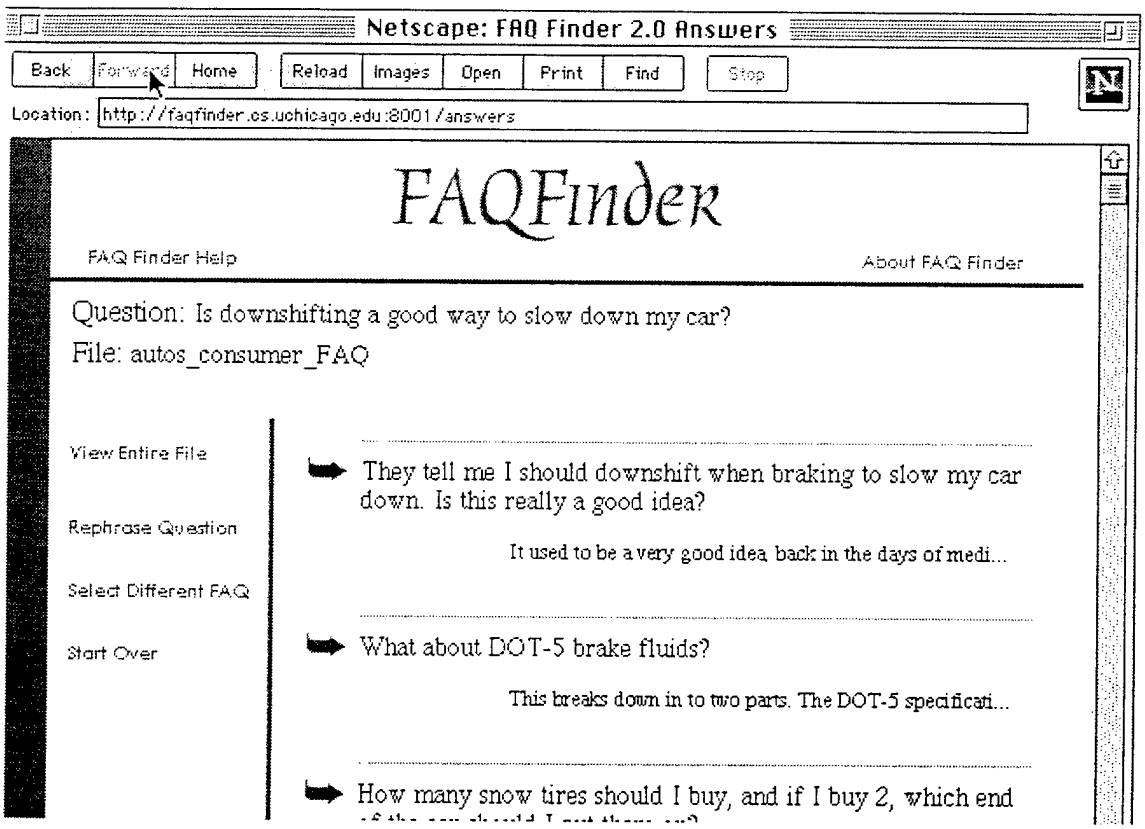

Figure 3: Choosing an answer.

How the system works

As the example shows, interaction with **FAQ Finder** occurs in a series of stages. The first step is to narrow the search to a small set of FAQ files likely to contain an answer to the user's question. Second, each QA pair is matched against the user's question to find the ones that best match it.

For the first stage of processing, **FAQ Finder** uses standard information retrieval technology, the public domain SMART information retrieval system (Buckley, 1985), to perform the initial step of narrowing the focus to a small subset of the FAQ files. The user's question is treated as a query to be matched against the library of FAQ files. SMART stems all of the words in the query and removes those on its stop list of frequent words. It then forms a term vector from the query, which is matched against similar vectors already created for the FAQ files in an off-line indexing step. The top-ranked files from this procedure are returned to the user for selection. We have not found it necessary to tinker with the default configuration of SMART. We treat this part of the system as a black box that returns relevant files.

The second stage of processing in **FAQ Finder** is a question matching process. Each question from the FAQ file is matched against the user's question and scored. We use three metrics in combination to arrive at a score for each question/answer pair: a statistical term-vector similarity score t, a semantic similarity score s, and a coverage score c. These metrics will be described in detail below. Overall match similarity m is a weighted average:

$$m = \frac{tT + sS + cC}{T + W + C}$$

where T, S, and C are constant weights that adjust the reliance of the system on each metric.

The statistical similarity score at the QA pair level is computed in a manner similar to SMART's document matching. A QA pair is represented by a term vector, a sparse vector that associates a significance value with each term in the QA pair. The significance value that we use is commonly-known as **tfidf** (Salton McGill, 1983). If is the term frequency (the number of times that a term appears in a QA pair), is the number of QA pairs that the term appears in the file, and is the number of QA pairs in the file, then **tfidf** is equal to . The idea behind this measure is to evaluate the relative rarity of a term

within a space of documents, and use that as a factor to weight the frequency of that term in a particular document. A term that appears in every QA pair in a file is probably of little value, and its idf , or value, would correspondingly be zero. A term that appears in only a single QA pair would have the highest possible value. Term vectors for user questions are computed similarly by using the idf values associated with terms in a given FAQ. Term vectors are then compared using another standard information retrieval metric, the cosine of the angle between the vector representing the user's question and the vector representing the QA pair.

The term-vector metric allows the system to judge the overall similarity of the user's question and the question/answer pair, taking into account the frequency of occurrence of different terms within the file as a whole. This metric does not require any understanding of the text, a good thing because the answers in FAQ files are free natural language text, often several paragraphs or more in length.

The **tfidf** measure has a reasonably long history in information retrieval, and has generally been thought to work best only on relatively lengthy documents. This is because only long documents have enough words for statistical comparisons to be considered meaningful. However, we have found that this metric contributes significantly to the overall performance of our matching algorithm (see evaluation discussion below), despite the extremely short length of the "documents" (QA pairs) invalid in matching.

The semantic similarity metric enhances term-vector comparison by taking into account a very shallow level of semantic analysis of lexical items that appear in user and FAQ questions. The semantic matching algorithm in **FAQ Finder** is designed to handle variations in lexical content between input and FAQ questions. For example, consider the following questions:

- How can I get my ex-spouse's debts off my credit report?
- Can I get credit if my ex-husband had a lot of debts?

Here, the difficulty is that there are many ways of expressing the same question, all using different (but semantically related) words and phrases. In large documents, these lexical variations might not affect term-vector comparison greatly, because over the course of the document a variety of synonymous terms might be used. However, in **FAQ Finder** since matching is being performed on a very small number of terms, the system needs a means of matching such synonymous.

This suggests the need for a level of semantic analysis of user and FAQ questions. However, in the **FAQ Finder** system, it is important to balance the depth of analysis with the breadth of coverage. Deep causal reasoning about questions would not be feasible because it would require too much knowledge engineering to cover all of the necessary areas of knowledge. For **FAQ Finder**, we believe that a shallow lexical semantics provides an ideal level of analysis for the system. Such a semantics has three important advantages: (1) It provides critical semantic relations between words; (2) It does not require expensive computation to compute relations; and (3) It is readily available.

For example, since the consumer credit FAQ file is full of questions about credit reports and debts, it is important that the system identify the relation between "ex-spouse" and "ex-husband." This can be done at the level of the words themselves, hence our term "shallow lexical semantics." As an example of deeper semantics, we can consider the following pair of questions: (a) How do I reboot my system? (b) What do I do when my computer crashes?

Here, there is a causal relation between the question variants: rebooting is a causal consequence of having one's computer crash. In order to match these questions, the system would have to understand the causality of the computer domain. Since **FAQ Finder** is intended to encompass the whole gamut of USENET topics, not just computers, it is impractical to expect even this simple level of domain-specific knowledge representation.

FAQ Finder obtains its knowledge of shallow lexical semantics from WordNet, a semantic network of English words (Miller, 1995). The WordNet system provides a system of relations between words and "synonym sets," and between synonym sets themselves. The level of knowledge representation does not go much deeper than the words themselves, but there is an impressive coverage of basic lexical relations. By using a marker-passing algorithm (Quillian, 1968), the **FAQ Finder** system uses the WordNet database to accept variations such as "ex-husband" for "ex-spouse." In particular, marker-passing is performed over the network's synonym and hypernym (i.e., is-a) links.[1]

WordNet is not a single semantic network; separate networks exist for nouns, verbs, adjectives, and adverbs. Syntactically ambiguous lexical items, such as "run," which could be either a noun or a verb, appear in more than one network. We found that unrestricted marker passing, using all networks in which a term appears, led to too many spurious matches, a common problem in marker passing systems in general (Collins Quillian, 1972). We tried several approaches to disambiguate terms to a single WordNet network, including using an existing part-of-speech tagger (Cutting, et al., 1992) and context-free parsing of questions, but in the end, we found that simply relying on the default (most common) word sense for each word worked as well as any of the more sophisticated techniques.

The third metric measures the degree of coverage of user terms by the FAQ question. The intuition behind this measure is to penalize questions that are lacking corresponding words for each word in the user's question. In other words, we do not care if the FAQ file question answers many questions at once, but we want to make sure that the important concepts in the user's question are covered. The coverage value is the percent of words in the user question that have a non-zero (computed in the semantic similarity metric) for some term in the FAQ question.

---

[1] Hypernym links are only provided in Wordnet for nouns and verbs; thus only synonym links are used for adjectives and adverbs.

Evaluating FAQ Finder

We evaluated the performance of **FAQ Finder** on a corpus of questions drawn from the log files of the system's use during the period May to December of 1996. A total of 241 test questions were used to perform the evaluation. We manually scanned each FAQ file for answers to each question, and determined that there were 138 questions that had answers in the FAQ file corpus, and 103 questions that were unanswered.

The most obvious precedents to **FAQ Finder** are information retrieval systems, and standard information retrieval evaluation techniques are a starting point for the evaluation of the system. However, evaluation in **FAQ Finder** is complicated by the fact that the task of the system is different than the information retrieval problem as it is typically posed. Normally, the assumption is that there is a document collection in which there may be a number of documents relevant to the user's query. In contrast, **FAQ Finder** works under the assumption that there is such a thing as a "right answer": a single FAQ QA pair that best addresses the user's question as it was posed. The system's job is to return that answer within the small fixed-size set of results that can be displayed on a single web page. Relevance is not that useful a measure to us because, within a given FAQ, most of the answers are probably somewhat relevant to the user's query.

Because **FAQ Finder**'s task is different, the traditional IR evaluation metrics of recall and precision must be modified somewhat. Recall normally is a measure of the percentage of relevant documents in the document set that are retrieved in response to a query, whereas precision is a measure of the percentage of retrieved documents that are relevant. In our case, however, since there is typically one right answer to be retrieved from a FAQ, these are not independent measures of performance. Assuming that an answer to a user question exists in a FAQ file and that the system returns 5 QA pairs, **FAQ Finder** will perform at either 100 recall and 20 precision (if the answer is retrieved), or 0 recall and precision (if it is not). If no answer exists, then precision will be 0 , and recall is undefined.

To measure the quality of retrieval, then, we calculate our version of recall, which amounts to the percent of questions for which **FAQ Finder** returns a correct answer when one exists. Our calculation is slightly different from traditional recall measures because it does not penalize the system if there is more than one right answer in the file. If there are several answers within a file that answer a user's question, it does not make sense to regard retrieval of only one of these answers as only partial success. If the user's question is answered, it is irrelevant that there was another QA pair that also answered it. Instead of precision, we calculate a value called rejection , the percentage of questions that **FAQ Finder** correctly reports as being unanswered in the file. We feel that these metrics better reflect **FAQ Finder**'s real-world performance than traditional recall and precision.

Rejection is adjusted in **FAQ Finder** by setting a cut-off point for the minimum allowable match score. As with precision, there is a trade-off between recall and rejection. If the rejection threshold is set too high, some correct answers will be eliminated; on the other hand, if the threshold is too low, then incorrect responses will often be given to the user when no answer exists in the FAQ file.

Results

SMART is highly effective at the file retrieval task. The correct file appears 88 of the time within the top five files returned to the user, and 48 of the time in the first position. This translates to 88 recall and 23 precision.[2]

---

[2] We do not use rejection in evaluating SMART's performance since it is a standard IR system, designed to find relevant documents, not answer questions.

Figure 3 shows the recall vs. rejection results that we obtained for the second stages of the **FAQ Finder** system. As the graph shows, rejection is somewhat low for reasonable values of recall, meaning that the system confidently returns garbage in most cases when there is no right answer in the file. If the rejection threshold is lowered to make it easier to identify questions without good answers, recall drops dramatically. However the top value for recall is encouraging: there is a better than a two-thirds probability that the system will find the right answer.

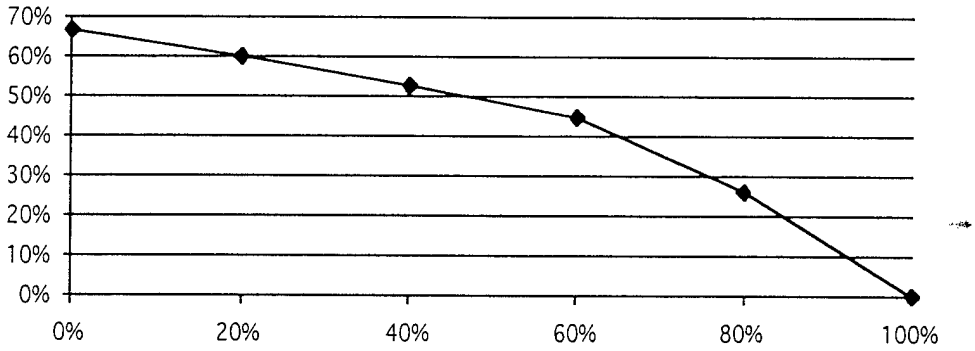

Figure 4: Recall vs. rejection for FAQ Finder

Ablation study

Our next step was to evaluate the contribution of different components of the matching scheme through an ablation study. We selectively disabled different parts of the system and ran the same corpus of questions. There were four conditions: a random condition, in which QA pairs were selected randomly from each FAQ file; a coverage only condition, in which the coverage score for each question was used by itself; a semantic score only condition, in which only the semantic scores derived from WordNet were used in evaluating answers; and, a statistical score only case, in which the term vector comparison was used in isolation.

Figure 4 shows average recall results for these conditions. Interestingly, both WordNet and our statistical technique are contributing strongly to system performance. These two methods had very similar average recall, but are clearly not equivalent measures, since their combination yields results that are better than either individually. These results confirmed our earlier results with a small corpus of questions, which showed an even more dramatic benefit from the combination of methods.

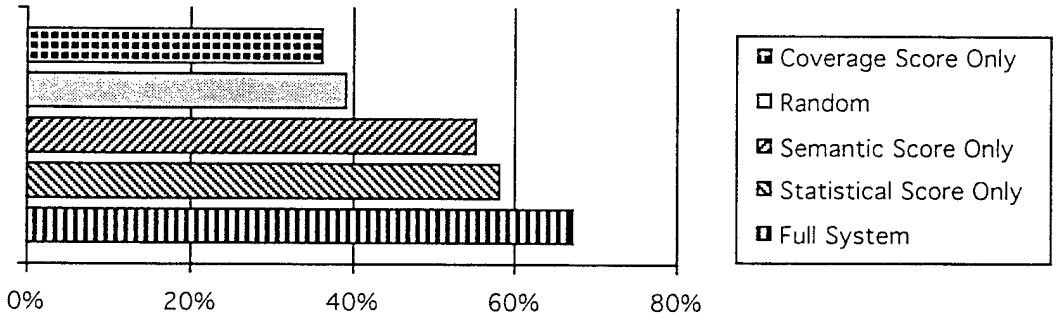

Figure 5: Recall for ablation study.

Figure 5 , which shows the recall vs. rejection analysis for these conditions, provides even more evidence that the two measures differ. The curve for the semantic scoring condition is particularly striking. Although recall in this condition is weaker than the system as a whole, this metric shows good rejection performance. This suggests that the application of semantic information might be used specifically to improve rejection.

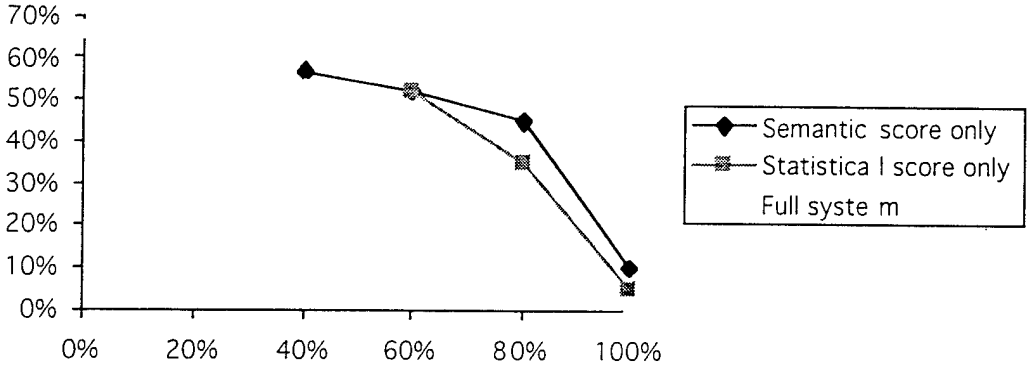

Figure 6: Recall vs. rejection for ablation study.

Discussion

A preliminary analysis of the failure cases, questions for which the system failed to find answers, suggests that the biggest culprit is usually undue weight given to semantically useless words. For example, a question such as "where can I find woodworking plans for a futon?" retrieves questions that incorporate the word "woodworking" as strongly as those that contain "futon", even though "futon" should be a much more informative term inside the woodworking than "woodworking", which applies to everything. The problem is that the term "woodworking" does not appear that often in the FAQ despite its close semantic relation to words that do appear.

Another type of problem commonly encountered with **FAQ Finder** is related to violations of the assumptions about FAQ files discussed at the beginning of this paper: the assumptions of question/answer format, locality of information, question relevance, and sufficiency of general knowledge. We have found many instances in which these assumptions are violated. For example, FAQ writers frequently use headings to mark sections of their documents and rely on the reader's interpretation of those headings in their question writing. In the "Investment FAQ" file, the following text can be found:

> Subject: Analysis - Technical:
> . . .
> Q: Does it have any chance of working?

The "it" is of course intended to refer to technical analysis. However, **FAQ Finder** is currently not capable of making use of this referent because it lies outside the question/answer pair, making it more difficult to match against a question like "Does technical analysis work?" The statistical component of the matcher may enable the question to be retrieved anyway, but we would prefer that the semantic component also be able to contribute in this situation. Part of our intent as we automate the tagging process is to make heading information available to the matcher.

There are other more difficult cases of ellipsis found in FAQ files. In the "Wide-Area Information Server FAQ," the following passage can be found:

> Q: What is Z39.50?
> A: ...
> Q: Do they interoperate?

The reference "they" refers to both Z39.50, an information retrieval standard, and WAIS, the subject of the FAQ. We do not expect **FAQ Finder** to be able to dissect references that are this oblique. It would, however, be useful to refer back to earlier questions if there is no heading information with which to resolve a referent.

One FAQ-specific phenomenon we have encountered is the use of metasyntactic variables . meaningless pieces of text that stand in for a filler, which can vary. For example, the "Pool and Billiards FAQ" contains the question

> Q: What are the rules for XXX?
> A: STRAIGHT POOL.
> B. EQUAL OFFENSE.

Metasyntactic variables often have a distinct form and can be easily recognized. We anticipate that a mechanism similar to a heading recognizer could be used to recognize the sub-answers within a multi-part answer such as this. Not every variable can be so treated, however. The "Woodworking FAQ" contains the question Q: Should I buy a Sears blurfl? The answer does not enumerate the entire catalogue of Sears power tools: the same advice is intended to apply to all. The reader is supposed be capable of matching the nonsense word against the name of any power tool. This is exactly the type of domain-specific knowledge that we have sought to avoid including in **FAQ Finder**. **FAQ Finder** can successfully match this question against questions like "Are Sears power drills a good buy?" because the word "Sears" is sufficiently distinctive, but it would fail to match against a question like "What kind of power drill should I buy?"

Future Work

The previous discussion suggests many areas that deserve future research attention. One of the most obvious open questions is the problem of improving the system's rejection of unanswerable questions. We have concentrated our tuning of the system to maximize recall, so that answerable questions will be answered. However, it is also useful to be informed that an answer does not exist within a FAQ file. This may suggest to the user that the question should be submitted to the FAQ's related newsgroup. If rejection were sufficiently good, we could incorporate such an option into the system itself.

One way of approaching this problem is to focus on the small retrieved set of QA pairs before they are returned to the user. We know from our evaluation that if an answer is present in the FAQ file, the system is likely to find it. Therefore if none of the QA pairs returned by the system are in fact a good answer, the chances are that the system should report that the answer does not exist. We also know that semantic information seems to have better rejection characteristics than statistical information. We may be able to perform a more in-depth analysis, involving deeper natural language processing to accept or reject each returned set of questions. Because this set is by definition small, such intensive processing would not be as computationally prohibitive as performing deeper natural language processing throughout the entire matching process.

An important part of maintaining the performance of **FAQ Finder** on a large set of FAQ files will be the incorporation of new vocabulary items into WordNet. Since WordNet was formed from a corpus of everyday English, its vocabulary does not include many technical terms or proper nouns. Unfortunately, due to the technical nature of many FAQ files, technical terms and proper nouns constitute a significant portion of the domain vocabulary of these files. In addition, these terms can be the most useful in retrieving relevant FAQ question/answer pairs, since they are often the most specific and discriminating terms. Thus, the fact that they are missing from WordNet can significantly impair the performance of **FAQ Finder**.

We are investigating ways in which information obtained from the parses of questions can be used to automatically acquire additional terms and build the appropriate synonym and hypernym links for these terms in one of the WordNet hierarchies. We will rely on feedback from the user to tell the system when a good match has been found between a user question and a FAQ question/answer pair. Using our previous local version of **FAQ Finder**, about 20 of users gave this type of feedback. We expect that the improved interface of **FAQ Finder** 2.0 will increase the frequency of user feedback on performance of the system. If the user indicates the system retrieved a relevant answer, then any words in either the user or the FAQ question that are not contained in WordNet have the potential to be acquired. The system will then attempt to match the unknown word with a synonym in the other question. Both questions will be parsed, and position in the parse tree used to determine which words are candidate synonyms of the unknown word.

Since the matching process between question pairs is likely to incorrectly propose some synonyms of unknown words, our approach is to collect synonyms for unknown words over time, and propose new WordNet entries by analyzing collections of possible synonyms for each unknown term. Clustering algorithms are likely to be of use here in determining the likely best entry(ies).

FAQ Finder Conclusions

FAQ Finder is a Web-accessible knowledge-based information access system that relies on the knowledge engineering inherent in FAQ files distributed on the Internet. The **FAQ Finder** project has shown that when there is an existing collection of questions and answers, question answering can be reduced to matching new questions against question/answer pairs, a considerably more tractable task than question understanding. The system combines statistical measures and shallow lexical semantics to match users' questions against question/answer pairs from FAQ files. Our evaluation, although conducted with a small corpus of questions, have demonstrated the effectiveness of the system.

The power of our approach rises from the fact that we are using highly organized knowledge sources that are designed to "answer" the commonly asked questions. We do not need our systems to actually comprehend the queries they receive (Lang, et al. 1992) or to generate new text that explain the answer (Souther, et al. 1989). They only have to identify the files that are relevant to the query and then match against the segments of text that are used to organize the files themselves.

Ultimately, FAQ files are a social phenomenon, created by people to record and make public their understanding of a field. In general, the **FAQ Finder** project is interesting in that it uses not just the existing archives on the Internet, but also the existing sociology. One of the more powerful aspects of the newsgroups is the collective desire on the part of the users to "get it right." This drive has already resulted in the existence FAQ files

themselves. Our aim in **FAQ Finder** is to further this goal by making the answers recorded in FAQ files more widely available.

Find Me Systems

While the explosion of on-line information has brought new opportunities for finding and using electronic data, it has also brought to the forefront the problem of isolating useful information and making sense of large multi-dimensional information spaces. In response to this problem, have developed an approach to building data "tour guides," called **Find Me** systems. These programs know enough about an information space to be able to help a user navigate through it, making sure that the user not only comes away with items of useful information but also insights into the structure of the information space itself. In these systems, we have combined ideas of instance-based browsing, structuring retrieval around the critiquing of previously retrieved examples; and retrieval strategies, knowledge-based heuristics for finding relevant information. We illustrate these techniques with examples of working **Find Me** systems, and describe the similarities and differences between them.

Introduction

What do buying a car, selecting a video, renting an apartment, choosing a restaurant, and picking out a stereo system have in common? They are all tasks that require an individual to pick from a large collection of similar items one which best meets that person's unique needs and tastes. Because there are many interacting features of each item to consider, such selection tasks typically require substantial knowledge to perform well. Our aim is to build systems that can help users perform such tasks, even when they do not have a lot of specific knowledge. Our approach, called *assisted browsing*, combines searching and browsing with knowledge-based assistance.

Suppose you want to rent a video. You are in the mood for something like *Back to the Future*. What are your options? You might want to see the sequel, *Back to the Future II*. Or maybe you want to see another movie about a person dropped into an unfamiliar setting, such as *Crocodile Dundee*, or *Time After Time*, another time-travel movie. If you really enjoyed the way *Back to the Future* was directed, may you would like *Who Framed Roger Rabbit?* , another Robert Zemeckis picture. Or perhaps, you want to see another film starring Michael J. Fox, such as *Doc Hollywood*. No computer system can tell you what movie to see, but an intelligent assisted-browsing environment can present you with these choices (and others), getting you to think about what you liked in *Back to the Future*.

The aim of assisted browsing is to allow simplified access to information along a multitude of dimensions and from a multitude of sources. Since browsing is the central metaphor, we avoid as much as possible forcing users to create specific queries. Knowledge-based retrieval strategies can be employed to consider all of the dimensions of the information and present suggestions that lead the user's search in reasonable directions. We have implemented our assisted browsing approach in a series of systems called **Find Me** systems. They are:

- **Car Navigator**: Selecting a new car,
- **Video Navigator** & **Pick-A-Flick**: Choosing a rental video,
- **Rent Me**: Finding an apartment,
- **Entree**: Selecting a restaurant,

- **Kenwood Home Theater**: Configuring a home theater system.

We see the **Find Me** approach as applicable to any domain in which there is a large, fixed set of choices and in which the domain is sufficiently complex that users would probably be unable to fully articulate their retrieval criteria. In these kinds of areas, person-to-person interaction also takes the form of trading examples, because people can easily identify what they want when they see it.

Figure 7 shows the entry point for **Entree**, a restaurant guide for the city of Chicago. Users can pick from a set of menu options to describe what they are looking for in a restaurant: a casual seafood restaurant for a large group, for example, or they can, as shown here, type in the name of a restaurant in some other city for which they are seeking a local counterpart.

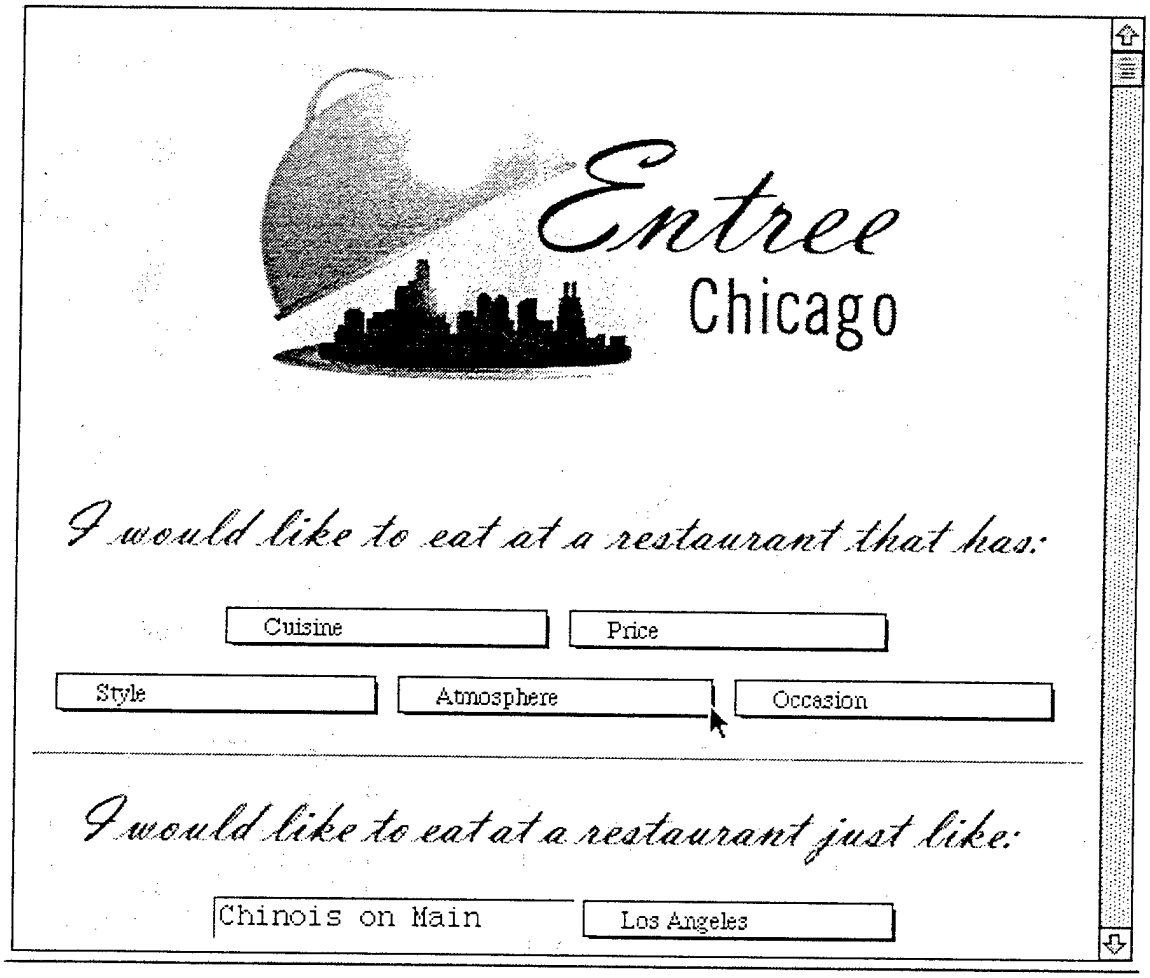

Figure 7: The initial screen for Entree

The system retrieves restaurants in the Chicago that are considered to be similar to the user's choice of "Legal Seafood," the top contender being "Bob Chinn's Crabhouse" as shown in Figure 8. The user can now continue to browse the space of restaurants by using any of the seven *tweaks*, modifications to the example. The user can ask for a restaurant

that is nicer, or less expensive, one that is either more traditional or more creative, one that is quieter or more lively, and also has the option of looking for a similar restaurant but with a different cuisine.

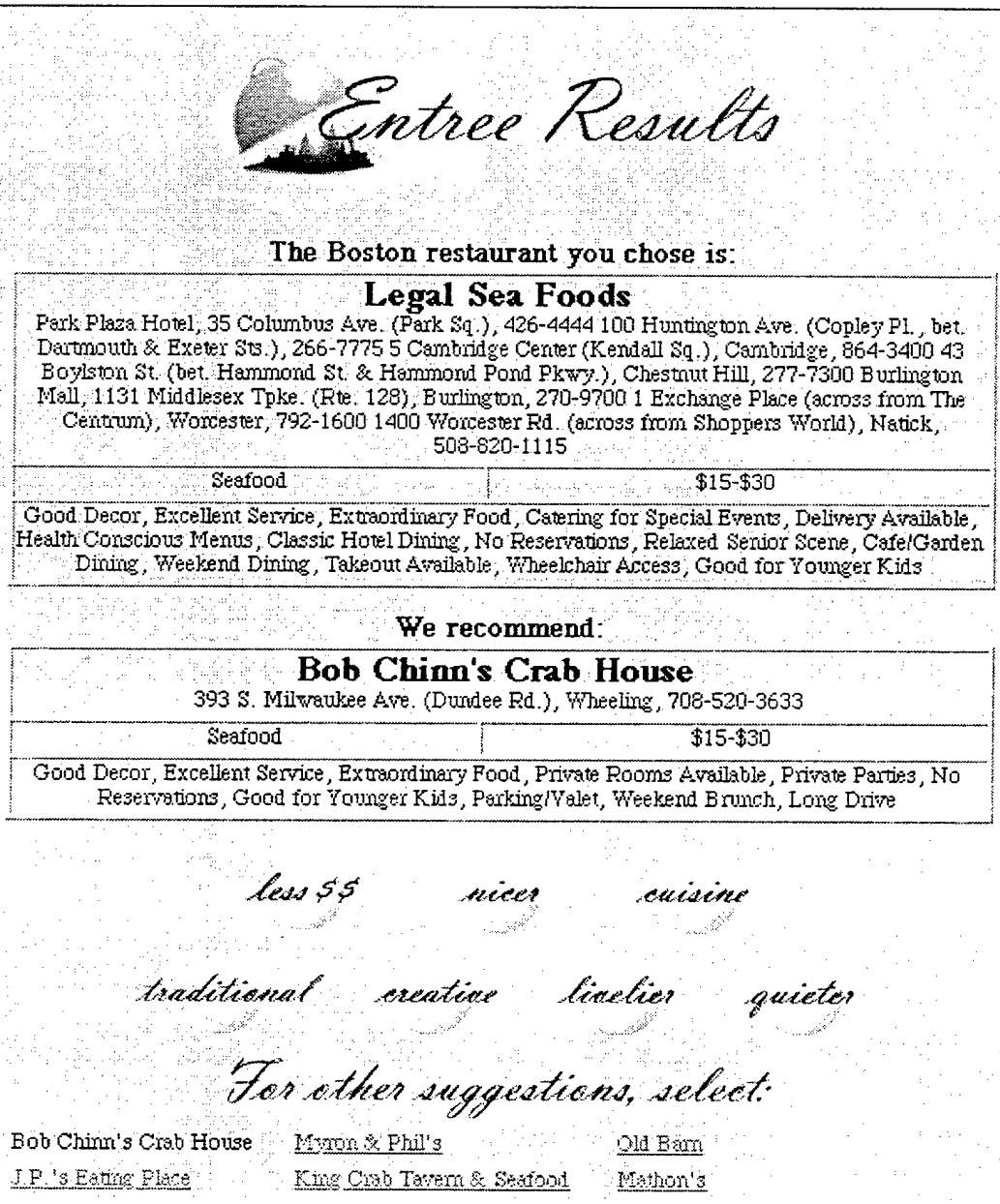

Figure 8: Tweaking in Entree

This example shows some of the kind of intelligent assistance that are used in **Find Me** systems:

Similarity-based retrieval: As has frequently been found in other information retrieval contexts, it is useful to allow a user to retrieve new items that are similar to an example currently being viewed (Ullman, 88 and Williams *et al* , 82). We found that in most cases

overall similarity of features was a poor metric for providing examples, because users attached different significance to features depending on their goals. For example, if your goal is to buy a car that will pull a big trailer, you will weight engine size more heavily when comparing cars than other features such as passenger leg room. So, the system should regard engine size as more significant is assessing similarity in this context.

Tweaking: Browsing is typically driven by differences: if a user were totally satisfied with the particular item being examined he or she would stop there. But, an unsatisfactory item itself can play a useful role in articulating the user's goals. For example, if you are looking for a science fiction movie to rent, you might look at *Terminator II*, but think "That would be good, but it's too violent for my kids." The examination of a particular example can bring to mind a new feature, such as level of violence, that becomes an explicit part of further search.

Although not present in **Entree**, there is another category of assistance that is often useful:

Explanations of trade-offs: Users, especially in unfamiliar domains, may fail to understand certain inherent trade-offs in the domain they are exploring. A car buyer might not understand the trade-off between horsepower and fuel efficiency, and attempt to search for a high-powered car that also gets 50 miles to the gallon.

These mechanisms are part of a dialogue between system and user in which the user comes to a better understanding of the domain of examples (through learning about trade-offs and seeing many examples) and the system helps the user find specific items of interest by gradually refining the goal.

Technical Overview

At the highest level of abstraction, all **Find Me** systems are very similar. They contain a database, they retrieve from it items that meet certain constraints, and they rank the retrieved results by some criteria. What gives each **Find Me** system its character is the details of how this general pattern is instantiated for any given domain, particularly in what criteria are used for retrieval, what criteria are used for ranking results, what tweaking transformations are incorporated into the system, and what additional knowledge is brought to bear in addition to the database itself.

It is important in building a **Find Me** system to understand the relationship between features of the data and the selection task itself. We cannot make use of all features available in a database in a uniform way. The hours of operation of a restaurant are rarely as important as how much a typical meal costs, for example. Also, it does not make sense to build every possible tweaking option into the interface. Rarely would a user look at an apartment and say "I want something just like that, but more expensive." **Find Me** systems concentrate a single possible use for the data in a database, but because of this focus, they can provide more assistance to the user.

The Entree implementation

Entree contains the essence of the **Find Me** idea, stripped down to its essentials. It therefore makes a good example with which to explain the functionality of a **Find Me** system. **Entree** consists of a handful of perl scripts that handle the output of web pages, and several C++ programs that implement **Find Me** functionality. Conceptually, a

restaurant *r* is represented in the system as a tuple *<i, d, N, F>*, where *i* is a unique integer identifier for each restaurant used to index the tuple, *d* contains the name of the restaurant and other descriptive text about it to be displayed for the user's benefit, *N* is a set of indexed *trigrams*, a decomposition of the restaurant's name into three letter sequences (see below), and *F* is the set of features of the restaurant itself.

When the user enters **Entree** from the initial page (Figure 7), there are two possibilities (a) a particular restaurant has been entered as a model, or (b) a set of high-level features has been selected from the set of menus.

In the first case, the system must attempt to find a restaurant with the name the user has supplied. Since users are likely to mistype, misremember or misspell such names, we have a fuzzy string comparison routine that uses trigrams, looking for the restaurant name in the database that shares the most trigrams with what the user typed. The comparison is also sensitive to the location in the name where the sequence occurs. We find that this enables many misspellings to match with the correct restaurant. The name matcher returns the id for the restaurant that the user has named, which is used to lookup the corresponding feature set.

In the case of the second entry point, the user selects a set of high-level features describing their dining interests. For example, a a casual seafood restaurant for a large group. These high-level features are decomposed into a set of low-level database features. In either case, the entry point provides the system with a set of features, *F*. In the menu case, **Entree** also gets some goal information it can later use to tune its ranking of results.

The next step is retrieval of *R*, the set of all restaurants containing one or more features from *F*. *R* is a large set, typically 20-50% of the entire database. For example, when the user selects "Legal Seafood," we retrieve all Chicago restaurants that serve seafood, but also all that charge about $15, all that are similarly casual, etc.

On *R*, we perform a hierarchical sort. Suppose we have an ordered list of goals $\{G_1,...,G_k\}$ we can apply the goal-related metric M_{G_i} to each retrieved example. Since the metric is discrete, we can create equivalence classes or buckets based on the score returned by this metric. Then the examples are ranked within each bucket with respect to the next most important goal, creating another series of more finely-discriminated buckets. We can repeat this process until either all possible ranking operations have been performed or there is a totally-ordered set of examples to show. To make the process efficient, we continuously truncate the set of buckets so that it contains only the minimum number of buckets needed to answer the query.

Entree has a default ordering of goals that is assumed if the user enters a restaurant by name: cuisine is the first priority, followed by price. So the first two passes on sorting will return all of the seafood restaurants ordered inversely by price, starting from the $15 price bracket. The next goal the system assumes is atmosphere: the feel of the dining experience. Finally, we use ratings of quality to rank the final list.

The restaurants are returned on a "results" page as shown in Figure 8, with a single restaurant highlighted, and a list of links to other similar restaurants below. Each of these links returns another results page, differing only in that the chosen restaurant is highlighted instead.

All of the tweaks are implemented in essentially the same way. We perform retrieval based on similarity, just as described above, however, we then filter out of R all of the restaurants that do not satisfy the tweak given. For example, if the user looking at Figure 8 decides to look for something "nicer," the system would calculate how "nice" it thinks "Bob Chinn's Crab House" is, and then creates a subset R' of all of the restaurants in R that are nicer than "Bob Chinn's." It then performs the ranking in exactly the same way as before, looking at cuisine, price, atmosphere, etc. In some cases, the result of the tweak filter will be empty, in which case, we report to the user that there are no more restaurants along the given dimension within the preferred cuisine. The system will not switch from "seafood" to "French" in order to continue along the "nicer" dimension, because cuisine is so basic to the restaurant-finding task. This option is available to the user via the "Cuisine" tweak.

Some Find Me **Systems**

In general, as we have built **Find Me**, we have worked from domains with small spaces of examples in which features are well-defined and user goals are straightforward, to larger domains with fuzzier features and more complex user goals. Each of the systems is profiled in this section.

Car Navigator

The first **Find Me** system was the **Car Navigator**, an assisted browsing system for new car models. Using the interface, which resembles a car magazine, the user flips to the section of the magazine containing the type of car he or she is interested in. Cars are rated against a long list of criteria such as horsepower, price or gas mileage, which are initially set by default for the car class, but can be directly manipulated. Retrieval is performed by turning the page of the magazine, at which point the criteria are turned into a search query and a new set of cars is retrieved. Depending on how the preferences have changed, the system may suggest that the user move to a different class of cars. For example, if the user started with economy cars and started to increase the performance requirement, the system might suggest sports cars instead. Figure 9 shows the user interface for **Car Navigator**.

It is possible for the user to set the preferences to an impossible feature combination: one that violates the constraints present in the car domain. This triggers an explanation of the trade-off that the user has encountered. For example, if a user requests good gas mileage and then requests high horsepower the yellow light will come on next to the gas mileage and horsepower features. The system explains that there is a trade-off between horsepower and gas mileage, and the user will have to alter his or her preferences in one area or the other.

In addition to the fine-grained manipulation of preferences, **Car Navigator** permits larger jumps in the feature space through buttons that alter many variables at once. If the user wants a car that is "sportier" than the one he is currently examining, this implies a number of changes to the feature set: larger engine, quicker acceleration, and a willingness to pay more, for example. For the most common such search strategies, **Car Navigator** supplies four buttons: *sportier*, *roomier*, *cheaper*, and *classier*. Each button modifies the entire set of search criteria in one step. Although direct manipulation of the features was appealing in some situations, we found that most users preferred to use the retrieval strategies to redirect the search.

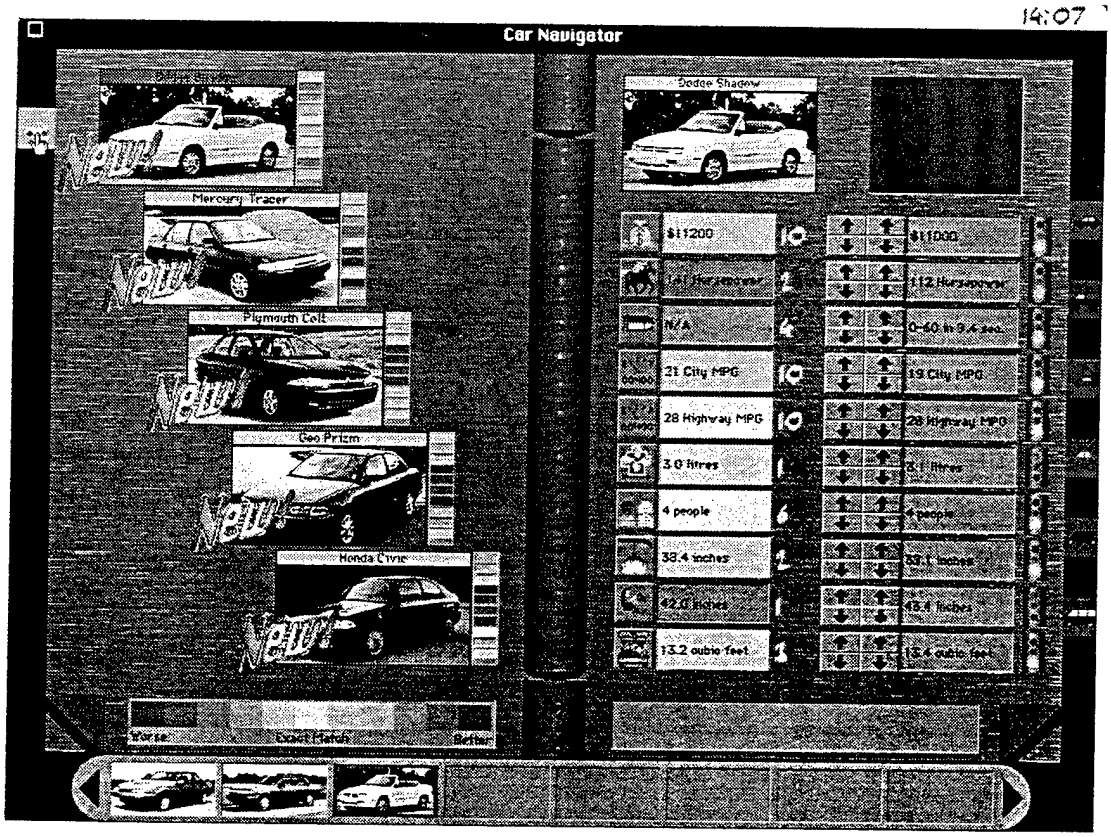

Figure 9: The interface for Car Navigator

The implementation details for this system are outlined in Table 1. The interface was implemented in C, and the database in Lisp, using TCP streams to pass retrieval requests. This design made the interface very responsive, but still allowed us the maximum flexibility in our handling of data.

System	**Car Navigator**
Platform	Macintosh
Language	Macintosh Common Lisp (~4000 lines)
	C (~3800 lines)
Database	Lisp internal
Data Size	1 MB (~600 cars)

Table 1: Implementation details for Car Navigator.

Video Navigator and Pick-A-Flick

We used our experience in building **Car Navigator** in the construction of a system for browsing movie videos for rental. This system, **Video Navigator**, draws on a database of 7500 movies from a popular video reference work (Wiener, 93).The system is organized as a sequence of shelves divided into categories. The user has several tools that can be used to make queries into the shelves. Once at a particular shelf, the user can select movies and look at additional information about them, such as plot summaries, cast lists, etc.

The retrieval mechanism in **Video Navigator** is implemented in a set of interface agents, called *clerks*. This design choice was due to the nature of the movie domain. Users have seen more movies than they have cars. They know more points in the information space, so need less help from the system in getting around. The clerks remain passive until the user selects a particular movie to examine. There are four clerks: one recalls movies based on their genre, one recalls movies based on their actors, another on directors, and still another arrives at suggestions by comparing the user against the profiles of other users. Whenever the user picks a movie to inspect, each clerk retrieves and suggests another related movie. It is as if the user has a few knowledgeable movie buffs following her around the store, suggesting movies based on their particular area of expertise. The user can choose to follow up or ignore the suggestions. Figure 10 shows the interface for **Video Navigator**.

It turned out to be difficult to implement tweaking in the movie domain. While we could easily derive buttons that might be useful: "less violence," for example, it quickly became clear that there were too many possible tweaks to have buttons for each. Ultimately, we would like to have users supply tweaks in natural language phrases and use simple natural language processing techniques to allow the system to recognize tweaks such as "too violent," "I hate musicals," or "Not Mel Gibson."

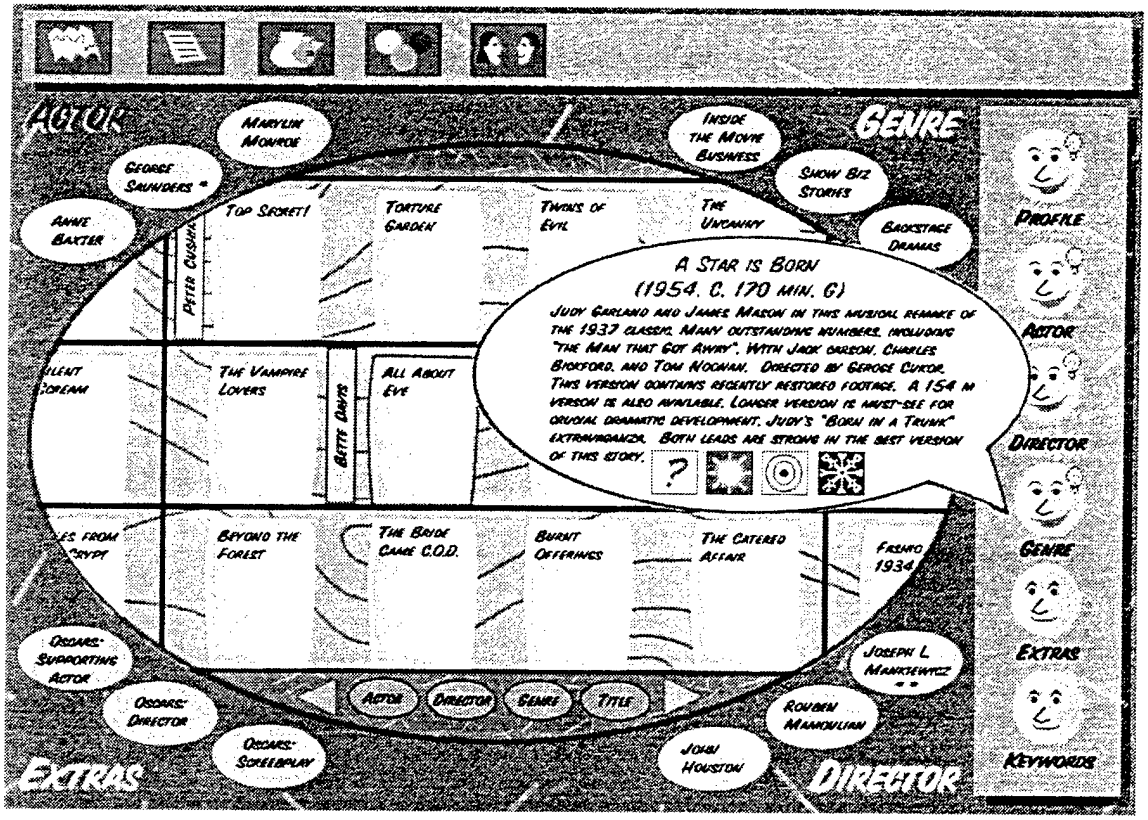

Figure 10: The interface for Video Navigator.

The implementation of **Video Navigator** is summarized in Table 2. We built the system entirely in Macintosh Common Lisp, using the built-in interface development tools. The knowledge base in **Video Navigator** consists of similarity relations between actors and influence relationships between directors.

System	Video Navigator
Platform	Macintosh
Language	Macintosh Common Lisp (4700 lines)
Database	Lisp internal
Data Size	1.9 MB (7500 movies) + 1.4 MB knowledge base

Table 2: Implementation details for Video Navigator.

Using the same knowledge base and algorithms, we created **Pick-A-Flick**, an adaptation of **Video Navigator** to the World-Wide Web. Instead of a browsing interface with a map and shelves, we allow the user to enter the name of a known movie in free text. This movie is used to generate suggestions using retrieval strategies like the clerks in **Video Navigator**.

For example, suppose the user enters the name of the movie *Bringing Up Baby*, a classic screwball comedy starring Cary Grant and Katharine Hepburn. **Pick-A-Flick** locates similar movies using three different strategies. First, it looks for movies that are similar in genre: other fast-paced comedies. As Figure 11 shows, it finds *His Girl Friday*, another comedy from the same era starring Cary Grant, as well as several others. The second strategy looks for movies with similar casts. This strategy will discard any movies already recommended, but it finds more classic comedies, in particular *The Philadelphia Story*, which features the same team of Grant and Hepburn. The director strategy returns movies made by Howard Hawks, preferring those of a similar genre.

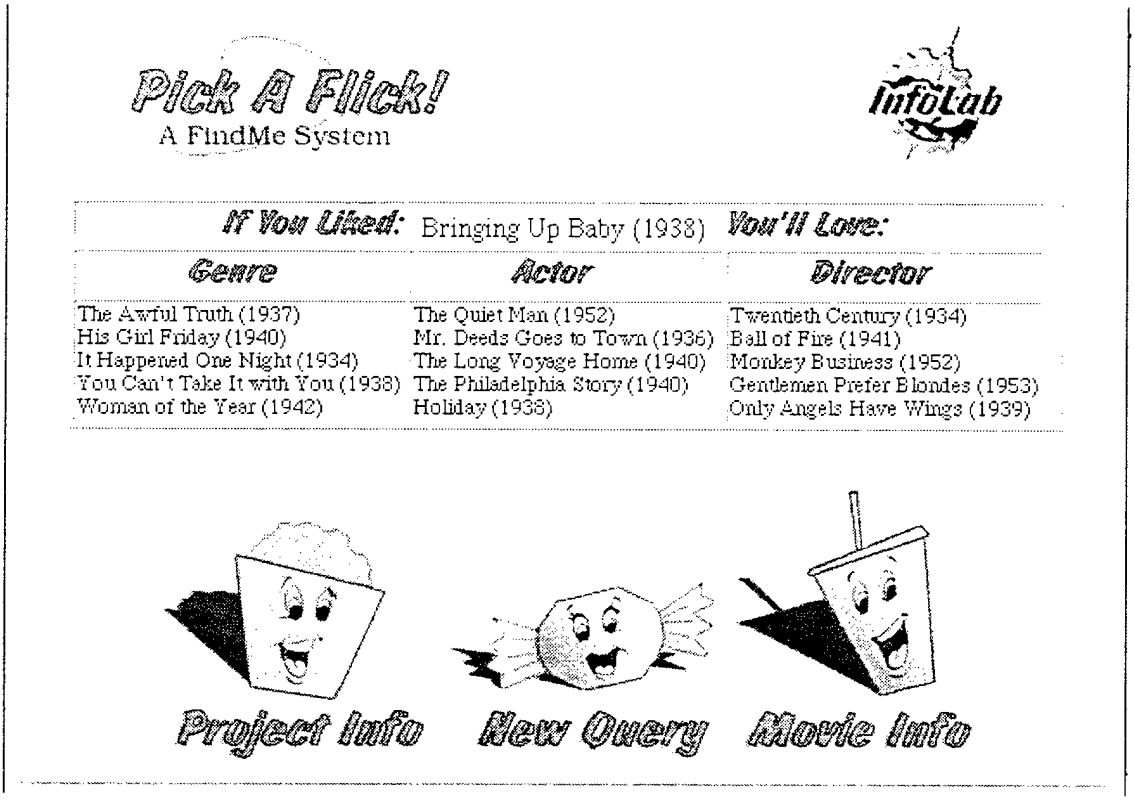

Figure 11: A search result from Pick-A-Flick.

We expanded our movie database when moving to the web platform, handling an order of magnitude more movies, as shown in Table 3. Since the Lisp image containing this database took about 60 seconds to load and initialize, it was impractical to load and run it in response to each web request. We set up the Lisp image as a server, responding to requests from a TCP stream. This stream is created and managed by a set of perl scripts that handle the web requests using the CGI protocol to interact with our web server.

System	Pick-A-Flick
Platform	Web (Sun Solaris)
Language	Allegro Common Lisp (2500 lines) perl (1500 lines)
Database	Lisp internal
Data Size	12 MB (80,000 movies)

Table 3: Implementation details for Pick-A-Flick

Rent Me

All of our subsequent **Find Me** systems have been web-based. **Rent Me** is an interface to a database of classified ads for rental apartments. A typical apartment seeker might have a goal like "I'd like a place like what I have now but a little bigger and in a neighborhood with more stuff to do nearby." Notions such as "like the apartment I live in now" are idiosyncratic and can only be evaluated by the user examining a particular apartment listing. Another important aspect of the goal stated above is its reference to knowledge outside of the domain of the apartment listings themselves. To know whether a neighborhood has "more things to do," one must know something about the city itself.

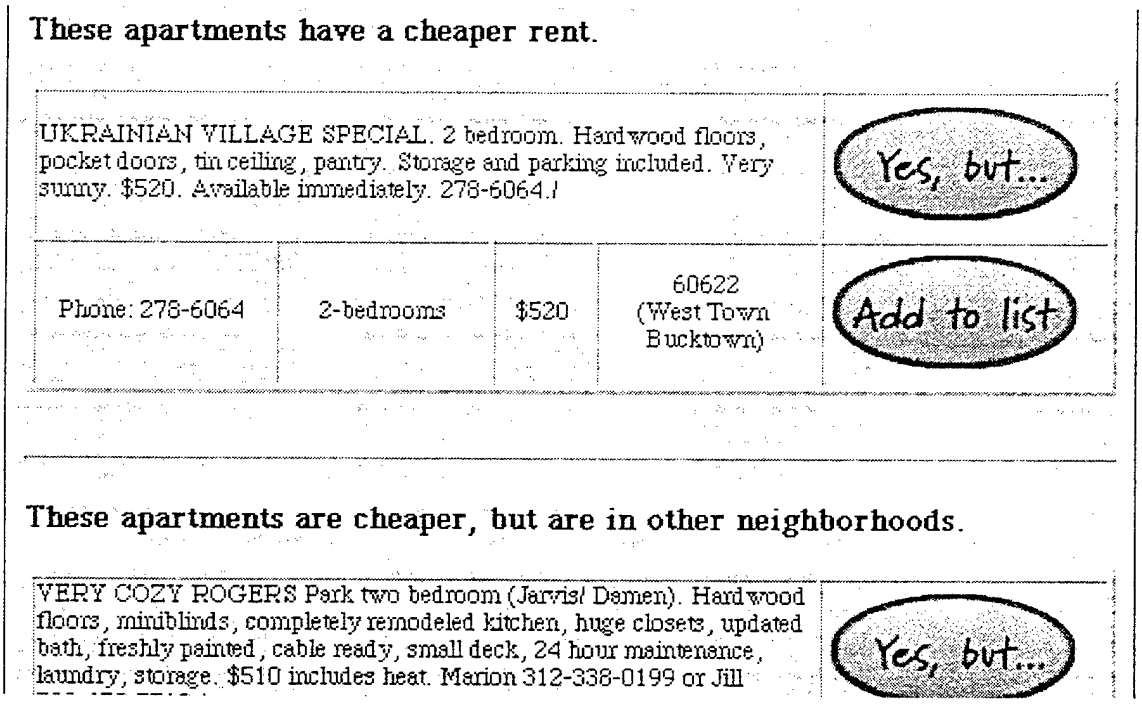

Figure 12: Tweaking an apartment in Rent Me.

The entry point for **Rent Me** is a set of menus: for neighborhood, price and size. The list of apartments meeting these constraints forms the starting point for continued browsing. As shown in Figure 12, the user can improve the search by selecting any apartment and using it as the basis for further retrieval by tweaking. The "Cheaper" button is used to tell the system to find similar apartments that are cheaper. The system performs another round of retrieval, keeping in mind the features of the apartment the user originally selected. As shown in Figure 13, it only finds one acceptable apartment in the same neighborhood, so it relaxes the neighborhood constraint and begins to look at other, similar, neighborhoods for cheaper apartments.

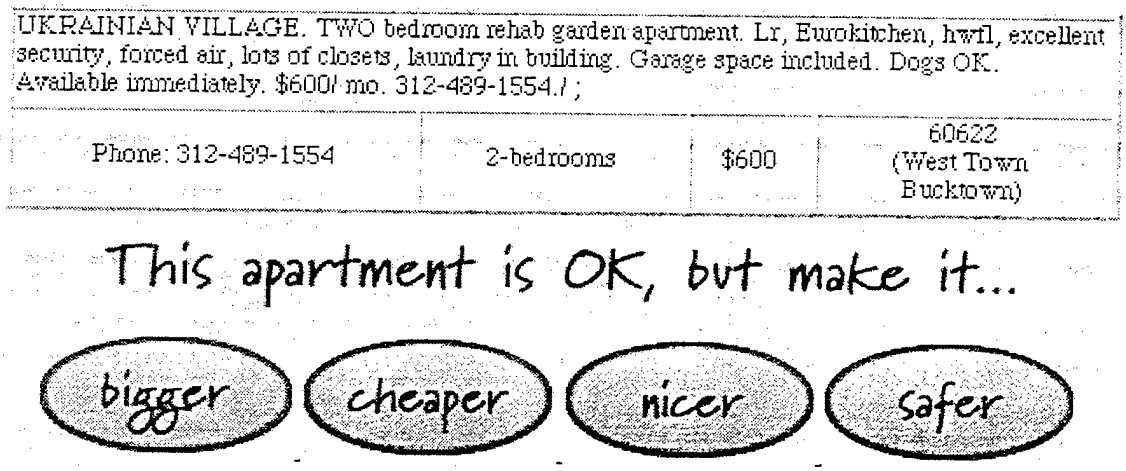

Figure 13: The result of applying the "cheaper" tweak.

Rent Me starts not from a database, but from a text file of classified ads for apartments. It builds the database from this text using an expectation-based parser (Schank & Riesbeck, 81) to extract features from the very terse and often-agrammatical language of the classified ads. Expectation-based parsing makes it possible to distinguish between "No dogs" and "Dogs welcome," a distinction lost to many keyword-based approaches.**Rent Me** was implemented much like **Pick-A-Flick** with a Lisp server containing the database and perl scripts running on a Unix platform. The Lisp code base is large because **Rent Me** also contains the code for the natural language parser.

System	Rent Me
Platform	Web (Sun Solaris)
Language	Allegro Common Lisp (9500 lines) perl (1200 lines)
Database	Lisp internal
Data Size	2.8 MB (3700 apartments)

Table 4: Implementation details for Rent Me.

Entree

Entree was our first attempt to build a **Find Me** system that was sufficiently stable, robust and efficient to survive as a public web site. Our previous systems were implemented in Common Lisp, and could not be made available for public access without

regular monitoring. Also, all of the **Find Me** systems discussed so far keep the entire corpus of examples in memory (the Lisp workspace). This technique has the advantage of quick access and easy manipulation of the data, but it is not realistic in that it cannot be easily updated or scaled up to very large data sets. We use a combination of DBM, a free database package for Unix, and flat text files to store the data for **Entree**.

The system has been operation on the World-Wide Web since August 1996 in the configuration described in Table 5. It was first used by attendees of the Democratic National Convention in Chicago. In addition to its database of restaurants, **Entree** also has knowledge of cuisines --- in particular, the similarities between cuisines. This enables it to smooth over some of the discontinuities that exist between our different data sources. In some sources, "Tex-Mex" was considered a cuisine, in others only "Mexican" was used.

System	Entree
Platform	Web (Sun Solaris)
Language	C++ (3200 lines)
	perl (1200 lines)
Database	DBM and flat text
Data Size	2.2 MB (4400 restaurants)
URL	http://infolab.cs.uchicago.edu/entree/

Table 5: Implementation details for Entree.

Kenwood Home Theater

Our most recent **Find Me** system allows users to navigate through various configurations for home theater systems. The user can enter the system two ways: by selecting a budget or by identifying particular components they already own. The user also must specify the type of room the system will operate in. The user can browse among the configurations by adjusting the budget constraint, the features of the room or by adding, removing or replacing components. Since we are dealing with configurations of items, it is also possible to construct a system component by component and use that system as a starting point. This makes the search space somewhat different than the other systems discussed so far, in that every combination of features that can be expressed actually exists in the system.

Figure 14 shows the system after the user has asked to look at a systems around $1500. The bottom part of the screen has button to alter the parameters around which the configuration was built: the price tag, the room size, and particular components involved.

Our database in **Kenwood Home Theater** is not of individual stereo components and their features, but rather entire configurations and their properties. Although the database is large as indicated in Table 6, each entry in the database is very simple, just the price for the overall configuration, the constraints it satisfies, and a flag for each of the possible components. An adapted version of **Kenwood Home Theater** is currently on the web site for Kenwood, USA, and is accessible at <URL:http://www.mykenwood.com/Build/>. (Choose "Build System.")

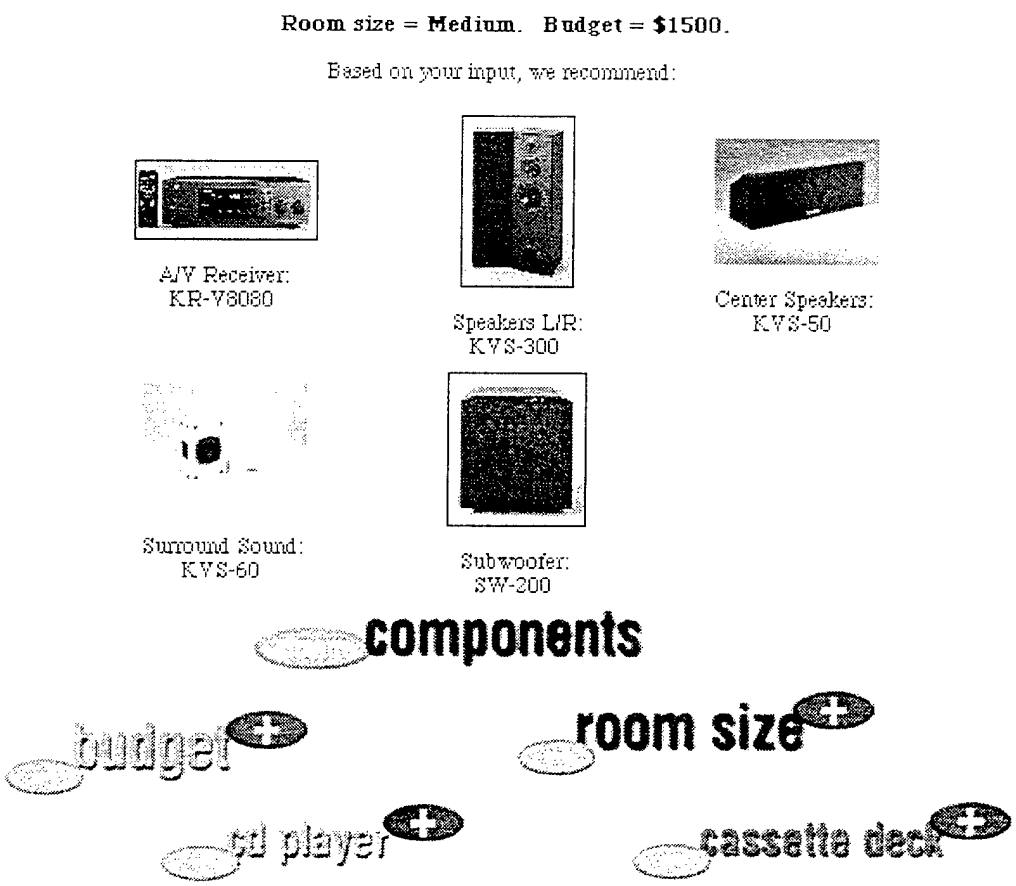

Figure 14: Results retrieved by the Kenwood Home Theater system.

System	Kenwood Home Theater
Platform	Web (Sun Solaris)
Language	C++ (2800 lines)
	perl (2700 lines)
Database	mSQL
Data Size	3 MB (340k configurations)
URL	http://www.mykenwood.com/Build

Table 6: Implementation details for Kenwood Home Theater.

Related Work

The problem of navigating through complex information spaces is a topic of active interest in the AI community. (See, for example, (Burke, 95; Hearst & Hirsch, 96; Knoblock & Levy, 95)). Much of this research is directed at browsing in unconstrained domains, such as the World-Wide Web, where pages can be on any topic and users' interests are extremely varied. As a result, these systems must use knowledge-poor methods, typically statistical ones.

Our task in **Find Me** systems is somewhat different. We expect users to have highly-focused goals: such as finding a suitable apartment to rent. The data being browsed all

represents the same type of entity, in the case of **Rent Me**, apartment ads. As a result, we can build substantial, detailed knowledge into our systems that enables them to identify trade-offs, compare entities in the information space, and respond to user goals. All of these properties make **Find Me** systems more powerful than general-purpose browsing assistants.

In the area of information retrieval, browsing is usually a poor cousin to retrieval, which is seen as the main task in interacting with an information source. The metrics by which information systems are measured do not typically take into account their convenience for browsing. The ability to tailor retrieval by obtaining user response to retrieved items has been implemented in some information retrieval systems through relevance feedback (Salton & McGill, 83) and through retrieval clustering (Cutting *et al*, 92).

Our approach differs from relevance feedback approaches in both explicitness and flexibility. In most relevance feedback approaches, the user selects some retrieved documents as being more relevant than others, but does not have any detailed feedback about the features used in the retrieval process. In other **Find Me** systems, feedback is given through the use of tweaks. The user does not say "Give me more items like this one," the aim of relevance feedback systems, but instead asks for items that are different in some particular way.

Examples have been used as the basis for querying in databases since the development of Query-By-Example (Ullman, 88). Most full-feature database systems now offer the ability to construct queries in the form of a fictitious database record with certain features fixed and others variable. The RABBIT system (Williams *et al*, 82) took this capacity one step further and allowed retrieval by incremental reformulation, letting the user incorporate parts of retrieved items into the query, successively refining it. Like these systems, **Find Me** uses examples to help the user elaborate their queries, but it is unique in the use of knowledge-based reformulation to redirect search based on specific user goals.

Another line of research aimed at improving human interaction with databases is the "direct query" approach (Schneiderman, 94). These system use two-dimensional graphical maps of a data space in which examples are typically represented by points. Queries are created by moving sliders that correspond to features, and the items retrieved by the query are shown as appropriately colored points in the space. This technique has been very effective for two-dimensional data such as maps, but only when the relevant retrieval variables are scalar values representable by sliders.

Like **Find Me**, direct query approach has the benefit of letting users discover trade-offs in the data because users can watch the pattern of the retrieved data change as values are manipulated. However, direct query systems have no declarative knowledge about trade-offs, and cannot explain to users how they might modify their search or their expectations in light of the trade-off. Also, as we found in **Car Navigator**, direct manipulation is less effective when there are many features to be manipulated, especially when users may not be aware of the relationships between features.

Our use of knowledge-based methods to the retrieval of examples has its closest precedent in retrieval systems used in case-based reasoning (CBR) (Hammond, 89; Kolodner, 93; and Riesbeck & Schank, 89). A case-based reasoning system solves new problems by retrieving old problems likely to have similar solutions. Because the retrieval step is critical to the CBR model, researchers in this area have concentrated on developing knowledge-based methods for precise, efficient retrieval of well-represented examples. For some tasks, such as case-based educational systems, where cases serve a variety of purposes,

CBR systems use a variety of retrieval strategies that measure similarity in different ways (Burke & Kass, 95).

Conclusion

Find Me systems perform a needed function in a world of ever-expanding information resources. Each system is an expert on a particular kind of information, extracting information on demand as part of the user's exploration of a complex domain. In **Find Me** systems, users are an integral part of the knowledge discovery process, elaborating their information needs in the course of interacting with the system. One need only have general knowledge about the set of items and only an informal knowledge of one's needs; the system knows about the tradeoffs, category boundaries, and useful search strategies in the domain.

Robustness in the face of user uncertainty and ignorance is another important aspect of **Find Me** systems. Most people's understanding of real world domains such as cars and movies is vague and ill-defined. This makes constructing good queries difficult or impossible. We believe therefore that an information system should always provide the option of examining a "reasonable next piece," of information, given where the user is now. These next pieces are derived through the application of retrieval strategies.

References

Buckley, C. 1985. Implementation of the SMART Information Retrieval Retrieval [sic] System. Technical Report 85-686, Cornell University

Burke, R., & Kass, A. 1995. Supporting Learning through Active Retrieval of Video Stories. *Journal of Expert Systems with Applications*, 9(5).

Burke, R. (ed.) 1995. *Working Notes from the AAAI Fall Symposium on AI Applications in Knowledge Navigation and Retrieval*, AAAI Technical Report FS-95-03.

Burke, R., Hammond, K., Cooper, E. 1996. Knowledge-based information retrieval from semi-structured text. In AAAI Workshop on Internet-based Information Systems , pp. 9-15. AAAI.

Collins, A. M. and Quillian, M. R. 1972. How to Make a Language User. In E. Tulving and W. Donaldson, Organization of Memory . New York: Academic Press.

Cutting, D.; Pederson, J. O.; Karger, D.; and Tukey, J. W. 1992. Scatter/Gather: A cluster-based approach to browsing large document collections. In *Proceedings of the 15th Annual International ACM/SIGIR Conference*, 318-329.

Cutting, D., Kupiec, J., Pederson, J., Sibun, P. 1992. A Practical Part-of-Speech Tagger. In Proceedings of the Third Conference on Applied Natural Language Processing . ACL.

Grady Ward, 1993. Moby Part-of-Speech II . Computer file. Arcata, CA: Grady Ward.

Hammond, K. 1989. *Case-based Planning: Viewing Planning as a Memory Task.* Academic Press. Perspectives in AI Series, Boston, MA.

Hearst, M. & Hirsch, H. *Working Notes from the AAAI Spring Symposium on Machine Learning in Information Access*, AAAI Technical Report SS-96-05.

Knoblock, C. & Levy, A. (eds.) 1995. *Working Notes from the AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments*, AAAI Technical Report SS-95-08.

Kolodner, J. 1993. *Case-based reasoning*. San Mateo, CA: Morgan Kaufmann.

Kulyukin, V., Hammond, K. Burke, R. 1996. Automated analysis of structured on-line documents. In AAAI Workshop on Internet-based Information Systems , pp. 78-86. AAAI, 1996.

Lang, K. L.; Graesser, A. C.; Dumais, S. T. and Kilman, D. 1992. Question Asking in Human-Computer Interfaces. In T. Lauer, E. Peacock and A. C. Graesser Questions and Information Systems (pp. 131-165). Hillsdale, NJ: Lawrence Erlbaum Assoc.

Lenhert, W. 1978. The Process of Question Answering . Hillsdale, NJ: Lawrence Erlbaum Assoc.

Miller, G. A. 1995. WordNet: A Lexical Database for English. Communications of the ACM , 38(11).

Quillian, M. R. 1968. Semantic Memeory. In Semantic Information Processing , Marvin Minsky, ed., pp. 216-270. Cambridge, MA: MIT Press.

Riesbeck, C., & Schank, R. C. 1989. *Inside Case-Based Reasoning*. Hillsdale, NJ: Lawrence Erlbaum.

Salton, G., & McGill, M. 1983. *Introduction to modern information retrieval*. New York: McGraw-Hill.

Schank, R.C., & Riesbeck, C. 1981. *Inside Computer Understanding: Five Programs with Miniatures*. Hillsdale New Jersey: Lawrence Erlbaum Associates.

Schneiderman, B. 1994. Dynamic Queries: for visual information seeking. *IEEE Software* 11(6): 70-77.

Souther, A.; Acker, L.; Lester, J. and Porter, B. 1989. Using view types to generate explanations in intelligent tutoring systems. In Proceedings of the Eleventh Annual conference of the Cognitive Science Society , pp. 123-130. Hillsdale, NJ: Lawrence Erlbaum Assoc.

Ullman, J. D. 1988. *Principles of Database and Knowledge-Base Systems Vol 1*. Computer Science Press, 1988.

Wiener, T. 1993.*The Book of Video Lists*. Kansas City: Andrews & McMeel.

Williams, M. D., Tou, F. N., Fikes, R. E., Henderson, T., & Malone, T. 1982. RABBIT: Cognitive, Science in Interface Design. In *Fourth Annual Conference of the Cognitive Science Society*, pp. 82-85. Ann Arbor, MI: