Full-Text Indexing Based on Lexical Relations

An Application: Software Libraries

•

CUCS-485-89

Frank Smadja
Department of Computer Science
Columbia University
New York, NY 10027
Smadja@cs.columbia.edu

Yoelle S. Maarek
IBM Thomas J. Watson Center
P.O Box 704
Yortown Heights, NY 10598
yoelle@ibm.com

August 1989
Proceedings of the 12th ACM SIGIR, Cambridge, MA, June 1989.

Frank Smadja was partly supported by DARPA under contract #N00039-84-C-0165 and NSF grant IRT-84-51438.

CUCS-485-89

Full Text Indexing Based on Lexical Relations An Application: Software Libraries *

Yoëlle S. Maarek
IBM Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Frank A. Smadja
Department of Computer Science
Columbia University
New York, NY 10027

Abstract

In contrast to other kinds of libraries, software libraries need to be conceptually organized. When looking for a component, the main concern of users is the functionality of the desired component; implementation details are secondary. Software reuse would be enhanced with conceptually organized large libraries of software components. In this paper, we present GURU, a tool that allows automatical building of such large software libraries from documented software components. We focus here on GURU's indexing component which extracts conceptual attributes from natural language documentation. This indexing method is based on words' co-occurrences. It first uses EXTRACT, a co-occurrence knowledge compiler for extracting potential attributes from textual documents. Conceptually relevant collocations are then selected according to their resolving power, which scales down the noise due to context words. This fully automated indexing tool thus goes further than keyword-based tools in the understanding of a document without the brittleness of knowledge-based tools. The indexing component of GURU is fully implemented, and some results are given in the paper.

Keywords: automatic indexing, software libraries, software reuse, lexical relations, natural language processing, co-occurrence knowledge.

1 Introduction

Software reuse has been shown to improve software productivity and reliability, and therefore becomes an issue of ever growing interest in software engineering [Horowitz 84]. Unfortunately, not enough adequate libraries of reusable software components are available. Software libraries, in contrast to restricted domain libraries such as statistical or mathematical libraries, need to be conceptually organized so as to facilitate locating as well as understanding components even in the case of imprecise queries [Maarek 87].

We have designed and implemented a tool, GURU, that automatically assembles software libraries into conceptual hierarchies. GURU extracts conceptual information from the natural language documentation associated with the software components to be stored. GURU is articulated around two main components, the indexing component and the classifying component.

- GURU's indexing component extracts conceptual attributes from natural language documentation. The attributes characterize the document and stand for a formal functional description of the analyzed software units.
- GURU's classifying component then incrementally assembles the indexed software units into a conceptually structured library by using a conceptual clustering technique.

GURU has been fully implemented and is currently tested on various domains. GURU has been able to fully automatically index and classify more than 250 software components taken from the UNIX[1] environment. In this paper, GURU's indexing component has been enriched by the use of EXTRACT. A complete description of GURU can be found in [Maarek 89].

GURU's indexing component makes use of a new indexing scheme that is based upon the concept of lexical relation due to Saussure [Saussure 49]. As the front end of GURU's indexing component, we have used EXTRACT [Smadja 89], a co-occurrence compiler that produces lexical relations from a given textual corpus. EXTRACT was introduced for automated lexicography and natural language generation purposes and produces the necessary raw information for GURU to analyze and produce

*In Proceedings of SIGIR'89, Twelfth International Conference on Research and Development in Information Retrieval, ed. N.J Belkin and C.J. van Rijsbergen, ACM Press, pp 198-206, Cambridge, MA, June 1989.

[1] UNIX is a trademark of AT&T Bell Laboratories

indices. GURU's indexing scheme allows extracting conceptual information from natural-language documentation. In contrast to classical indexing schemes, GURU's scheme does not require any *a priori* information on the context of the document to be analyzed, and provides a conceptual representation of the document without attempting to actually understand it.

Section 2 presents some definitions and related work in automated indexing and Section 3 explains how EXTRACT produces lexical relations. Section 4 describes how to build indices from lexical relations. Finally, Section 5 gives an example output of GURU's indexing component along with some results.

2 Related Work: Keywords or Knowledge?

Let us first define our terminology. Let D be the universe of textual documents and R be the set of possible representations (finite or infinite). A function $\sigma : D \longrightarrow R$ that maps a document into its representation is an *indexing function*, and for $d \in D$, $\sigma(d)$ is called the *description* or *descriptor* of d. The elementary constituents of a descriptor are called the *indices*, or *attributes* of the document. We have distinguished two approaches in automatic indexing: the *keyword-based approach* and the *knowledge-based approach*.

In most keyword-based indexing systems, a list of predefined keywords is provided and documents are analyzed in order to check whether or not they contain these keywords. Some systems refine the description by including adjacency or frequency information. In SIRE [Salton 83], for instance, frequency information is added to the inverted file, and is used to rank the results of inverted file retrieval. Other systems, such as SMART [Salton 71,Salton 83], use a similarity measure between documents based upon keyword-frequency. Keyword-based systems, even when augmented with frequency information, present a major drawback when dealing with large textual databases: they lack *granularity*. Indeed, the presence or absence of keywords is not a sufficient criterion for distinguishing between documents when they become too numerous. In large-scale applications equivalence classes defined by σ become too large to be of interest. Consequently, too many documents are often retrieved, which lowers the precision of the retrieval system without raising its recall.

As keyword-based systems perform poorly on large databases, supporters of the knowledge-based approach claim that efficient retrieval from large databases requires *understanding* the text as well as indexing it [Flass 85]. As total understanding of a text is impracticable, the major trend in the knowledge-based approach is the text-skimming approach. The latter consists of skimming texts to determine their main themes without necessarily paying attention to *each* word [Mauldin 86,Mauldin 87]. Mauldin's system, Ferret, allows understanding a text by using *sketchy scripts*. Sketchy scripts are case-frame structures that encode knowledge about the main themes of the texts, *i.e.*, common actions and events. For indexing a document, a particular script has to be chosen in the script library and instantiated according to the results obtained when skimming this document.

The knowledge-based approach allows a deeper understanding of the documents than the classical keyword-based approach. However, it is more context sensitive. Indeed, scripts cannot be adequately instantiated if they do not fit predefined relevant slots depending on the domain of the analyzed text. Scripts, though helping in the understanding, are a source of brittleness. Such an approach is applicable in restricted domains where the knowledge encoded in the scripts can easily be circumscribed. However, it is prohibitive in the context of software libraries in which the domain is too wide to be encoded beforehand.

In comparison to the knowledge-based approach, the keyword-based approach is less context dependent, however, the main flaw, is that in concentrating on isolated words, a lot of information is lost. In particular, information on the relationships in which words are involved. This information may be necessary for distinguishing between texts having similar keywords. In this paper, we propose extracting more information from textual documents than single keywords, without introducing *a priori* information such as in knowledge-based approach, by using a new indexing scheme. This scheme is presented in the next sections.

3 From Texts to Lexical Relations

3.1 Lexical Relations, Previous Work

It is widely accepted that indexing a text at a high level of granularity requires understanding it to some extent. The ideal solution would be to use a full-fledged natural-language understander to actually understand the documents to be indexed. However, such a system is still far from being at hand, the main problem being the semantic analysis. Until the combined efforts of linguistics and artificial intelligence provide us with more powerful methods and tools, we propose to limit ourselves to non-semantic knowledge. More than single keyword distributions must be retrieved from the natural language texts without actually understanding them, *i.e.*, without using semantic information.

As a solution to this problem we have used here lexical relations as basic indexing attributes. A lexical relation between two units of language stands for a correlation of their common appearance in the utterances of the language [Saussure 49]. The observation of lexical relations in a text have been shown to reveal a lot on both syntactic and semantic levels, and provides us with a powerful way of taking context into account [Smadja 89]. In our context, considering lexical relations allows keeping track of contextual information and thus go beyond the keyword barrier.

Lexical relations have been of interest for a long time in various fields of study such as linguistics, lexicography and information retrieval. Linguists such as Saussure, [Saussure 49], Halliday [Halliday 66] and later Mel'čuk [Mel'čuk 73] have investigated on the issue and have come up with a more and more precise definition of lexical relations. Their efforts have resulted in several models reflecting their various interests. Halliday followed Saussure's early incentive and investigated the interactions of lexical relations with syntax and semantics; he forcefully indicated the pervasiveness of the phenomenon. Mel'čuk went further and integrated lexical relations into his full fledged linguistic model.

At the crossroad of lexicography and psycholinguistics, Rodale [Rodale 47] compiled a dictionary restricted to an extensive listing of multiple word combinations. More recently, Benson *et. al.* proposed a semantics based model of lexical relations and effectively used it in their combinatory dictionary, the BBI [Benson 86]. The BBI is currently the most complete account of co-occurrence knowledge and accurately lists several thousands of collocations.

In information retrieval, collocations have been widely used. Karen Sparck Jones [Sparck Jones 86] in her recently published 1964 PhD thesis extensively investigates the importance of lexical relations in the context of semantic classification. She defines semantic primitives in terms of the textual behaviors of words. In her work, Sparck Jones is however more concerned with paradigmatic lexical affinities. In contrary, in this paper, we are only interested with syntagmatic lexical relations. Out of concerns of information retrieval and automated lexicography, Choueka [Choueka 88] proposes several implemented algorithms for retrieving collocations from large corpora. Choueka is more interested in the retrieval of newly-coined expressions such as *President Reagan*, *home run*, *United Nations* than true lexical relations. His work can be seen as an essential first step to automated lexicography and develops a useful methodology for the handling of large corpora. Although we are not interested in the same kind of collocations, our extracting method is clearly related to Choueka's methodology.

3.2 The Extracting Tool

3.2.1 EXTRACT, Presentation

The extracting tool we are using in this paper, EXTRACT [Smadja 89] (See Section 3.3.), was originally developed out of concerns for language generation and lexicography. EXTRACT deals with what Halliday terms "lexicogrammaticalness" and its original goal was to retrieve lexical relations involving modifier-modified syntactic units. For instance, EXTRACT retrieves lexical relations such as: ***commit-suicide***, ***make decision***, ***answer-question***.

In this paper, we will refer back to the original definition introduced by Saussure and consider that two words are bound by a lexical relation if their co-occurrences in a given syntactic unit are correlated. We actually restrict ourselves to ***open-class words***[2] as meaning bearing, whereas lexical relations involving closed-class words are not. Lexical relations relating open-class words can be classified according to the syntactic relation between them. For instance, lexical relations of type ***subject-verb***, ***verb-direct object***, ***verb-indirect object***, etc. In order to identify lexical relations, pairs of words joined by such syntactic links must first be retrieved. Consider the following sentence,

```
''Normally each line found is copied to the
standard output.''
```

Some of the potential lexical relations in this sentence are:

- lexical relation of type verb-direct-object, *e.g.* `(find line)`, `(copy line)`.
- lexical relation of type verb-indirect-object, *e.g.* `(copy output)`.
- lexical relation of type noun-adjective, *e.g.* `(standard output)`.

Among these lexical relations, some correspond to abstractions of the considered document, and some do not. Since we are interested here in the indexing of textual

[2] Closed class words refer to small syntactic categories, such as articles, prepositions etc. In contrast, open class words are nouns, adjectives and adverbs and are therefore much more numerous. Closed class words are somehow reachable by grammar rules whereas open class words are dealt with in the lexicon [Huddleston 84].

documents, we are going to differentiate the statistical distribution of open-class words across several documents. It has been demonstrated that the frequency of occurrence of a term within a document is related to the importance of the word in a text [Luhn 58]. This is also true for the common appearance of pairs of words and *a fortiori* for lexical relations. The next section explains how lexical relations are identified in the corpus. Section 4 presents how indices are produced from them, and how only conceptually relevant lexical relations are kept.

Ideally, lexical relations are extracted from a text by parsing it. Two words are involved in a lexical relation, if they belong to the same syntactic constituent, *e.g.*, noun phrase, verb phrase, sentence, etc. However, in real life, *free-style* texts contain a lot of non-standards features over which automatic parsers would stumble. Moreover, since we are dealing with numerous components containing numerous modules, large and numerous samples of texts have to be scanned which would strain resources of the machine on which the parser would run. As an alternative, simply taking note of the neighborhood of appearance of open-class words is more than satisfactory in our context.

3.2.2 The Extracting Technique

It has been shown that 98% of lexical relations relate words separated by at most five words within a single sentence, [Martin 83]. In other words, most of the lexical relations involving a word w can be retrieved by examining the neighborhood of w, wherever it occurs, within a span of five words (-5 words and +5 words around w).

To retrieve the lexical relations from a document, we use here the front-end of the EXTRACT co-occurrence compiler described in [Smadja 89]. EXTRACT allows identifying co-occurrences within a document. It applies the scanning technique described above. EXTRACT's front-end takes as input a document d, a span parameter (in our case, five) and a dictionary specifying closed-class words, and produces a list of tuples (w_1, w_2, f), where (w_1, w_2) is a lexical relation between two open-class words identified in d, and f is the number of its occurrences in d. The retrieval process consists of the following three steps for each lexical entry, w:

1. **Scan:** Scan the whole text for each appearance of w.

2. **Compile:** For each sentence containing w, make a note of its collocates[3]. All collocates are stored along with their syntactic category and their frequency of appearance.

3. **Lemmatize** A basic morphological analysis of every word involved in a lexical relation is performed. The morphological analysis is built on top of the UNIX `spell` program. Each word is mapped into its morphological root[4] using simple inflectional transformations. This inflectional information as well as the primitive procedures used for the mapping are derived from the UNIX `spell` program. The lemmatization is not perfect, ambiguous words remain ambiguous, but this does not greatly weaken our indexing scheme since the same lemmatization procedure is applied for queries.

EXTRACT is fully implemented and has been tested on a 300,000 word corpus taken from the UNIX news net. In spite of the small size of the corpus, we have been able to make useful lexicographic observations. EXTRACT is currently being tested on a more than 2,500,000-words corpus taken from the archives of `The Jerusalem Post`. The corpus consists of several thousand articles that have been recently published in the newspaper. Those two corpora have allowed us to identify several hundred lexical relations, among them many corresponding to frequently used co-occurrence relations that are unpredictable in terms of syntax or semantics. EXTRACT is currently being used in the framework of language generation work using specialized corpora. Experiments using EXTRACT can be found in [Smadja 89].

In the context of GURU, EXTRACT is used as the lexical relation identifier. EXTRACT produces a set of lexical relations from natural language documentation. Once extracted, these lexical relations must be filtered out by GURU in order to identify the conceptually relevant ones.

3.3 Noise in Single Words vs Noise in Lexical Relations

The frequency of occurrence of a word within a document reflects the importance of the concept(s) it stands for in the text. It should be noted however that the importance of a word does not increases linearly with its frequency of appearance. As explained by Luhn, neither *low-frequency* nor *high-frequency* terms are meaning bearing [Luhn 58]. High-frequency terms within a document are also referred to as the *noise* in the document. Only those terms appearing in a middle range have a high *resolving power*, where the resolving power of a word is the ability of a word to characterize a document. For instance, the word *"the"* which is the most

[3]By collocate, we mean the nearby open-class lexical item.

[4]Let us note that we do not extract the absolute morphological stem of any word but rather the inflectional stem.

frequent in almost all the documents analyzed has, in most contexts, a very low resolving power.

Let us show how using lexical relations as atomic unit instead of single words reduces the noise problem and thus increases the recall of the retrieval system. Noise mainly originates from the following two sources:

1. Closed-class words that are used very often and do not actually bear meaning. Such words are likely to be retrieved in a lot of document descriptions and thus weaken indexing power.
2. Words that are very often used in a particular context also bear no meaning. When dealing with specialized documents, a lot of words relevant to the general context are likely to come across without bearing any supplementary meaning. For instance, if processing a car engine manual, the word "engine" is likely to appear often. In the UNIX manual, the word *"file"* is among the most frequent and does not provide much additional information on specific functionality.

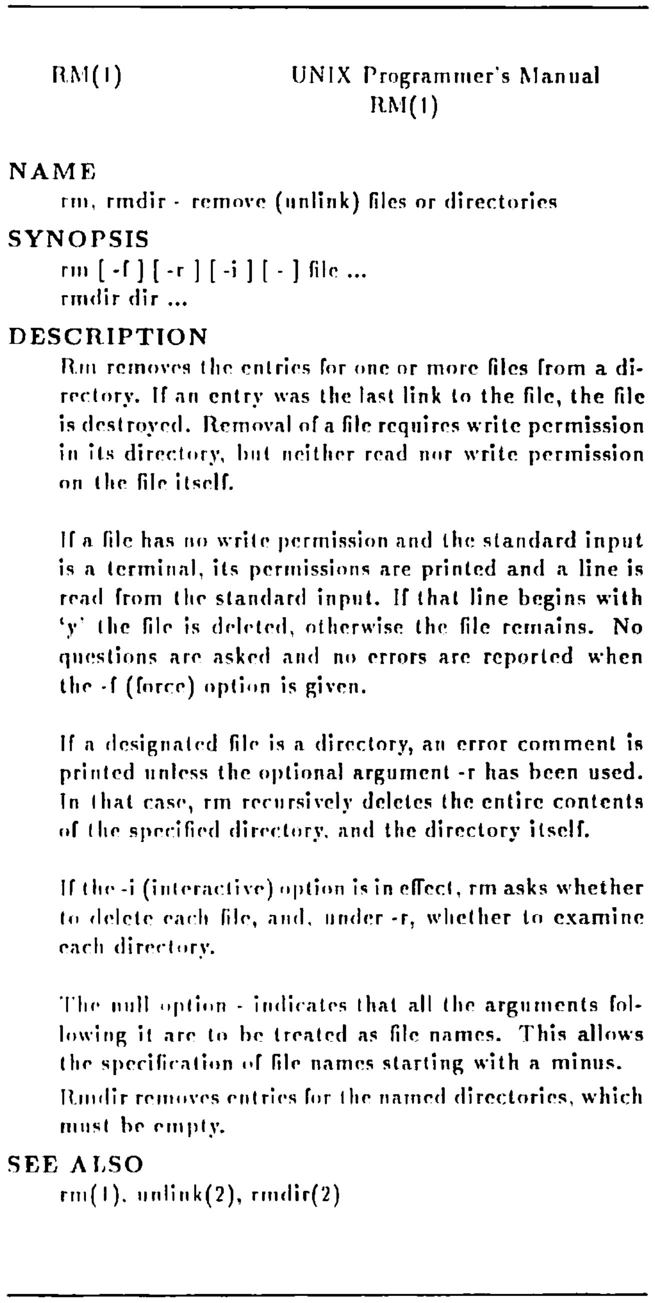
By taking only open-class words into account, closed-class words, which are the major source of noise in frequency observations, are automatically removed. In addition, the other source of noise is greatly reduced, compared to keyword-based approaches. For example, when analyzing a document from the UNIX manual, instead of getting too many occurrences of the word "file", we get lexical relations such as `(write file)`, `(delete file)`, that are meaning bearing, with much smaller frequencies of appearance. The next section shows these results on the `rm` manual page of the UNIX programmer's manual.

3.4 An Example Output, the `rm` Manual Page

Table 1 presents in its second column partial outputs of EXTRACT applied to the UNIX manual page of `rm` (See Figure 1). The lexical relations are ranked by frequency order and are compared to the most frequent single words of this manual page. In order to make a fair comparison, we have removed closed-class words from the single words list[5]. This comparison reveals that the most frequent lexical relations bring much more knowledge on the functionality of the function `rm` than single keywords.

As we see, in the lexical relations list, crucial concepts such as `(delete file)` or `(file removal)` are among

[5] The closed-class word *"the"* appears the most often in `rm` with a number of occurrences equal to 19, the second most frequent being *file* with 13.



```
RM(1)                UNIX Programmer's Manual
                              RM(1)

NAME
     rm, rmdir - remove (unlink) files or directories
SYNOPSIS
     rm [ -f ] [ -r ] [ -i ] [ - ] file ...
     rmdir dir ...
DESCRIPTION
     Rm removes the entries for one or more files from a di-
     rectory. If an entry was the last link to the file, the file
     is destroyed. Removal of a file requires write permission
     in its directory, but neither read nor write permission
     on the file itself.

     If a file has no write permission and the standard input
     is a terminal, its permissions are printed and a line is
     read from the standard input. If that line begins with
     'y' the file is deleted, otherwise the file remains. No
     questions are asked and no errors are reported when
     the -f (force) option is given.

     If a designated file is a directory, an error comment is
     printed unless the optional argument -r has been used.
     In that case, rm recursively deletes the entire contents
     of the specified directory, and the directory itself.

     If the -i (interactive) option is in effect, rm asks whether
     to delete each file, and, under -r, whether to examine
     each directory.

     The null option - indicates that all the arguments fol-
     lowing it are to be treated as file names. This allows
     the specification of file names starting with a minus.
     Rmdir removes entries for the named directories, which
     must be empty.
SEE ALSO
     rm(1), unlink(2), rmdir(2)
```

Figure 1: The `rm` manual page from the on-line UNIX documentation

Keywords list	Lexical Relations
13 file	**3 delete file**
8 is	3 file file
6 rm	3 file permission
5 directory	3 file write
4 rmdir	2 file require
4 are	**2 file removal**
3 write	2 file name
3 permission	2 file link
3 option	**2 entry remove**
2 removes	2 directory write
2 standard	2 directory specify
2 unlink	**2 destroy file**
...	...

Table 1: Keywords and lexical relations classified by frequency in the `rm` manual page

the most frequent, whereas the keyword `remove` is in a relatively remote position in the keyword frequency list. This has the double effect of reducing the noise problem and raising the recall of the GURU retrieval system. However, the noise problem is not totally eliminated by using lexical relations. Some lexical relations, such as `(file file)`, are not very significant in spite of their high frequency. In the next section, we see how it can yet be scaled down by using a measure for evaluating the resolving power for lexical relations.

4 From Lexical Relations to Indices

When analyzing a document, a lot of lexical relations are identified. These lexical relations are more or less significant. As seen in Table 1, frequency of appearance is a good indicator of relevance. However, some noise still exists which is mainly due to words appearing too often in a certain context. In order to reduce the influence of such words, we propose to select among the lexical relations identified, only the most representative ones, *i.e.* those containing the more information.

The *quantity of information of a word w* in a given textual universe is defined in [Salton 83] as:

$$\text{INFO}(w) = -\log_2 P\{w\}$$

where $P\{w\}$ is the measured probability of occurrence of the word w in the considered universe.

Based upon this definition of the quantity of information for single words, we define the *quantity of information of a pair of words* (w_1, w_2) in a given textual universe as follows,

$$\text{INFO}((w_1, w_2)) = -\log_2(P\{w_1, w_2\})$$

where $P\{w_1, w_2\}$ is the probability of occurrence of the pair of words (w_1, w_2) in the textual universe.

To simplify the computation of this factor, in the rest of this work, we consider words within the textual universe as independent variables. This assumption represents only an approximation since words in English are definitely not independent, but are rather distributed according to the rules of the language. However it is a valid starting approximation that is currently used in corpus based linguistics [Garside 87], [Martin 83]. This precisely allows to single out exceptions, that is correlated words or dependent variables. Dependent variables would account for purely *lexical* affinities[6] between words. We are not interested in identifying such lexical relations here. Our primary concern is rather to identify *conceptual* affinities within one specific part of the corpus. Given this assumption, we can express the quantity of information of a lexical relation,

$$\text{INFO}((w_1, w_2)) = -\log_2(P\{w_1\} \times P\{w_2\})$$

where $P\{w_1\}$ and $P\{w_2\}$ are the probabilities of occurrence of the words w_1 and w_2 in the textual universe.

Then, we define ρ the *resolving power* of a lexical relation within a document d as follows:

Let (w_1, w_2, f) be a tuple retrieved while analyzing a document d, where (w_1, w_2) is a lexical relation appearing f times in d. The *resolving power*[7] of this lexical relation in d is defined as:

$$\rho((w_1, w_2, f)) = f \times \text{INFO}((w_1, w_2))$$

The higher the resolving power of a lexical relation is, the more characteristic of the document it is. The resolving power, as defined above, allows evaluating the importance of a lexical relation within a text, by taking into account both its frequency of appearance in the text and the quantity of information of the words involved. Thus, even if it appears 3 times in a manual page, the lexical relation `(file file)` will have a

[6] EXTRACT's original goal is to retrieve this sort of lexical co-occurrences, i.e., lexical relations that cannot be accounted for on purely semantic grounds. For example, one says, *"to commit a murder"* as well as *"to perpetrate a murder"*, but one only says, *"to commit suicide"*. *Commit-suicide* is what we call a lexical co-occurrence relation.

[7] This notion is related to that of mutual information [Ash 65].

lexical relation	ρ
permission write	66.712204
rmdir unlink	58.742599
file permission	53.760002
delete file	50.979298
file write	49.116001
file removal	44.747799
directory file	43.998001
entry remove	43.306801
destroy file	40.747799
input permission	40.297401
directory remove	38.820801

Table 2: Most significant lexical relations of the rm manual page

small resolving power, simply because the quantity of information of the word `file` is low.

By selecting the lexical relations with the highest resolving power, we obtain a characteristic description of the document analyzed. The number of lexical relations to be kept depends on the size of the library to be built. The larger the library, the more numerous the lexical relations selected in order to avoid two distinct components having the same σ-representation. In our test case, the UNIX library, we have empirically verified that keeping the 10 most significant lexical relations of each document is largely sufficient.

We define the σ-representation of a document as follows. Let D be the universe of natural language documents, R be the set of possible representations (finite or infinite), and n a user-given parameter depending on the size of the library. To each document d in D, is associated a set LR_d of tuples (w, w', ρ) formed by the extracted lexical relations and their corresponding resolving powers. Our indexing scheme is defined by the function $\sigma : D \rightarrow R$ where $\sigma(d)$ is the subset of LR_d consisting of the n lexical relations having the highest ρ values.

5 Evaluation

As an example, we consider the UNIX manual page for the **rm** function, given in Figure 1. Table 2 presents its eleven most significant lexical relations.

Let us compare the results given in Table 2 with the permuted index of **rm** manual page. The permuted index is the index system currently used in UNIX in order to locate a particular function or command. The permuted index file is generated by the **ptx** function provided by UNIX. The permuted index system is a primitive keyword-based indexing system. We consider it here as a comparison mainly because it is widely used (together with **grep**) by the UNIX users, and because it does not make use of context-dependent information. It can be seen as a ***semi-manual*** indexing tool since the manual page much be formatted such that the **NAME** section represents the key concepts of the document. The user can thus locate an object as soon as s/he knows (or guesses), one of the keywords appearing in the **NAME** section. This method presents many drawbacks as soon as the size of the library increases, because of its low level of granularity. Too many components may be classified under the same keyword. However, we do not deal here with the problems that may occur at retrieving stage but rather with the nature of the description. We argue that by using a lexical-relation based approach a much richer σ-representation is produced. In particular, key concepts and synonyms of these concepts can be identified in absence of any thesaurus.

Thus, if we compare the index represented by the most significant lexical relations of the **rm** manual page (See Table 2) and the permuted index of **rm**, (See **NAME** section in Figure 1) we notice that key concepts such as `(delete file)` or `(destroy file)` are present among the lexical relations and totally absent from the permuted index, even in keyword form.

Using an lexical relation-based indexing method allows including more knowledge in descriptions than simple keywords. As a consequence, our indexing scheme is more exhaustive than any keyword-based language. As a tradeoff, as soon as a language gains in exhaustivity, there are risks that it looses in specificity. In other words, achieving a high recall may lower the precision of the IR system. This drawback is avoided in GURU by selecting lexical relations with high resolving power. Both high recall and high precision rates are thus obtained.

An experiment has been conducted that allowed us to evaluate the retrieval effectiveness of GURU as compared to that of **man -k**, with the UNIX manual as our testbed.[8] We have analyzed the first section of the UNIX manual which comprises more than 120 000 words. The indices produced have been directly fed to the GURU classifying tool, but could as well be used in the context of various other library systems. We have conducted a comparative test between GURU and **man -k** by measuring their respective recall and precision rates. The results of this test revealed that GURU was significantly better than **man -k** both in terms of recall and precision. The results of this test depend not only on the

[8] **man -k** can be seen as the complete IR system provided with the UNIX environment, **ptx** representing its associated indexing component.

quality of the indexing component but also of the classifying component and are therefore not to be presented here. The complete evaluation of GURU's effectiveness can be found in [Maarek 89].

6 Conclusion

GURU is a tool that automatically builds large software libraries from the natural language documentation generally associated with them. We presented here the indexing method of GURU. GURU's indexing method is centered around the concept of lexical relation and the notion of quantity of information. For producing indices out of natural language documentation, GURU first extracts lexical relations and then selects the conceptually relevant ones by measuring their resolving power.

The method proposed here is purely structural and thus can be performed automatically. Classifying retrieved lexical relations according to their resolving powers allows us to define an indexing function outperforming both simple and augmented keyword-based approaches without introducing the brittleness of the knowledge-based approach. GURU has been successfully tested on the context of the UNIX software library. More detailed results and evaluation of GURU can be found in [Maarek 89], and a complete description of EXTRACT can be found in [Smadja 89].

Acknowledgements

Dan Berry helped a lot on the conceptual development of GURU's indexing scheme. Yaacov Choucka gave knowledgeable advice to F.Smadja during his stay at Bar-Ilan University. John Justeson read earlier versions of this paper and made useful criticisms and suggestions. Y. Maarek performed most of this work while at Technion, Israel Institute of Technology, Israel. F. Smadja performed part of this work while visiting at Bar Ilan University. Department of English Linguistics, Israel, and part of this work while supported by DARPA grant N00039-84-C-0165, and NSF grant IRT-84-51438.

References

[Ash 65] R. B. Ash, *Information Theory.* Interscience Tracts in Pure and Applied Mathematics, No. 19, Interscience Publishers, New York. 1965.

[Benson 86] M. Benson, E. Benson, R. Ilson, *The BBI Combinatory Dictionary of English, A Guide to Word Combinations.* John Benjamin Publishing Company, Amsterdam/Philadelphia, 1986.

[Blair 85] D.C. Blair and M.E. Maron, *An Evaluation of Retrieval Effectiveness for a Full-Text Document-retrieval System.* Communications of the ACM 28:3, pp 289-299, March 1985.

[Choueka 88] Y. Choueka, *Looking for Needles in a Haystack.* In Proceedings of the RIAO, p:609-623, 1988.

[Flass 85] P.R. Flass, *Technical Correspondence.* Communications of the ACM, 28(11), pp 1238, November 1985.

[Garside 87] R. Garside, G. Leech and G. Sampson, (eds), *The Computational Analysis of English: A Corpus Based Approach.* Longman, London, 1987.

[Halliday 66] M.A.K. Halliday, *Lexis as a Linguistic Level.* In C.E. Bazell, J.C. Catford, M.A.K Halliday and R.H. Robins (eds.), *In memory of J.R. Firth*, Longmans Linguistics Library, pp 148-162, London, 1966.

[Horowitz 84] E. Horowitz and J. Munson, *An Expensive View of Software Reuse.* IEEE Transactions on Software Engineering, Vol SE-10, September 1984.

[Huddleston 84] R. Huddleston, *Introduction to the Grammar of English.* Cambridge Textbooks in Linguistics, Cambridge University Press, 1984.

[Luhn 58] M. Luhn, *The Automatic Creation of Literature Abstracts.* IBM Journal of Research and Development, Vol. 2, No. 2, pp 159-165, April 1958.

[Maarek 87] Y.S. Maarek and G.E. Kaiser, *On the Use of Conceptual Clustering for Classifying Reusable Ada Code.* ACM SigAda International Conference on the Ada Programming Language, pp 208-215, Boston, MA, December 1987.

[Maarek 88] Y.S. Maarek, *Using Cluster Analysis for Assisting Maintenance of Large Software Systems.* In Proceedings of the IEEE Israel Conference on Computer Systems and Software Engineering, pp 178-186, Tel Aviv, Israel, June 1988.

[Maarek 89] Y.S. Maarek, *Using Structural Information for Managing Very Large Software Systems.* D.Sc. Dissertation. Computer Science Department, Technion, Israel Institute of Technology, Israel, January 1989.

[Martin 83] W.J.R. Martin, B.P.F. Al and P.J.G van Sterkenburg, *On the processing of a text corpus: from textual data to lexicographical information.* Lexicography: Principles and Practice, Ed. R.R.K Hartmann, Applied Language Studies Series, Academic Press, London, 1983.

[Mauldin 86] M. Mauldin. *Information Retrieval by Text Skimming.* Thesis Proposal, Carnegie-Mellon University, Pittsburgh, May 1986.

[Mauldin 87] M. Mauldin, J. Carbonell and R. Thomason. *Knowledge-Based Information Retrieval.* In Proceedings of the 29th Annual Conference of the National Federation of Abstracting and Information Services, Elsevier Press, 1987.

[Mel'čuk 73] I.A. Mel'čuk, *Lexical Functions in Lexicographic Description.* In Proceedings of the Berkeley Linguistics Society, 8, 1973.

[Rodale 47] J.I. Rodale, and Staff, *The Word Finder.* Rodale Books, Inc. Emmaus, Pennsylvania, 1947.

[Salton 71] G. Salton, *The SMART Retrieval System - experiment in Automatic Document processing.* Prentice-Hall, New Jersey, 1971.

[Salton 83] G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval.* Mc Graw Hill Computer Series, Mc Graw Hill, New York, 1983.

[Saussure 49] F. De Saussure, *Cours de Linguistique Generale, Quatrième edition.* Librairie Payot, Paris, France, 1949.

[Smadja 89] F.A. Smadja, *Lexical Co-occurrence: The Missing link.* Submitted to the Journal of the Association for Literary and Linguistic computing, 1989.

[Sparck Jones 86] K. Sparck Jones, *Synonymy and Semantic Classification.* Edinburgh University Press, Scotland, 1986.