A dynamic intelligent frequency based search engine

Troy Saulnier
André Trudel
Jason Zwicker
Jodrey School of Computer Science
Acadia University
Wolfville, NS, B0P 1X0, Canada

Abstract

We present a search engine that can be either used with a database or closed website. The main feature of the search engine is that it keeps track of record, keyword, and query frequency usage and uses this information to modify and improve its performance over time.

Keywords: Search engine, useage frequencies, dynamic responses.

1 Introduction

Consider the following call center scenario. A customer calls with a question that the Customer Service Representative (CSR) cannot directly answer. The CSR must search a very large database for the answer. Restrictions on the search are:

1. The results of the search must be returned instantaneously. The caller is waiting and it is important to minimize the duration of the call.
2. The CSR must be able to sift through the results of the search quickly to determine the correct result.
3. Customers should be given consistent replies when dealing with different CSR's.

This call center scenario generalizes to the problem of efficiently and intelligently searching a large database or closed web site on an intranet.

We present a search engine that keeps track of record, keyword, and query frequency usage and uses this information to modify and improve its performance over time. After the user enters a query, the search engine looks for previously answered queries that are similar to the current query. They are shown to the user and if the user chooses one of the similar queries, he/she is immediately shown the location of the answer in the database. Otherwise, the user is shown a ranked list of records that possibly contain the answer. For each record, its first 80 characters of text is displayed. If the user cannot find an appropriate record or too many records are shown, the user can refine the search. The user will be shown a restricted list of keywords that are related to the query. The user has the option of adding/removing keywords in order to refine the search. Once the search is successful, the search engine will keep track of the record and keywords used to answer the query, and the query itself. This data is used to modify its future performance.

In the next section, we describe the back end files used by the search engine. We then describe the search algorithm. We conclude with a description of the system testing.

2 Indexes and cross-indexes

Our search engine requires extensive indexes and cross-indexes of the database. Specifically, it requires information about the keywords, records, and queries. This additional information is stored in files which are external to the database. Alternatively, this information could be stored in a separate relational database.

To construct the indexes and cross-indexes, we assume each record in the database can be identified by a unique integer. If we were to use a web site instead of a database, we could assign a unique integer to each page.

2.1 Keyword file

Every word is a candidate keyword. Since our problem domain consists of a database or closed website, the number of keywords is large but finite.

To optimize the search, we minimize the number of keywords using standard information technology techniques.[1] We first remove all one, two and three character words. We then plot a frequency graph for the remaining words. In the graph, the point (x, y) means there are y words that occur exactly x times in the database. The general shape of the graph is $1+e^{-x}$ (i.e., the majority of words occur infrequently). We use the graph to determine cutoff points. All words that occur infrequently (e.g., less than 4 times) or frequently (e.g., more than 300 times) are cut. We then combined singular and plural versions of the same word (i.e., by stripping off "s" suffixes). Since the database is in French, we also combined same words of different gender (i.e., by stripping off "e" suffixes). The final step of the keyword pruning process involves applying a stop list[2] to the keywords. A stop list contains words that are typically useless as keywords (e.g., "when").

For each remaining keyword k we store the following:

$$k \quad count \quad (r_1, c_1), \ldots, (r_n, c_n)$$

where *count* is the number of times k appears in the database. There is a pair (r_i, c_i) for each record r_i that k appears in. c_i is the number of times a query was successfully answered using k and r_i. c_i is initialized to zero and $0 < n \leq count$.

2.2 Record file

For each record number r we store the following:

$$r \quad count \quad k_1, \ldots, k_n \quad [text]$$

where *count* is the number of times record r was used to successfully answer a query. *count* is initialized to zero. $k_1, \ldots, k_n$ are the keywords contained in r. If r has no keywords, we store:

$$r \quad 0 \quad [text]$$

(*count* will always remain at zero). *text* is the first 80 characters contained in r.

2.3 Query file

There is no fixed format for queries. They can consist of a sequence of keywords, or a grammatically correct sentence with punctuation. For every successfully answered query we store the following:

$$(query) \quad r \quad count \quad k_1, \ldots, k_n$$

where *query* is the actual query as entered by the user. r is the record number which contains the answer to the query. *count* is the number of times *query* was asked by the user. The keywords associated with *query* are $k_1, \ldots, k_n$ where $n > 0$. Let Q and R be the set of keywords appearing in *query* and r respectively: $Q \subseteq \{k_1, \ldots, k_n\} \subseteq R$.

3 Search algorithm

In this section, we describe how our search engine deals with a query. After a query is typed in, we extract its keywords $\{k_{q1}, \ldots, k_{qi}\}$. Because we are only interested in keywords, the grammatical structure of the query is irrelevant.

We then go to the query file and find all entries

$$(query) \quad r \quad count \quad k_{qf1}, \ldots, k_{qfn}$$

such that $\{k_{q1}, \ldots, k_{qi}\} \subseteq \{k_{qf1}, \ldots, k_{qfn}\}$. This gives us a list of previously answered queries which are in a sense similar to the current query. They are sorted in decreasing order by *count* and displayed to the user. If the user considers one of these queries to be identical or very similar to his/her query, the record containing the answer is displayed and the appropriate counters in the keyword, record, and query files are incremented.

Otherwise, the current query is not similar to any previously answered query. We go to the record file and find all entries

$$r \quad count \quad k_{r1}, \ldots, k_{rj} \quad [text]$$

such that $\{k_{q1}, \ldots, k_{qi}\} \subseteq \{k_{r1}, \ldots, k_{rj}\}$. This gives us the records that possibly contain the answer. They are sorted in decreasing order by *count* and the *text* field is displayed to the user. The user can request that the full contents of any of these records be displayed. If the correct record is found, the keyword, record, and query files are updated.

Otherwise, the user can refine their query. Two keyword lists are presented to the user. The first consists of $\{k_{q1}, \ldots, k_{qi}\}$ and the second contains the alphabetically sorted union of all the keywords that appear in a record r where

$$r \quad count \quad k_{r1}, \ldots, k_{rj} \quad [text]$$

and $\{k_{q1}, \ldots, k_{qi}\} \subseteq \{k_{r1}, \ldots, k_{rj}\}$. The first list of keywords are the current list of keywords being used

[1] In our application, we reduced the number of keywords by 90%.

[2] Our database application was French. Unfortunately, there are no publicly available French stop lists. We had to create our own.

1. We hypothesize that our search engine improves its performance and accuracy over time. These improvements are achieved via gathering useage data.
2. The search engine modifies its future behavior based on its knowledge of the past (e.g., record, query, and keywords used). We tested this hypothesis in a real world setting.
3. Our search engine was used in a call center during live customer calls. With less than 10 minutes training, the CSRs were able to use our system in real time without any noticeable call delays.

Figure 1: Sample database

call 3 (3,0)
center 1 (3,0)
knowledge 1 (2,0)
performance 1 (1,0)
real 2 (2,0), (3,0)
search 3 (1,0), (2,0), (3,0)

Figure 2: Keyword file

for the query. The second list are the only possible keywords that can appear with the first list. Note that the user does not have to deal with all the keywords. The user can generalize the search by moving a keyword from the first list to the second. Moving a keyword from the second to the first list refines the search. When the user has finished moving keywords across lists, we return to the above step where text from each relevant record is displayed.

4 Example

A small database where record numbers appear to the left of each record is shown in figure **??**. Figure **??** contains a portion of the keyword file and the corresponding record file is (only 10 characters of text is used) is shown in figure **??**. Initially, the query file

1 0 performance, search [We hypothe]
2 0 knowledge, real, search [The search]
3 0 call, center, real, search [Our search]

Figure 3: Record file

(Does the search engine work ...) 3 1 real, search

Figure 4: Query file

call 3 (3,0)
center 1 (3,0)
knowledge 1 (2,0)
performance 1 (1,0)
real 2 (2,0), (3,1)
search 3 (1,0), (2,0), (3,1)

Figure 5: Updated keyword file

is empty. If the first query posed is "Does the search engine work in real time?", the keywords contained in the query are "real" and "search". We go to the record file and look for records containing both keywords. The first 10 characters of records 2 and 3 are shown to the user. The user can request that these records be displayed. After the user chooses record 3 as the correct answer, the files are updated as shown in figures **??**, **??**, and **??**.

If the next query is simply the keyword "real", we go to the query file and display the query "Does the search engine work in real time?" since it contains the keyword "real". If this is the information the user is seeking, record 3 is displayed to the user and the files are updated. Otherwise, we go to the record file and find all records containing "real". Records 2 and 3 are shown to the user. We proceed as with the previous query at this point.

Due to space limitations, we cannot show actual screen shots or a query refinement.

5 Testing

We hypothesize that our search engine improves its performance and accuracy over time. These improvements are achieved via gathering useage data. The search engine modifies its future behavior based on its knowledge of the past (e.g., record, query, and keywords used). We tested this hypothesis in a real world setting. Our search engine was used in a call center during live customer calls. With less than 10 min-

1 0 performance, search [We hypothe]
2 0 knowledge, real, search [The search]
3 1 call, center, real, search [Our search]

Figure 6: Updated record file

utes training, the CSRs were able to use our system in real time without any noticeable call delays. Unfortunately, the number of calls received during the seven days of testing was low. The system needs to be tested over a significantly longer period of time in order to measure a large performance change.

We were surprised to observe that our system, which was not optimized for speed and initially had no useage data, performed as fast as the currently used commercial product.

6 Related Work

One of our goals was to provide a more functional interface to the large database used by the CSRs. Our first iteration of this bears some resemblance to dynamic queries [1], in that users are empowered to quickly alter their queries on the fly based on feedback from the system. While our interface is keyword based as apposed to being graphical, the basic concept of allowing many queries to be executed quickly based on the systems instant feedback to the user still holds.

We also used the success, failure, and frequency of queries [2], as well as the fact that organizational knowlegde was being captured and refined [3], to improve the performance of the engine over time.

Our search is local in the sense that we do not consider links emanating to or from a page as is done in the Google search engine (www.google.com). Instead, we use frequencies to focus our search.

7 Conclusion

We presented a search engine suitable for closed indexed collections of data (i.e., either a database or website). The main features of the search engine are that it keeps track of queries, records, and keywords used, and uses this information to modify its performance over time. Advantages of our system are:

- It is easy to produce reports and statistics on most frequently used queries, records, and keywords. This is crucial information for a call center. Most frequently used queries, records, and keywords are knowledge that must be added to the training curriculum for new call center employees. Also, the most frequently used records are an area of the information base that must be kept up to date, and accurate.
- If many users (i.e., the whole call center) use the search engine, then frequency statistics and queries are gathered for the whole group. This means users, especially novice ones, benefit from the experience of everyone else.
- By reusing old queries, we get uniformity of answers across users.
- The system is ideal for problem domains where queries are reused often.

Future work includes:

- The current prototype is a standalone system. We need to modify it to accomodate multiple users.
- The query, record, and keyword files are not automatically updated whenever the database is modified.
- The current prototype is designed for a database. We will rewrite it for a website.
- We will investigate the feasibility of storing the information in the query, record, and keyword files in a relational database.
- Many performance improvements can be achieved by optimizing the search engine's internal data structures.
- Investigate and test different ranking strategies for displaying queries and records.
- Investigate and test temporal or rate based strategies for pruning the queries file when it becomes large.

Acknowledgements

This research was supported by an industrial research contract with Bell Mobility Cellular Inc.

References

[1] B. Scheiderman et. al., "Dynamic Queries: Database Searching by Direct Manipulation", *CHI '92*, pp. 669-670, 1992.

[2] M. Morita and Y. Shinoda, "Information Filtering Based on User Behavior Analysis and Best Match Text Retrieval", *Japan Advanced Institute of Science and Technology*, pp. 272-281, 1994.

[3] M. Ackerman, "Augmenting Organizational Memory: A Field Study of Answer Garden", *ACM Transactions on Information Systems*, Vol 16, No. 3, pp. 203-224, 1998.