Lazy Query Expansion[*]

Alexander Gelbukh

Center for Computing Research (CIC),
National Polytechnic Institute (IPN),
Av. Juan Dios Bátiz s/n esq. Mendizábal,
Col. Zacatenco, C.P. 07738, D.F., Mexico
gelbukh*cic.ipn.mx

Abstract

An information retrieval or document base system has to somehow deal with various phenomena of equivalence of some strings. These are lowercase versus uppercase matching, morphological inflection, derivation, and synonymy of words: e.g., given a query *computer*, find *Computers*, *computing*, *workstation*. The latter problems are very important in languages with richer morphology and less stable terminology than in English. Also, much better recall is achieved by matching hyponyms and hypernyms using a thesaurus, e.g., given a query *computers*, find also *supercomputer*, *microcomputer*, *mainframe*, *machine*, *device*, *processor*, *UNIX*, etc. Technically, this can be handled at the time of indexing by reducing related strings to a common form, or at the time of query processing by expanding the query with the whole set of the related forms. We argue for that the latter way allows for greater flexibility and easier maintenance, while being more affordable than it is usually considered. We propose to expand the query with only those words that really appear in the document base. Our experiments with a thesaurus-based information retrieval system we are developing for the Senate of Mexican Republic show only insignificant increase of the real user queries on average with the 200-megabyte document base of the Senate, in spite of highly inflective Spanish language.

Keywords: full-text database, information retrieval, query expansion, natural language.

[*] An extended version of the paper *Lazy Query Enrichment: A Simple Method of Indexing Large Specialized Document Bases*, In Proc. DEXA-2000, 11th International Conference and Workshop on Database and Expert Systems Applications, Lecture Notes in Computer Science, Springer-Verlag, 2000.

Work done under partial support of the Senate of Mexican Republic (unnumbered project), Mexican Government (CONACyT and SNI), and CGPI-IPN (Mexico).

1. Introduction

Nearly any information retrieval system has to somehow deal with the problem of non-literal matching of the query and the document keywords. For example, given a query *computer*, the system should be able to retrieve (or not, depending on the user-defined settings and query options) the documents containing the strings *Computer*, *computers*, *computation*, *mainframe*, *motherboard*, *Internet*, etc. There are two places in the system architecture where this problem can be dealt with:

- at the moment of indexing the documents—*index expansion*—or
- at the moment of processing of the specific query—*query expansion*.

The former technique is most commonly used due to apparently prohibitively serious problems caused by the latter one. We will show, however, that the former method has its own disadvantages, and that the problems of the latter method can be efficiently solved.

Our main motivation in this work was the development of an information retrieval system for the Senate of Mexican Republic. The document base of the Senate contains the laws of the Mexican Republic, the bills under consideration in the commissions of the Senate, the protocols of the sessions, the discourses of the Senators, etc. Our customer formulated the order of the priorities as follows.

- The *quality*, expressive power, and flexibility were the main priority due to the importance of the search results for the legislation of the Senators.
- Small *size of the index* and reasonably low maintenance load on the server were the second priority according to the hardware resources available for the system.
- No or *minimal changes* to the existing technology of the maintenance of the database and its structure were to be introduced.
- The system was to be operational while the dictionaries and grammars were under development, and the im-

provements and corrections to them were to be immediately available to the users.
- Computational efficiency of processing of a single query was of lower priority since the number of the users—the Senators and their aids—is rather limited.

The characteristic properties of the document base at hand were the following:

- Large size, in the order of a gigabyte, to be extended to several gigabytes,
- Specialized contents with limited variety of lexicon and syntactic constructions,
- Still, unrestricted language with the possibility of occasional use of nearly any word or word form.

In this work, we are interested in a flexible, computationally efficient, conceptually simple, and easily maintainable solution of the problem of non-literal matching under the requirements and circumstances listed above.

1.1 Related work

There is a vast literature on approximate string matching; various data structures, such as tries, B-trees, etc. were suggested (Aho, 1990; Gusfield, 1997; Frakes & Baeza-Yates, 1992). These works are based on implicit or explicit patterns that describe the similarity between the source string and the matched strings (e.g., minimal editing distance) or the set of the strings to be found (e.g., regular expressions), at the level of individual letters. For example, a pattern *com** can be used to search for all forms of the Spanish word *comer* 'to eat,' though this pattern will also match 172 other Spanish words like *cometa* 'comet.' However, in our case we consider the problem of matching arbitrary word sets that might not share any simple letter pattern. For example, the strings *dormía*, *duermo*, and *durmiendo* are forms of the same Spanish verb *dormir* 'to sleep;' the strings *church*, *priest*, and *pilgrim* represent the same English concept *religion* though they do not match any particular letter pattern for approximate string matching to be applied.

The problem of generating and matching the word forms in various languages, including English and Spanish, is well studied in linguistics. Various methods and data structures are suggested in computational linguistics for handling the corresponding dictionaries and morpheme lists (Gelbukh, 2000; Hausser, 1999; Koskenniemi, 1983). However, in this article we do not discuss the problems of natural language morphology. Instead, we are interested in application of a morphological analyzer to the purely database management task of retrieval of a keyword in all its forms. The list of the word forms is supposed to be already known while these forms are not supposed to match any particular letter pattern.

The use of concept hierarchies for topical document analysis was suggested in (Guzmán-Arenas, 1998) and applied to the information retrieval tasks in (Gelbukh *et al.*, 1999). Various large hierarchical thesauri have been compiled (Cassidy, 2000; Fellbaum, 1998; Lenat *et al.*, 1990). However, here we are interested not in the handling of the statistical weights nor in compilation of the concept dictionary, but rather in the way the documents relevant for a specific node can be in practice found in an existing large information system.

2. Types of non-literal string matching

Here we will discuss in more detail the types of the strings that the user might want to be matched. An important point in each case is the great degree of flexibility necessary to meet the requirements of a specific user or a specific search.

2.1 Letter case

This is the simplest type of non-literal matching: usually the strings like *computer*, *Computer*, *COMPUTER*, and *ComPuTer* are to be considered equivalent. The designers of information retrieval systems tend to consider it obvious that before indexing the database, all words are to be automatically converted to, say, lowercase.

However, under certain circumstances, the user might want to search for a specific string such as *Bill* or *Mainframe* (personal names) but not *bill* nor *mainframe*, *CIS* (the name of an organization) but not *Cis* (personal name), *CYCLing* (the name of a conference) but not *cycling* nor *Cycling*.

2.2 Morphology

The second class of strings that frequently are considered equivalent are the word forms of the same lexeme: *compute*, *computing*, *computed*, or its derivational variants: *computer*, *computation*, *computational*, *computability*.

Such equivalence is determined by linguistic software—stemmer or, more generally, morphological analyzer (Hausser, 1999; Koskenniemi, 1983, Gelbukh, 2003 ###). For each word, it provides (possibly ambiguously) an identifier—a normal form like *compute*, a stem or root like *comput-*, a number, etc.—that is common for all such equivalent word forms. Then matching of the two strings consists in reducing, or *normalizing*, them to such identifiers and comparing the results.

Morphological analyzers can be of different degree of complexity, which depends on the language of the documents, on the desired precision, and on whether only inflection within one lexeme (*compute/computed*) is to be taken into account or also derivation (*compute/computer*) word formation (*computer/uncomputability*). In the simple case, such an analyzer can use a simple list of endings, such as *-s*, *-ed*, *-ing*, *-er*, *-ability* and a small list of exceptions, such as *go*, *goes*, *went*, *gone*. For highly inflective languages such as Spanish, the list of endings can be quite large, currently 3451 endings being used in our system. A more sophisti-

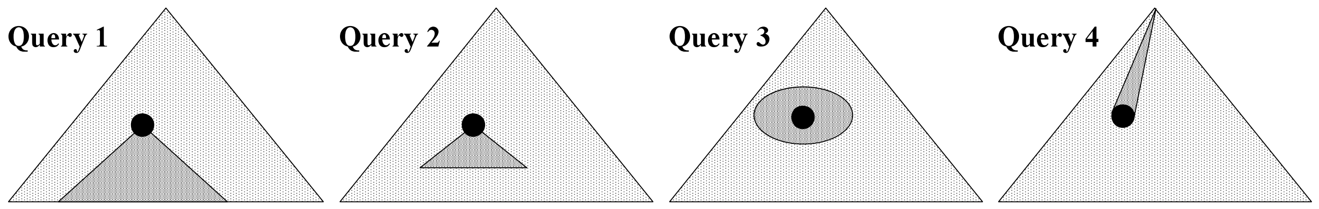


Fig. 1. Types of neighborhoods in a hierarchical thesaurus.

cated—and thus more precise—morphological system can use complex patterns and/or rely on a large dictionary. However, even a dictionary-based system must contain a heuristic algorithm for handling the words absent in the dictionary.

Note that heuristic-based morphological algorithms perform much better on analysis (normalization) than on synthesis (generation) of the word forms corresponding to a given stem. For example, a simple list-based algorithm can normalize *uncomputability* to *-comput-*, however, given a stem *-comput-*, it is more difficult to make a choice between **incomputibility*, **ircomputibility*, etc.

Matching the morphological forms of the same stem is not always desirable for the user. For example, the user can be interested in *computers*, but not in *computation*. Very annoying can be morphological reduction of ambiguous forms, especially in highly inflective languages such as Spanish. For instance, the Spanish verb *comer* 'to eat' form about 700 morphological variants like *comiste* 'you ate' or *comiéndotela* '(you) eating it up', one of which—namely *como* 'I eat'—happens to be homonymous with a very frequently used conjunction *como* 'as,' 'how.' Thus, to find the documents with the Spanish lexeme *comer* with a reasonable precision, one has to sacrifice recall a little bit by forming the query as "all word forms of *comer* but *como*."[1]

Thus, the user should be able to control the application and the degree of morphological normalization applied to the query by the system.

2.3 Concept hierarchy

The third class of the words that might be considered equivalent are synonyms (*processor*/*CPU*), hyponyms (*computer*/*mainframe*), hypernyms (*computer*/*device*), and possibly other related words. Since no algorithm can infer such relationships between words on its own, a dictionary—namely, a hierarchical thesaurus—is used to provide this option to the user.

[1] Note that this effect cannot be achieved with a simple logical expression "all documents containing the word forms of *comer* but those containing the word *como*" since its meaning is not equivalent to the desired one. Namely, the recall with such a query would be extremely low since nearly any Spanish document does contain the string *como* 'as,' 'how.'

In our system, we use a 33,000-word dictionary organized in a deep hierarchy of related concepts, similar to, and in part derived from, the Roget thesaurus (Cassidy, 2000). By related concepts, we mean not only the *is-a* relationship, but also other words that are of interest to the user looking for the documents on a given topic (Guzmán-Arenas, 1998). For example, the entry for *religion* contains such words as *Bible*, *priest*, *pray*, *church*, *pilgrim*, etc. Thus, the user looking for the documents on *religion* will be offered a document that mentions *Bible*. Optionally the degree of such relationship can be weighted quantitatively to measure the relevance of the found document (Gelbukh *et al.*, 1999).

Obviously, the user should have a full control over the set of words to be considered equivalent to the query keyword(s). The following options are of particular interest, see Fig. 1:

1. **All instances** of a given concept, i.e., all words below a given node. For example, with this type of query, given a query "Find everything on *mathematics*," the system should retrieve all documents on *algebra*, *geometry*, *calculus*, and within these topics, everything on *linear algebra*, *group theory*, ..., *differential geometry*, *foliation theory*, ..., *differential calculus*, ..., and finally retrieve the documents that mention the words *vectors*, *manifolds*, *differentials*, etc. Another example: "What events happened in *Europe*?" This query should be interpreted in such a way that the documents mentioning *England*, *Italy*, *Austria*, ..., *London*, *Manchester*, *Birmingham*, etc. be retrieved.
2. **Near-immediate instances** of a given concept, i.e., the words below a given node but not deeper than n levels. For example, a student might want to know what is mathematics: "Find documents on *mathematics* in general." In this case, the system does retrieve the documents that mention the words *equation*, *inequality*, *theorem*, but not *isostrophy* nor *semilattice*, the latter words being too specialized. Another example: "What is the politics of the European countries?" In this case, only the documents that mention *England*, *Italy*, *Austria*, ..., and probably *London*, *Rome*, *Vienna*, but not *Manchester*, *Birmingham*, etc. are to be retrieved.
3. **Similar** concepts, the words located in the concept tree not farther than n steps from the given one, be the steps in the down, up, or horizontal direction in the tree. For example, "What disciplines are similar to *mathematics*?"

In this case, the relevant words are *physics*, *astronomy*, *algebra*, *geometry*, etc.

4. **General** concepts, i.e., the words of which the given node is an instance. For example, "In what hemisphere is *Morelia* located?" In this case, the documents that might mention the hemisphere where *Michoacán*, or *Mexico*, or *North America* is located (Morelia being a city in the Michoacán state, Mexico). Another example: "What right an *Associate Professor* has?" In this case, useful documents can mention the rights of a *teacher*, *employee*, *citizen*, or *human*.

In fact, the constraints 1 to 4 often have to be combined. For example, in a type 4 query, a limit on the number of levels—as in the type 1 query—or on the most general level is useful, since too general concepts appear in too many documents and also scarcely provide any knowledge unknown to the user. Or, type 4 queries can be combined with type 1 or 2 queries. For example, for the query "In what hemisphere is *Michoacán* located?" both more general (as in type 4) and more specific (as in type 1) concepts are to be searched for. Note that, at least currently, the desired type of the query generalization cannot be inferred automatically by the system and should be chosen explicitly by the user.

3. Index expansion

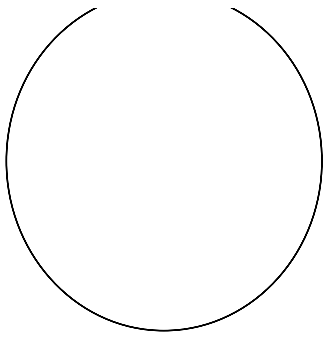
In the previous section, we discussed four cases of identity of the strings: letter case, morphologically inflected forms, synonyms, and a concept tree (type 1 queries, see Fig. 1). A naïve—and the most frequently used—approach to represent the first three cases of identity is *index reduction*: at the moment of indexing, all letters are reduced, say, to lowercase; all word forms are reduced to the main form (*computing*, *computed*, *computes, computation, computer* → *compute*), and synonyms are replaced with one chosen representative (*CPU* → *processor*). The latter case—a concept tree—can be handled by additionally indexing each document with the hypernyms of the words it contains (*mainframe* → *computer*, *device*, *artifact*); with this method, a query "*devices*" will retrieve also *mainframe*.

In this article, we argue that this naïve approach has serious disadvantages. First of all, as we have shown in the previous section, depending on the desired precision/recall ratio, the user might *not* want such strings to be considered identical. Thus, indexing process should not cause any loss of information—i.e., all the letter strings should appear in the index *as is*, without any change, even in the letter case (i.e., reducing to lowercase). To achieve this, the strings reduced in letter case, or morphologically, or by a thesaurus should appear in the index in addition to (rather than instead of) the original strings, e.g.: *Mainframes* → *Mainframes*, *mainframes*, *mainframe*, *computer*, *device*, *artifact*. Since the new keywords are added to the index instead of replacing the original ones, we call this process index expansion.

To allow for certain flexibility of the queries, some improvements to this indexing scheme can be suggested. For instance, the additional keys are to be marked somehow to be distinguishable from the original ones, e.g., *Mainframes* → *Mainframes*, CASE-*mainframes*, MORPH-*mainframe*, UP-*computer*, UP-*device*, UP-*artifact*. With this, a user query "exactly the string *Mainframes*" can be internally translated by the system into the SQL query "*Mainframes*"; the query "the word form *mainframes*" into "CASE-*mainframes*" that matches both *Mainframes* and *mainframes*; the query "the lexeme *mainframe*" into "MORPH-*mainframe*" that matches both *mainframe* and *mainframes*; the type 1 query "*devices*" into "UP-*device*" that matches both *mainframe* and *printer*.

Other possible improvements will be discussed in section 8. However, the index expansion method presents some inherent problems:

- *Lack of flexibility*. Only the types of queries for which the index was specifically designed can be executed. The user cannot choose what words of a given set are to be considered equivalent, e.g., "all word forms of *compute* but *computing*;" see also the discussion of the example with the Spanish *comer* in section 2.2, and also the footnote there.
- *Lager index*. Unlike index reduction, index expansion can significantly—from twice to tenfold, depending of the use of only morphology or also a thesaurus—increase the index size. In many cases, especially with large databases, this is not affordable.
- *Maintenance difficulties*. Too close coupling of the indexing process and potentially complex lingware—the morphological analyzer and the thesaurus—presents both organizational and technical problems.
  - Adding the intelligent search engine to a long-existing operational database maintenance technology requires significant changes in the latter, which implies changing existing software and documentation, training the maintenance engineers, etc. In our case, preserving intact the existing technology was one of the strongest requirements of the customer.
  - Unlike stable database maintenance procedures, complex dictionary-based lingware tends to be—at least for a certain period of time—in constant development: new words are added to the dictionary, the morphological tables and algorithms are corrected, new links are added to the thesaurus. With index reduction or expansion, each time a change is made to the lingware, the whole database is to be re-indexed, which is often not affordable, especially when the linguistic processing is slow and resource-consuming. On the other hand, delaying re-indexing for a long time greatly discourages any improvements to the lingware,



from the point of view of both the developers and the customer.

4. Query expansion

An alternative to handling non-literal string matching at the moment of indexing is handling it at the moment of query processing. A naïve approach to this method is the following. The letter strings found in the documents are indexed *as is*, without any changes. Then, at the moment of query processing, the user query is automatically substituted with an appropriate logical expression, e.g., the query "*compute* and *matrix*" internally is executed as "(*compute* OR *computes* OR *computed* OR *computing*) and (*matrix* OR *matrixes* OR *matrices*)." This procedure is a variant of so-called query expansion (Kowalski, 1997, Voorhees 1998).

This method does not present any of the problems listed in the previous section. Namely, it has the following advantages over index expansion or reduction:

- *Flexibility*. The user can edit the resulting expression (say, by checking or unchecking the checkboxes next to each generated form) to achieve any desired combination. For example, the query "all forms of the Spanish verb *comer* but *como*" can be naturally expressed by the user and processed by the system.
- *Smaller index* as compared with index expansion. Only the strings literally present in the document are present in the index.
- *Easy maintenance*. The indexing procedure is trivial and does not include, nor depends on, any lingware. No changes to the existing non-intelligent indexing technology are necessary when adding an intelligent search engine to an operational database. No re-indexing is necessary when changes are made to the lingware, and such changes are available immediately to the user.

However, the disadvantages of this naïve approach are so obvious that it cannot be considered a practical option. Namely, the following two problems render such a method unusable:

- *Too large queries*. As we have mentioned, the Spanish verb *comer* form about 700 variants, which results in too large query. With a thesaurus, the concept *Europe* would contribute to the query all countries, cities, rivers, mountains, nations, types of food, etc. specific for Europe. In addition, each of these strings should be capitalized in all possible ways.
- *Generation*. As we have mentioned in section 2.2, generating all forms of a given lexeme (*compute* → *compute*, *computing*, ..., *uncomputable*, ...) is a task significantly more difficult than guessing the correct main form or stem of a given word form (*compute*, *computing*, *uncomputable* → *compute* OR -*comput*-). In case of a heuristic-based morphological algorithm, the number of hypotheses in form generation is usually much greater than in reducing to the stem.

However, limited version of this approach (similar to Type 2 expansion, see Fig. ###) has been tested, with promising results. Voorhees (###) reports that semantic query expansion using WordNet (even in such a limited form) significantly improves the results in terms of precision and recall, especially when the query is short. Fig. ## approximately shows the figures reported by Voorhees.

####In the next section, we will show how these problems can be worked around.

5. Lazy query expansion

First, we will briefly discuss the common properties of languages that guarantee the applicability of our method, and then proceed to the method itself.

5. 1 Diversity of language

The diversity of language in a certain text collection is limited. Indeed, the most frequent words as repeated many times, not leaving much space for other words. These intuitive considerations are formalized by two empirical statistical laws called Zipf law and Heaps law.

To explain them, it is important to distinguish between words as types (i.e., different words, words as elements of the language) and running words, i.e., occurrences of a word in a specific text. For example, in the phrase *John loves Mary and Mary loves John* there are 4 different words (types) and 7 running words (occurrences). By frequency of a word, we mean the number of its occurrences in a given text.

The Zipf law (###) states that the most frequent words are *much* more frequent than the less frequent words. Given a text, denote $r(w)$ the statistical rank of a word w in this text, i.e., the number of different words which has higher frequency than w. Then the frequency f(w) of the word w is approximately

$$f(w) \approx \frac{C}{r(w)^z},$$

where C is a constant and z is near 1. Fig. ### graphically illustrates the real distribution of the frequencies ordered by the tank for a typical document, together with the approximation given by the formula: as one can see, only the most

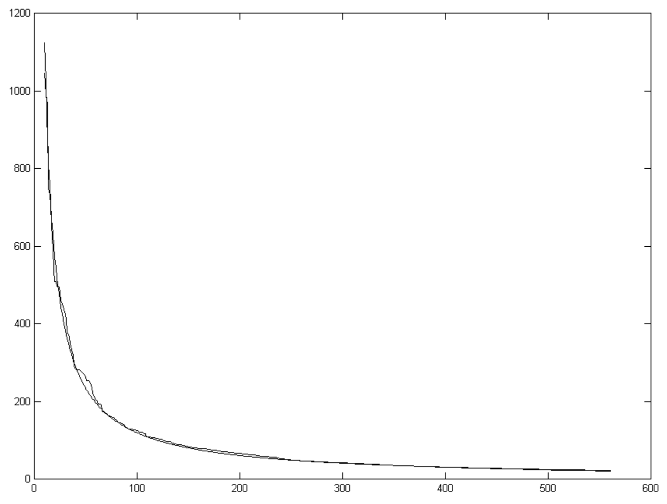


Fig. ###

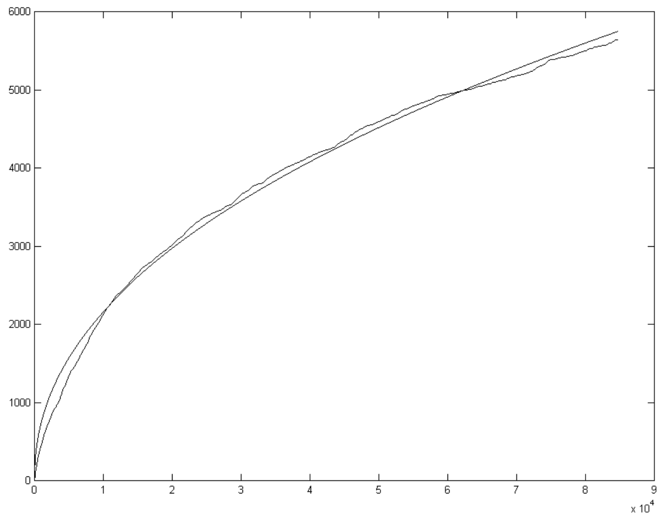


Fig. ###

frequent 300 lexemes appeared in this English document more that 50 times.

Presumably, one of the consequences of the Zipf law is the Heaps law (###). It states that the number of different words is a text is much less than the size of the text. Given a text, denote v(n) the number of different words whose first occurrence is the running word with the number less than n in the text. Then this value is approximately

$$v(n) \approx D n^h,$$

where D is a constant and h is reported in (###) to be in the range between 0.4 and 0.6. Fig. ### illustrates the distribution of the number of different words occurred before the n-th position for a typical document: as one can see, this 90,000-word English document contains only 6,000 different lexemes.

We have conducted some experiments to validate these laws. Unlike (###), in these experiments, we approximated the whole range of distribution (for Zipf coefficient, we ignored the first 10 ranks). Fig. ### shows a sample of the results for three English, Spanish, and Russian documents. All documents had the size of 100,000 running words or more. This figure also shows the average values for the corresponding languages found in our experiments. The average was calculated over 39 texts for English and Russian and 3 long texts for Spanish. In fact, Fig. ### presents the results for one of the documents from our collection.

Thus, the number of words in a large enough text is approximately proportional to a square root of its size. What, however, is the number of different words in a language? There are different ways to answer this question. A large dictionary usually has about 200,000 words (lexemes, or stems). Using the values given in Figure ###, we can predict that an English texts needs to be at least 48 million words long for each of these words to appear at least once in it. However, together with scientific and special terms, the estimated number of words in a language is about 1 million, which gives 1 milliard word long text (10 GB).

However, one can count not different lexemes but different wordforms (strings). Even though the strings ***ask***, ***asks***, ***asked***, ***asking*** belong to the same dictionary entry ***ask***, they can be counted as separate types. In our experiment, we observed that 90,000 most frequent Russian words (lexemes) generate 2,234,000 different strings. Using again the values from Figure ###, one can see that a 36 million words long text is needed for each of these wordforms to occur at

	Zipf				Heaps			
	Wordforms		Lexemes		Wordforms		Lexemes	
	z	*C*	*z*	*C*	*h*	*D*	*h*	*D*
	0.96	9480.53	0.97	10638.11	0.50	21.74	0.45	32.08
English	0.98	12854.57	0.99	18140.46	0.61	5.40	0.58	6.41
	1.01	7431.42	1.00	7869.44	0.63	6.02	0.59	8.67
	1.04	3224.48	1.10	6107.22	0.72	2.59	0.62	4.83
Spanish	1.06	55944.81	1.14	110879.83	0.58	12.66	0.53	13.88
	1.04	3027.26	1.10	5589.66	0.65	5.49	0.57	8.83
	0.95	4764.25	0.90	5156.40	0.78	3.75	0.64	10.75
Russian	0.91	9170.19	0.99	16305.47	0.68	7.09	0.50	25.85
	1.02	12004.91	1.04	16610.19	0.77	2.74	0.69	4.20

Fig. ###

	Zipf				Heaps			
	Wordforms		Lexemes		Wordforms		Lexemes	
	z	*C*	*z*	*C*	*h*	*D*	*h*	*D*
English	1.00	15396.60	1.01	17335.70	0.57	12.61	0.53	17.24
Spanish	1.05	20732.20	1.12	40858.90	0.65	6.92	0.57	9.182
Russian	0.93	5162.33	0.96	7926.62	0.76	4.18	0.63	10.05

Fig. ###

String	ID	String	ID	String	ID
computer	computer	*computes*	compute	*mainframe*	computer
Computer	computer	*computing*	compute	*mainframe*	device
CompuTer	computer	*uncomputability*	compute	*mainframe*	artifact

Fig. 2. The table used for lazy query expansion.

least once. Thus, for a 200,000 words dictionary, 100 million words long text (1 GB) is needed. We expect similar results for Spanish.

Finally, in agglutinative languages like Turkish the number of wordforms is potentially unlimited, so in such a language the number of different words (counting wordforms, not stems) is infinite.

The conclusion is that even in a quite large text, only a small fraction of the words (stems or wordforms) potentially existing in the given language occurs.

It seems obvious that in specialized document collections (such as medical records or legal contracts) even a smaller part of all words potentially existing in the language occur (since the words of other language styles and topics do not occur in such collections), though we did not conduct the corresponding experiments.

5. 2 Lazy query expansion method

The improvement we suggest for the method of query expansion consists in including into the expanded query only the strings known to be present in at least one document of the given database. Since only a small fraction of all possible forms of a word or sub-concepts of a concept is present in the database, this greatly reduces the size of the expanded query. At the same time, when applied to the specific database, such a reduced query is equivalent to the fully expanded query. We call this modification of the expansion procedure lazy expansion.

The process of lazy query expansion can be sketched as follows.

- A list of all strings that appear at least once in the given database is compiled.
- This relatively small list is indexed as described in section 3, which produces an index table such as the one shown in Fig. 2.

  In this figure, by the identifier (ID) we mean a reduced form, such as reduced to the lowercase, morphologically reduced to the main form, promoted up the tree in the thesaurus, etc., see section 2 (we did not show in this table the improvements discussed in the sections 3 and 8).
- At the moment of processing the query, each keyword of the query is subject to an appropriate indexing process depending on the user-defined option, for example, to morphological reduction to its main form, e.g., *computable* → *compute*, thus giving a potential ID. In case of ambiguity, all potential IDs are obtained.
- The ID(s) for each query keyword are looked up in the right column of the table, and the word is substituted with the list of the corresponding literal strings found in the left column.

For example, with the table above, the query "what things are *computable*?" is transformed first into the query "ID = compute" and after the lookup in the table, into the query "*computes* OR *computing* OR *uncomputability*." Note that it does not contain such strings as *compute*, *computed*, nor the very source form *computable* since they do not occur in the documents in the database.

To process a complex query, such as, for example, type 3 query discussed in the section 2.3, the thesaurus is navigated correspondingly and the query is first built as a disjunction of the relevant lexemes or concepts as shown on Fig. 1. Then such a query is further expanded through the index table as described above.

The suggested modification of the query expansion method does not have any disadvantages of the latter, thus presenting the following advantages as compared with full query expansion:

- *Smaller queries*. Only the words really appearing in the database are included into the query. The difference is especially sensible in the languages with developed morphology. For example, of about 700 forms of the frequently used Spanish verb *comer* 'to eat,' in the database of the Senate of Mexican Republic appeared only 29, e.g., *comiéndose* 'being eaten,' *comérselo* 'to eat it up,' etc.; of more than 100 forms of *falsificar* 'to falsify,' appeared only 11, e.g., *falsificarla* 'to falsify her,' *falsificadas* 'those$_{\text{feminine}}$ being falsified,' etc.
- *Only reduction*. The algorithm does not use any generation; instead, only reduction—such as reduction to lowercase or morphological reduction—is used. This greatly simplifies the lingware, allowing for a rather simple heuristic-based morphological algorithm.

Of course, the suggested method still has some disadvantages as compared with the full query expansion or index expansion methods.

- *Need to maintain the list of strings*. As compared to the full query expansion, the suggested method requires to maintain an additional data structure. In the next section we will show that this does not present serious maintenance problems.

- *Still increase in query size.* As compared to the index expansion, the queries are still increased in size, though not as much as with full query expansion.
- *Options may look strange.* As compared to the full query expansion, the list of strings presented to the user for editing (see section 0) may look incomplete, especially if the user does not understand how the method works and why some word forms, e.g., *computes*, *computing*, and *uncomputability* are present in the list while other ones, e.g., *compute* or *computed*, are not.[2] This, however, should not be a serious problem.

6. System architecture

The success of application of our method depends on the index updating procedure, which we will describe below. Another interesting feature of our method is the possibility of gradual query expansion, which we describe in the next subsection.

6. 1 Updating the index

In the previous section, the necessity of maintenance of the list of strings and the index table was mentioned as a potential source of complications or undesired coupling of the indexing technology with the lingware. Here we will show how we avoid these problems in our system. There are two potential sources of problems:

- Updating the list of strings and the index table when the database changes, and
- Updating the index table when the lingware changes.

The latter point does not present any real problem since the list of the strings found in the database is very small in comparison with the whole database. Thus the index table is simply rebuilt from this list each time the lingware is changed, with no significant load on the system.

The former point is only slightly more difficult. To keep the existing database maintenance technology independent of the linguistic module that builds and uses the list of the strings, we use an independent process (an agent, or a daemon in UNIX terminology) that periodically (at time intervals depending on the current system load) synchronizes the list with the actual database. There are two possible ways to achieve such synchronization.

With one method, the database index is accessed by the agent and enumerated alphabetically. The agent re-builds the word list and compares it with the current one, thus detecting what words have been introduced and what ones have disappeared. The disappeared entries are removed from the table, and the new ones reduced (to the lowercase, morphologically, and with the thesaurus) and added to the table.

Another method requires an additional Boolean property of the document—*indexed*—to be kept in the database. When a new document is added to the database, this property is set to *false*. The agent periodically addresses the database with the query "*indexed* = *false*," retrieves some of the found documents (depending on current system load), extracts the letter strings from them, adds to the list and the table those ones that are not yet there, and marks the document as *indexed* = *true*.

The two methods have the following advantages and disadvantages:

- The second one allows treating the DBMS as a black box, while the first one requires the direct operation on its internal structures.
- The first one does not require any changes in the database maintenance technology used by the customer, while the second one requires a small change in the database structure.
- The second method allows indexing documents with the properties not related to individual words, but rather to specific word combinations or to the whole document, such as the main topic of the document (Gelbukh et al., 1999; Guzmán-Arenas, 1998).
- The second method does not provide a convenient way to detect deleted documents **and thus the words having disappeared from the index**.

The latter is not a serious problem since the strings present in the list but absent in the database do not affect the results of the search though do reduce system performance. One of the possible solutions to this problem is to periodically (say, once a month) rebuild the whole list. For this, the property *indexed* is made of the type *date* rather than *Boolean*. To rebuilt the list initiating the rebuild process, say, on 01-apr-2000, the agent retrieves the documents with "*indexed* < 01-apr-2000," analyzes found documents, and sets the property to "*indexed* = today."

6. 2 Overall system architecture. Partial expansion

Our system is built on top of the existing operational technology treated as a black box. The information flow is intercepted in the three points:

- The user query is intercepted, analyzed, and substituted with an expanded query using the lazy query expansion technique. The new query is presented to the user in a user-friendly form for possible editing. If necessary, the expanded query is broken down into a series of smaller queries (see below).

[2] Note that the list cannot be completed with the words absent in the database since the heuristic-based morphological algorithm being used for reduction is not designed for generation of all possible word forms.

- The response of the DBMS is intercepted, analyzed, and—in case of a broken query—one response is compiled of several partial query results.
- At the moments of low system load, an agent periodically analyzes the database to update the word list and thus the substitution table used for lazy query expansion.

To improve performance in cases when the expanded query is too large, the query is expanded partially with the words that are most closely related to the original user query. For example, in type 1 query discussed in section 2.3, first of all reducing to lowercase and then morphological reducing is tried. Only in the case if such a partially expanded query does not result in a sufficient number of found documents, the type 2 expansion is tried, see Fig. 1. If this query is not sufficient, the fully type 1 expanded query is performed. As soon as, however, a partial query results in a sufficient number of the documents (say, 10), they are sorted by relevance and presented to the user. Any further search is performed only if the user asks for more results. With this technique, in the most cases much smaller queries proved to be sufficient.

This technique is based on the presupposition that the documents containing the words nearer (in the sequence *letter case* > *morphology* > *thesaurus*) to the original user query keywords are always more relevant for the user. Actually, the user should be able to control the exact order of the partial queries. For example, in some cases morphological declension might be preferable to the lowercase reduction, e.g., *State* can be considered nearer to *States* than to *state* (cf. also section 7.2).

7. Experimental results

We investigated a 200 MB subset of the database of the Mexican Senate containing a representative mixture of the discourses of the Senators, laws, and other working documents of the Senate. The corpus contained 21,378,740 letter strings (running words), of them, only 174,386 strings different ones (0.8%). Then we reduced the strings in various ways. Obviously, the ratio of such reduction is equal to the ratio of inflating the query when lazy query expansion is used.

First, lowercase reduction gave 102,715 different strings, which shows that with only letter case equivalences taken into account, the query is inflated insignificantly—less than twice. The results for morphological and thesaurus-based expansion are discussed in the next subsections.

7. 1 Morphological query expansion

For our experiments we used a very simple morphological procedure based on a list of all possible chains of postfixes (suffixes, endings, and clitics) potentially used in Spanish, total of 3,578: *-a*, *-aba*, *-abais*, *-abamos*, *-aban*, *-andoselas* '-ing them$_{\text{feminine}}$ to him', ..., *-eandoselo*, *-eandoselos*, *-eandoseme*, *-eandosenos*, ..., *-ismo*, *-ismos*, *-ista*, *-istas*, ..., etc. Reduction of a word consists simply in removing the postfix; in case of ambiguity, all possible variants of reduction are tried: *hablaba* 'was speaking' → *hablab-*, *habl-*. Note that our reduction involves meaningful suffices, e.g., *comunismo* 'communism', *comunista* 'communist', *comunes* 'common$_{\text{plural}}$' → *comun-*, which increases the expansion ratio; our intention was to increase the recall with a simple and robust method.

Clearly, the method we used produces many incorrect stems and sometimes erroneously considers different words as if they had common stem, e.g., *démosle* 'let us give him' and *día* 'day' are assigned a common stem *d-*, see below. At later stages of the development of our system, a dictionary-based morphological analyzer will be used and the meaningful suffixes will be treated in the thesaurus; the postfix-list-based method will only be applied to the words absent in the morphological dictionary. This will further improve the query inflation ratio and the precision of the search.

Morphological reduction with our simple method showed that the database used 55,489 different stems. Therefore, the average number of strings per stem—i.e., the average ratio of lazy query expansion using only morphology—was about 4. We distinguished lowercase and uppercase letters; for example, the stem *cultiv-* was represented by three strings: *Cultiva*, *cultiva*, and *cultivaron*. The largest number of strings (including typos) per stem was 279 (stem *d-*: *D*, *Dádme*, *Dé*, *Démos*, *Démosle*, *Démosles*, *Dénnos*, *Día*, *Días*, *Díza*, *DA*, *DADO*, ..., *duelo*, *duelos*), then 201 (stem *s-*: *S*, *Sán*, *Sé*, *Sí*, *SA*, *SADAS*, *SAL*, *SALA*, *SALAS*, *SALES*, *SAN*, ..., *suelo*, *suelos*), then 200 (*v-*), 190 (*c-*), 172 (*m-*), 171 (*p-*), 150 (*t-*), 140 (*r-*), 131 (*l-*), 125 (*est-*: *éstó*, *ésta*, *éstan*, *éstas*, *éste*, *éster*, *ésto*, *éstos*, *ESTA*, *ESTABLE*, *ESTADO*, *ESTADOS*, ..., *estira*, *esto*, *estos*), the latter being the first non-single-letter stem in the list. For 183 stems, i.e., only 0.3% of the total stems present in the database, the number of strings per stem was greater or equal to 50, the total number of strings corresponding to these stems being 12377, i.e., 7% of the total number of different strings in the database.

As one can see, the words causing high expansion ratio are short words, mostly the forms of auxiliary verbs, or words with very broad meaning, or words incorrectly identified by our morphological procedure as having the same stem. Thus, though the average ratio of morphological lazy query expansion calculated by the strings of the database is 4, excluding the words with broad meaning that are not used in real queries and improving the morphological procedure would further decrease this figure. Indeed, in the real user queries, the average ratio we observed (with our postfix-list-based morphology) was about 3, which is a very promising result.

To evaluate this result—3 times query inflation with case and morphology reduction—let us consider the inflation ratio for case and morphology reduction with full query expansion, i.e., without the information on what strings are actually present in the database. In Spanish, a lexeme has on average about 300 wordforms (nouns 2, adjectives 4, verbs 700); so, the query is to be expanded 300 times. Then, each of these forms, in theory, can have about 1,000 letter case variants (10 letters, each one in lowercase or uppercase). This gives, for each query word, 700,000 variants word to be tried. Even if only three case variants are considered (*car*, *Car*, *CAR*), the number is 2,100—while lazy expansion gives only 3.

7. 2 Thesaurus-based query expansion

Here are two examples of thesaurus-based query expansion of type 1; see Fig. 1. In our dictionary, the concept *a Mexican city* consists of 2,413 names. When the name of city consisted of several words, e.g., *La Paz*, we considered both strings independently, as if the list included both words *la* 'the' and *paz* 'peace,' which resulted in 2,129 strings (due to repetitions of parts of the names). The database happened to mention 1,130 such words with case ignoring comparison, or 1,780 strings if uppercase and lowercase letters are distinguished.

Actually, only 1,077 of these were the names of cities, the rest being the words accidentally matching the name of a city (or a part of such a name) because of prior lowercase reduction, e.g., the word *paz* 'peace' matching the city name *La Paz*. This example shows once more the importance of the user's control over the types of reduction applied to the query: in this case, the thesaurus-based reduction should be applied without the lowercase reduction. With the index expansion technique, the decision on the order of application of reduction types is made at the moment of indexing (though it can be made in an intelligent manner for each word individually) and cannot then be changed by the user.

The concept *body parts* in our dictionary is represented by 97 words: *barba* 'beard', *barbilla* 'chin', ..., *mejilla* 'cheek' ..., *torso*, *tripa* 'intestine.' The database happened to mention 55 of them, or 86 strings with letter case distinguished. Most of these words were mentioned in the discourses of the Senators of rather sonorous style, e.g., "And will we now turn the other cheek?", "We will not drop on our knees!"

To evaluate this result, let us notice that with full expansion, all the words below some node are to be tried. Say, in the case of the concept *a Mexican city*, 2,413 variants would be tried, while lazy expansion gives 1,130.

Therefore, with pure thesaurus-based expansion, lazy expansion does not provide substantial improvement as compared with full expansion. The query inflation ratio is high and might be not affordable in practice.[3] However, as we have mentioned, due to the very nature of a thesaurus reflecting "general" knowledge that often proves to be not suitable for a particular user, as well as due to rather low quality of existing dictionaries, this type of expansion especially needs in the degree of user's control that cannot be provided by index expansion. Therefore, we consider the disadvantage of the query expansion method to be technical, i.e., temporal, while its advantage—a greater degree of control—to be fundamental. Note that as the dictionary is being elaborated, the inflation ratio will not significantly increase since the new words being added to the dictionary are of low frequency in the texts. In the next section, a possible workaround for the problem of too high query inflation will be discussed.

[3] In our system, the application of query expansion is based on the Informix DataBlade's synonym list feature. Our tests showed that this feature works with the entry size up to 3,000 synonyms for one headword. Thus, with this particular platform, thousand-fold expansion is still possible.

8. Future work: A combined technique

Even with the proposed technique, query expansion still slows down the system by inflating the user query, especially in case of thesaurus-based expansion. On the other hand, index expansion has an advantage of using the context of the word:

- Multiword expressions and idioms in the thesaurus, such as *hot dog*, can be handled naturally at the stage of indexing of the full text of the document.
- The words can be disambiguated in context; e.g., with the query "*wells*," the text *oil well* will be found while *he did it well* not.
- The structure of the document can be taken into account, e.g., words in the title or abstract can be indexed differently from the words in the main text.
- The global properties of the document not related with any specific word in it, such as the main topic of the document (Gelbukh et al., 1999; Guzmán-Arenas, 1998), can be used for indexing.

To provide these features without sacrificing the flexibility of the query language, the index expansion can be used in combination with the query expansion. The first step to such combination is the following. Both methods are implemented in the system; in particular, the documents are indexed with index expansion as explained in section 3. The user queries of standard types such as full morphological reduction or a full type 1 thesaurus-based query (see section 2.3), are processed fast with the expanded index without any query expansion. In the rather rare case when the user somehow modifies the list of strings to be matched, index expansion is used.

The division of work between the two methods can be optimized. For example, only deep levels of the thesaurus can be considered for index expansion (*matrix, equation, inequality*, ... $\rightarrow$ UP-*mathematics*, see section 3), while the upper level hierarchy, if need by the query, can be taken into account by query expansion (*science* → UP-*mathematics* OR UP-*physics* OR UP-*chemistry*). What is more, the way the users most frequently modify their queries can be automatically, semi-automatically, or manually learned and implemented in the index expansion. For example, if the users frequently exclude the form *como* from the paradigm of the Spanish verb *comer* (see section 2.2), then it should be excluded at the stage of index expansion; the query with the unmodified paradigm will be internally (and transparently for the user) implemented, if needed, through query expansion as "MORPH-*comer* OR *como*."

In addition, the index expansion can be further improved when used in combination with query expansion. In the section 3 we introduced the marks for the keywords added to the index during expansion, such as MORPH-, UP-, etc. To allow even more flexibility necessary for the type 2 or 3 thesaurus-based queries (see Fig. 1), the distance (in terms of levels) from the source word to the generalized concept is to be marked in the index. With this, the example from the section 3 can be rewritten as follows: *Mainframes* → ..., UP-1-*computer*, UP-2-*device*, UP-3-*artifact*. Now the type 2 query "*devices*, but not more than 2 levels down" is implemented as query expansion "UP-1-*device* OR UP-2-*device*" and thus will fetch *Mainframes*, while the query "*artifacts*, but not more than 2 levels down" internally implemented as "UP-1- *artifact* OR UP-2-*artifact*" will not.

The queries of the other three types are implemented similarly by enumerating the relevant nodes. Even if the user excludes some words or nodes from the sub-hierarchy, which results in a non-standard query, the nodes kept intact can be enumerated in the UP-... notation instead of enumerating all keywords as was suggested in sections 0 and 5. The original keywords like UP-*computer* can be kept in the index and used only for the most frequent—type 1—queries.

The combined technique compensates for the query inflation problem caused by query expansion, especially of the thesaurus-based type. It has the advantage of higher performance due to smaller queries, without sacrificing flexibility. Obviously, it implies both advantages and disadvantages of index expansion. Specifically, it gives the advantage of the possibility to consider the context. On the other hand, it introduces the methodological and technical disadvantages of index expansion listed in the section 3, such as maintenance problems and undesirable coupling of the DBMS and the lingware. A closer investigation of the combined technique will be the direction of our future work.

9. Conclusions

We have shown that the traditional technique for non-literal string matching in information retrieval—index expansion—has some inherent disadvantages, and have suggested a new technique—lazy query expansion—that allows greater flexibility of queries, better overall system architecture, and easier maintenance.

Our method still has two problems: (1) the expanded queries in some cases are significantly larger and thus work slower, and (2) it is not obvious how to take the context of the keyword into account. As a solution, a combination of the query and index expansion methods was discussed.

References

Aho, Alfred V. *Algorithms for finding patterns in strings.* In J. van Leeuwen (ed.), Handbook of Theoretical Computer Science, chapter 5, pp. 254-300. Elsevier Science Publishers B. V., 1990.

Cassidy P. *An Investigation of the Semantic Relations in the Roget's Thesaurus: Preliminary Results.* In: A. Gelbukh (ed.), Computational Linguistics and Intelligent Text Processing, IPN-UNAM, Mexico, to appear. See also Proc. of CICLing-2000, February 2000, CIC-IPN, Mexico City, ISBN 970-18-4206-5.

Gelbukh, A. *A data structure for prefix search under access locality requirements and its application to spelling correction.* Proc. of MICAI-2000: Mexican International Conference on Artificial Intelligence, Acapulco, Mexico, 2000.

Gelbukh, A., **G. Sidorov**, and **A. Guzmán-Arenas**. *Use of a Weighted Topic Hierarchy for Document Classification*, Matoušek et al., TSD-99: Text, Speech, Dialogue. Lecture Notes in Artificial Intelligence N 1692, Springer, 1999.

Gusfield, Dan. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, 1997; ISBN: 0521585198.

Guzmán-Arenas, Adolfo. *Finding the main themes in a Spanish document*, Journal Expert Systems with Applications, Vol. 14, No. 1/2. Jan/Feb 1998, pp. 139-148.

Fellbaum, Ch. (ed.) *WordNet as Electronic Lexical Database.* MIT Press, 1998.

Frakes, W., and **R. Baeza-Yates**, editors. *Information Retrieval: Data Structures and Algorithms.* Prentice-Hall, 1992.

Hausser, Ronald. *Three principled methods of automatic word form recognition.* Proc. of VEXTAL: Venecia per il Tratamento Automatico delle Lingue. Venice, Italy, Sept. 1999.

Koskenniemi, Kimmo. *Two-level Morphology: A General Computational Model for Word-Form Recognition and Production.* University of Helsinki Publications, N 11, 1983.

Kowalski, Gerald. *Information Retrieval Systems Theory and Implementation*, Kluwer Academic Publishers, 1997.

Lenat, D. B. and **R. V. Guha**. *Building Large Knowledge Based Systems.* Reading, Massachusetts: Addison Wesley, 1990. See also more recent publications on CYC project, http://www.cyc.com.

Alexander Gelbukh was born in Moscow in 1962. He obtained his Master degree in Mathematics in 1990 from the department of Mechanics and Mathematics of the Moscow State "Lomonossov" University, Russia, and his Ph.D. degree in Computer Science in 1995 from the All-Russian Institute of the Scientific and Technical Information (VINITI), Russia. Since 1997, he is the head of the Natural Language and Text Processing Laboratory of the Computing Research Center, National Polytechnic Institute, Mexico City. He is a member of SNI, Mexico, since 1998. He is the author of about 100 publications on computational linguistics. See http://www.cic.ipn.mx/~gelbukh.