



Friedrich-Alexander-Universität

Erlangen-Nürnberg

Institut für Germanistik

Abteilung für Computerlinguistik

Michael Piotrowski

NLP-Supported Full-Text Retrieval

Master's Thesis

CLUE

The amount of information available in electronic form is growing exponentially, making it increasingly difficult to find the desired information. This is especially true of the World Wide Web, which has no central administration and thus no ordering scheme to help users find the information they need. Furthermore, most of the information is *narrative*, i.e., in the form of unstructured documents written in natural languages, as opposed to structured information stored in databases.

Information retrieval is primarily concerned with the storage and retrieval of unstructured information. Thus, along with the growth of the World Wide Web, information retrieval systems gain importance since they are often the only way to find the few documents actually relevant to a specific question in the vast quantities of text available. *Internet search engines* like *AltaVista* or *Lycos* are very popular and commercially successful.

Although information retrieval systems mainly deal with natural language, linguistic methods are rarely used. Most systems only use *stemming*, i.e., the mechanical cutting off of inflectional and derivational suffixes to better match index terms to query terms. Since most research on information retrieval is done for English, which has a relatively weak morphology, this is seldom regarded as problematic. Some researchers even consider stemming as completely unnecessary. There is, however, considerable evidence that stemming and more linguistically motivated methods do have a positive impact on retrieval performance for languages such as Dutch, German, Italian, or Slovene, which are morphologically richer than English. Morphologic phenomena like compounds and changes of the stem are still not handled by conventional stemmers. As German, for example, makes extensive use of these morphologic processes (consider compounds like *Bundesverfassungsgericht*, and changes of the stem like in *Häuser*, the plural of *Haus*), the application of full morphologic analysis to the information retrieval task intuitively seems to be promising.

This thesis sets out to determine the usefulness of morphologic analysis in information retrieval systems, particularly for the retrieval of German-language documents. An experimental retrieval system called IRF/1 was developed as a test bed. It is described in this thesis. IRF/1 is used to compare the retrieval effectiveness of different text processing methods for a test collection of about 300 magazine articles. The evaluated methods are:

1. stemming (as a baseline),
2. base form reduction using morphologic analysis,
3. same as (2) but compounds are split into the base forms of their constituents, and
4. same as (3) but the base forms of compounds are kept along with their parts.

Using the standard information retrieval measures of *recall* and *precision*, the comparison finds morphologic analysis to be generally more effective than stemming. While morphologic base form reduction only provides relatively little improvement over stemming, decomposition of compounds results in a decisive increase in retrieval effectiveness for German.

It can be concluded that morphologic analysis with decomposition of compounds is a very promising approach to improving information retrieval for German and should be further investigated.

Zusammenfassung

Immer mehr Wissen ist in elektronischer Form verfügbar, wodurch es immer schwieriger wird, die gewünschten Informationen zu finden. Diese Aussage trifft besonders auf das World Wide Web (WWW) zu. Das WWW wird nicht zentral verwaltet, weshalb es auch kein Ordnungsschema gibt, das es den Benutzern erleichtern würde, die gewünschten Informationen zu finden. Dazu kommt, daß das meiste Wissen in *narrativer* Form niedergelegt ist, d. h. in Form unstrukturierter natürlichsprachlicher Texte, im Gegensatz zu strukturierten Daten, die in Datenbanken gespeichert sind.

Information retrieval (IR) beschäftigt sich hauptsächlich mit der Speicherung und dem Wiederauffinden unstrukturierter Informationen. Daher hat die Bedeutung von IR-Systemen mit dem Wachstum des WWW stark zugenommen, da sie häufig die einzige Möglichkeit sind, die wenigen für eine bestimmte Frage tatsächlich relevanten Informationen zu finden. *Internet-Suchmaschinen* wie *AltaVista* oder *Lycos* sind deshalb äußerst beliebt und kommerziell sehr erfolgreich.

Obwohl IR-Systeme vor allem mit natürlicher Sprache umgehen, werden linguistische Methoden selten benutzt. Die meisten Systeme begnügen sich mit *Stemming*, dem mechanischen Abschneiden von Endungen, um Anfragen und im Index gespeicherte Wörter besser in Übereinstimmung bringen zu können. Da der größte Teil der Forschung im Bereich des IR für das Englische betrieben wird, das nur über eine relativ schwach ausgeprägte Morphologie verfügt, wird dies selten als ein Problem betrachtet. Einige Forscher halten *Stemming* sogar für vollkommen überflüssig. Es gibt jedoch deutliche Hinweise dafür, daß *Stemming* und linguistische Methoden positive Auswirkungen auf die Effektivität der Suche für Sprachen wie z. B. Deutsch, Italienisch, Niederländisch oder Slowenisch haben, die eine ausgeprägtere Morphologie als Englisch haben. Morphologische Phänomene wie Komposition und Stammänderungen werden aber von konventionellen *Stemmern* nicht behandelt. Da z. B. im Deutschen diese morphologischen Prozesse sehr häufig auftreten (man denke an Komposita wie *Bundesverfassungsgericht* und Stammänderungen wie in *Haus – Häuser*), erscheint der Einsatz von vollständiger morphologischer Analyse auf linguistischer Grundlage für das IR intuitiv sehr attraktiv.

Diese Arbeit versucht, den Nutzen von morphologischer Analyse in IR-Systemen zu bestimmen, insbesondere für die Suche nach deutschsprachigen

Texten. Zu diesem Zweck wurde ein experimentelles IR-System namens IRF/1 entwickelt, das in dieser Arbeit beschrieben wird. IRF/1 wird dann dazu verwendet, die Sucheffektivität von verschiedenen Textanalysemethoden für eine Sammlung von etwa 300 Zeitschriftenartikeln zu vergleichen. Die untersuchten Methoden sind:

1. Stemming (als Vergleichsgrundlage)
2. Grundformreduktion mit Hilfe morphologischer Analyse
3. Wie (2), aber Komposita werden in die Grundformen ihrer Bestandteile zerlegt
4. Wie (3), aber zusätzlich zu ihren Teilen wird die Grundform von Komposita verwendet

Beim Vergleich der Methoden mit Hilfe der Standard-IR-Maße *Recall* und *Precision* wird festgestellt, daß die morphologische Analyse allgemein effektiver ist als Stemming. Die morphologische Grundformreduktion ist zwar nur geringfügig besser als Stemming, dafür erhöht die Kompositazerlegung die Effektivität der Suche für das Deutsche ganz entscheidend.

Man kann also den Schluß ziehen, daß die morphologische Analyse mit Kompositazerlegung in der Tat ein sehr vielversprechender Ansatz zur Verbesserung von deutschsprachigem IR ist, die weiter untersucht werden sollte.

Contents

1	Introduction	9
1.1	The Information Age	9
1.2	Goals	10
1.3	The CLUE Framework	10
1.4	Overview	11
2	Information Systems	13
2.1	Introduction	13
2.2	Applications	13
2.3	Information Retrieval	14
2.3.1	Overview	14
2.3.2	Implementation Issues	15
3	Information Retrieval and NLP	19
3.1	Morphology	19
3.2	Lexicon	20
3.3	Syntax and Semantics	20
3.4	Conclusions	22
4	Design and Implementation	25
4.1	Overview	25
4.2	Design Decisions	25
4.3	Preprocessing	28
4.4	Automatic Language Identification	30
4.4.1	Overview	30
4.4.2	Design and Implementation	30

4.4.3	Performance	33
4.4.4	Possible Enhancements	34
4.5	Indexing	34
4.5.1	Morphologic Analysis	34
4.5.2	Storage	37
4.6	Retrieval	43
5	Evaluation	51
5.1	Introduction	51
5.2	Evaluation Criteria for IR Systems	52
5.3	Relevance	53
5.4	Standard Effectiveness Measures	53
5.5	Test Collections	55
5.5.1	Overview	55
5.5.2	Reducing the Need for Human Relevance Judgments	56
5.5.3	Conclusions	58
5.6	Evaluation of IRF/1	58
5.6.1	Expected Behavior	58
5.6.2	Test Collection	59
5.6.3	Measuring Procedures	60
5.6.4	Evaluation Results	63
6	Conclusions	65

1

Introduction

Knowledge is power—nam et ipsa scientia potestas est.

—Francis Bacon, *Meditationes sacrae*

1.1 The Information Age

Never before has it been possible to store and distribute the amounts of information computers and networks handle today. With their support it is also possible to generate more information faster than ever, and deliver it almost instantly anywhere in the world. While the production of material goods is declining in Western countries, the production, dissemination, and processing of information is rapidly gaining in importance: we are clearly on our way from an industrial society to an information society.

Back in the 16th century, Francis Bacon, who is quoted above, had already recognized the importance of knowledge. In his times information was difficult to find because information was scarce and only accessible to the privileged few. Today, the *World Wide Web (WWW)* provides instant access to information for a rapidly increasing number of people (there were an estimated 30–40 million Internet users in 1997, with a growth rate of 1 million new users per month¹) and from hundreds of thousands of sources. *AltaVista*, the largest search engine on the WWW, boasts to have indexed over 100 million documents. Nevertheless we face problems similar to those of Bacon's contemporaries, although for a different reason: there is too much information; we are overwhelmed by information, most of which is irrelevant to our current information needs. This situation, where it is impossible to find the relevant information because there is too much information to evaluate, is often referred to as *information overload*.

¹ According to studies compiled by MSS Internet Services, available online at <http://www.4-mss.com/internet/htmls/statistics.html>

The problem of information overload is more acute than ever. In fact, the WWW is at risk of becoming unusable for serious work because too much “noise” is making the relevant information nearly impossible to find:

The most immediate cause of information overload on the Web is caused by the Web trying to fill the dual role of being both a private and public information and communication medium. Issues that are privately important tend to be publicly uninteresting. When the background noise of the medium drowns out most of the useful content for the wider audience, as is now happening on the Web, the effectiveness of the medium is undercut.

This, incidentally, is the same problem that ultimately ruined the citizen’s band radio industry. The CB became a relatively useless communication medium because the industry did not anticipate, and could not handle, concurrent escalation in both volume of traffic and the proportion of noise. Fortunately, such propensity for self-destruction may be avoided in cyberspace because of its digital nature. Computational relief is forthcoming from all quarters. [4, p. 20]

Since most of the world’s information is probably stored in unstructured documents written in natural languages—as opposed to databases—*natural language processing (NLP)*, i.e., the computational analysis of language, will play an increasingly important role in the information systems which will help us to deal with information overload: information retrieval systems, information filtering systems, personal information agents, and new types of systems yet to be devised.

1.2 Goals

The goal of this thesis is to evaluate whether linguistic methods can improve the performance and user-friendliness of full-text information retrieval systems.

The importance of full-text retrieval has been greatly increased by the advent of the World Wide Web, where most information is in the form of unstructured natural language documents. However, few full-text retrieval systems apply linguistic methods to the retrieval task, and if they do, they are at best very rudimentary. An example might be the so-called *stemming*, i.e., the cutting off of suffixes with the help of a list of endings.

The question is whether the consistent application of linguistic methods can improve the search results and thus the usability of information retrieval systems. While stemming may yield acceptable results for English, I believe that it can only deliver sub-optimal results for highly inflectional languages such as German or Italian because changes of the stem or compounding are not handled. The results of experiments with the *SPIDER* system [56] show for example that retrieval performance for Italian is up to 130% better with stemming than without. However, over 220 non-intuitive stemming rules are necessary to achieve these results.

1.3 The CLUE Framework

This thesis is not isolated but has to be seen in the context of the Workbench activities at the Department of Computational Linguistics at the University of

Erlangen (CLUE). The Workbench is a strategic project which aims at concentrating the research and development efforts at CLUE towards a comprehensive system, and to promote the reusability and connectivity of the software. From a functional point of view the goal of the Workbench is twofold:

1. Provide an environment for the development of linguistic components and resources, e.g., grammars, lexicons, corpora, etc.
2. Provide a platform for end-user applications which use linguistic methods to improve their services. These applications would not only profit from the linguistic services the tools of the Workbench provide, but would also automatically get other benefits from the Workbench, such as networking, collaborative multi-user support, etc.

The heart of the projected Workbench design are the *Malaga* grammar development system [5], which will provide the linguistic processing, and a relational database management system, which will provide a uniform and comprehensive storage model for all necessary data, e.g., lexica, corpora, test sentences, etc.

The experiences made with IRF/1, the experimental information retrieval system developed for and described in this thesis, will help to implement the Workbench: IRF/1 uses *Malaga* for morphologic analyses and stores its data (the index) in a relational database. Apart from this, the retrieval functionality could be used to allow full-text searching of the Workbench documentation, or of the literature (papers, reports, etc.) needed by the users of the Workbench.

1.4 Overview

The structure of this thesis can be outlined as follows: Chapter 2 gives an overview about different kinds of software systems dealing with information, one of which, information retrieval systems, is treated in detail. Chapter 3 analyzes the relationship between information retrieval and natural language processing. Chapter 4 describes the implementation of IRF/1, an experimental retrieval system developed for this thesis, which incorporates CLUE natural language processing tools. Chapter 5 describes the principles and problems of evaluation for information retrieval systems in general, and for IRF/1 in particular. Finally, chapter 6 presents the conclusions which can be drawn from the work done for this thesis.

2

Information Systems

*“Mr. Helpmann, I’m keen to get into Information Retrieval.
Mr. Helpmann, I’m dying to get at this woman . . . no, no,
no.”*

—Terry Gilliam, *Brazil*

2.1 Introduction

Computer programs which handle information are commonly subsumed under the term *information systems*. The tasks of information systems are:

- storage,
- processing,
- retrieval, and
- distribution

of information items. The “information items” can be very diverse. The goal of information systems is to help people by satisfying their information needs, which ultimately means to help them to solve their problems. Although index cards and printed catalogs could also be considered information systems, only electronic information systems will be discussed here. Non-electronic reference material is losing its former importance very quickly against the obvious advantages of electronic systems.

2.2 Applications

Information systems have a broad range of applications, e.g.:

- they support managers in making decisions by making vital company data accessible,
- they support research in libraries by making bibliographic data available for searching,
- they support lawyers and judges by allowing them to retrieve information about similar cases from large databases of legal decisions,
- they support employment agencies by helping to match available jobs to job seekers,
- they support engineers and maintenance crews of complex technical systems with fault databases and complete, searchable manuals which can be consulted on site,
- they provide vendors with information about how many units of a particular product are on stock, and
- they help empirical linguists by making it possible to manage, search and evaluate large corpora.

Naturally, different applications require different types of information systems. According to their different applications, these systems have different characteristics. Examples are:

- database management systems (DBMS) for structured data like personnel records or inventories,
- on-line public access catalogs (OPAC) for libraries,
- decision support systems for managers,
- corpus research tools for corpus linguistics, and
- information retrieval (IR) systems for the retrieval of unstructured information like memos or reports.

Of all the possible information systems, this thesis is only concerned with IR systems.

2.3 Information Retrieval

2.3.1 Overview

Although the term *information retrieval* seems to be very wide, information retrieval generally focuses on *narrative* information. The items typically processed by information retrieval systems include letters, newspaper and magazine articles, books, medical summaries, research papers, Web pages, and so on. These items are generally referred to as *documents*.

Sometimes information retrieval (IR) is used as a more general term, covering all kinds of retrieval tasks, and *document retrieval* or *text retrieval* is used to refer to the task outlined above. Very often, however, information retrieval and document retrieval are used synonymously as document retrieval is the prevalent area of research. I do not make this distinction either.

One important property of documents is that the information is encoded in natural language, which exhibits the following problematic properties:

- Ambiguity: e.g., *shot* has many different meanings, including the act of firing a gun, a photograph, an attempt, or an injection.
- Imprecision: there are many different ways to express a concept, e.g., *sonographic detection of fetal ureteral obstruction*, *obstetric ultrasound*, and *prenatal ultrasonic diagnosis* all refer to the same concept of using ultrasound to diagnose pregnancy (from [58, p. 5]).
- Implicitness: much of the information in a text is not expressed explicitly. Although *start*, *begin*, and *initiate* could be considered equivalent, the preference of one term over another by an author might convey important information which is difficult to formalize.
- Vagueness: Natural language is often very vague, e.g., *rather large* or *relatively small* are not easily quantifiable in numbers.

As user requests for information, called *queries*, are also formulated in natural language or use natural-language terms, they pose the same problems, making it even harder to find the relevant information. Smeaton concludes from this observation:

It is because of all these indeterminates that IR is a difficult problem. It is also because of these indeterminates that users tolerate incorrect output from an IR system and do not expect 100% accuracy, i.e. all retrieved documents to be relevant.” [58, p. 4]

In a document retrieval system it is thus not certain that all relevant documents are found, or that all retrieved documents are actually relevant to the query. The ratio of documents retrieved versus the number of available documents relevant to the query, i.e., the fraction returned out of all desirable documents is called *recall*. The ratio of the number of relevant documents retrieved versus the total number of documents retrieved, or the useful fraction of what was actually retrieved is called *precision*. These two numbers are the most common measures for the performance of IR systems. This subject will be discussed in depth in chapter 5.

Database management systems, on the other hand, can give very precise answers to detailed queries but cannot provide information on the basis of queries that are only vaguely worded. A DBMS requires precisely stated queries, and the main issue—and performance measure—is therefore how to efficiently process the query and provide the requested data.

2.3.2 Implementation Issues

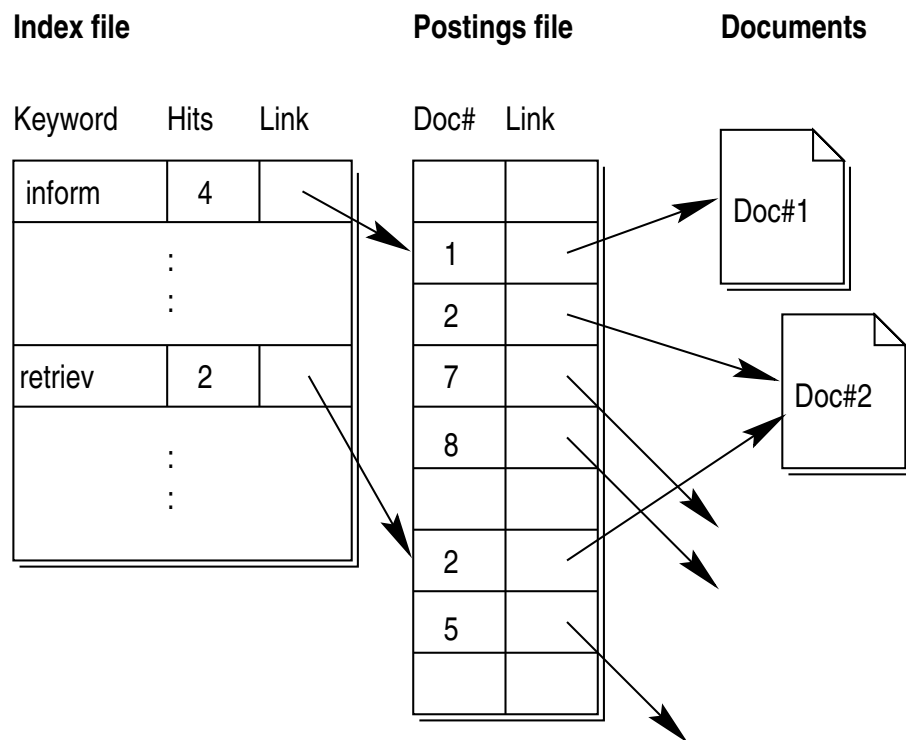
How does a typical information retrieval system work? Conceptually, an IR system is similar to a traditional library: there is a collection of documents and an access method. The simplest access method is linear searching, i.e.,

one document after another is scanned to see if it matches the query. The *UNIX grep* utility could be considered a retrieval system of that type. Using the library metaphor, this would correspond to reading all the books of the library to find the ones you actually need. This approach is possible for a small collection but is too inefficient for larger collections.

One solution to this problem would be to order the collection according to some criterion, e.g., alphabetically by author. When searching for all documents by a specific author this would make it unnecessary to read all documents. However, if all documents containing some term are desired, they still have to be searched sequentially.

A better solution is to build an index for the collection. An index corresponds to a library catalog: instead of having to go through the shelves one can look up the topic of interest in the catalog and finds the positions of the relevant documents. Similarly, an index of an information retrieval system allows to find the documents matching a particular query without having to look at the documents themselves. This speeds up the search considerably (by several orders of magnitude). However, an index has to be built before it can be used. One common index type are *inverted files*. Inverted files work as follows. Each document in the collection is assigned a list of attributes which are supposed to represent the document. The most common type of attributes are keywords. The inverted file is then the sorted list of keywords of all documents, where each keyword has links to the documents that contain that keyword (see figure 2.1).

Figure 2.1: Conceptual organization of an inverted file index (adapted from [29, p. 29])



The question is now how to select the attributes for a document. As today's collection sizes are too large for manual indexing I will only discuss automatic indexing. This means that the only practical methods are those where the index terms—the keywords used for indexing—are extracted from the documents themselves.

Controlled vocabulary approaches used to be quite popular for the selection of index terms from documents. Under this approach only words which are in a list of allowed index terms will be used for indexing. This method is closely related to traditional manual indexing methods using a controlled indexing language. While this method theoretically allows to build a very good index, it has several drawbacks. First, the controlled vocabulary has to be specified. This is only possible for a relatively small domain. Second, words that are not included in the vocabulary are not searchable, and third, searching cannot be done by end users but only by expert intermediaries who are acquainted to the indexing language and know how to specify a query.

The modern approach, however, is to use the full text of the documents as indexing terms (hence the term *full-text retrieval*). After the removal of *stop words* (words which are too frequent to be of any use, e.g., determiners and other function words), the remaining word forms are normally *conflated*, i.e. supposedly semantically related word forms are mapped to a common form (the actual index term).¹ Finally, the index terms are *weighted* according to their frequency and stored in the inverted file.²

Term conflation is usually done for two reasons: first, as stated above, different morphological forms of a word should be mapped to a common form. The assumption is that that the word forms are semantically related, and that in a search for, say, *stemming*, occurrences of *stemmed* and *stem* are also relevant. The second, though less important reason is that the conflation of terms reduces the size of the index.

Since most research in information retrieval is being done for English, which has a weak morphology, i.e., words have few morphologic variants, the most common stemming method is *suffix stripping*³. Suffix stripping uses a list of frequent inflectional and derivational suffixes which will be cut off from word forms to produce their stems. The two most frequently used algorithms in IR are the Lovins stemmer [47] and the Porter stemmer [52]. Although studies on the effectiveness of stemming to improve recall and precision produced equivocal results, most IR systems still include a stemmer. The opinions on stemming range from “no effect” (e.g., [25]) to “significant improvement” (e.g., [41]).

Results for other languages, especially morphologically more complex ones than English, are much clearer, though. These languages include Hebrew [10], Slovene [51], Dutch [38], German, and Italian [56]. The stemmers used are typically variations of the Porter stemmer, but there are some attempts to use more linguistics: in [38] a *dictionary-based stemmer* is included in the evaluation, and in [56] a dictionary-based stemmer is used for German. Both of these stemmers are based on the *CELEX* lexical databases [3] for Dutch and German, respectively, and remove suffixes by dictionary lookup. Using the dictionary, these stemmers also analyze compounds into their constituents. In the next chapter, I will discuss the results of these and other approaches at using natural language processing (NLP) for information retrieval.

¹ Alternatively, query terms can be expanded.

² This method is the one that is most frequently used. Syntactic and semantic approaches will be briefly discussed in chapter 3. Term weighting will be described in chapter 4.

³ Prefixes are sometimes also removed but they are usually considered to change the meaning of a word and better not be stripped.

3

Information Retrieval and NLP

In the last chapter we have seen that IR systems are required to handle natural language both in the documents and in the user queries. Consequently, there have been many attempts to meet the challenges posed by the properties of natural language with techniques from natural language processing (NLP). Problems arise on almost all levels of linguistic analysis, i.e., morphology, lexicon, syntax, semantics, and pragmatics. In the following I will try to summarize these problems and describe attempts to solve them.

3.1 Morphology

One of the first problems related to the use of natural language in information retrieval is that of morphologic variation. This refers to the fact that words may occur in inflected forms, or that derivation is used to produce new but related words, or that words are combined into compounds. Morphologic variations can very often be regarded as semantically related and thus equivalent for retrieval purposes. In English, the number of possible inflected and derived forms is relatively small, and variation is mostly restricted to the attachment of suffixes. Compounds which are not yet lexicalized are in most cases written as separate words. Stemming, i.e., the removal of suffixes using a list of possible suffixes, is therefore considered a practical way to map related word forms to a common stem.

As mentioned earlier, the effectiveness of stemming for retrieval of English-language documents is still being discussed because most evaluations (Table 8.1 in [21, p. 141] summarizes a number of such experiments) come to the conclusion that it provides only little if any improvement in recall and precision. Some conclude from these results that all NLP for information retrieval is more or less useless:

These results [claiming that stemming produces little if any improvement in precision/recall] are disturbing for those of us working in natural language processing (NLP). If it is hard to show that something as simple as stemming is helpful, how can we possibly justify our interests in more challenging forms of natural language

processing such as part of speech tagging, word sense disambiguation, synonymy, phrase identification and parsing? [11, p. 310]

The aforementioned experiments for languages other than English [38, 56, 10, 51], however, provide considerable evidence for the usefulness of stemming for more inflectional languages. In the *SPIDER* [56] and *UPLIFT* [38] projects it was also found that for German and Dutch, two languages which have very productive compounding processes, compounds should be split into their constituent parts. Since decomposition of compounds cannot be done by a traditional stemmer, full-form dictionaries [3] were used in these experiments. Compounds not included in the dictionaries were tried to split by matching parts to simple dictionary entries.

3.2 Lexicon

When morphological variation has been taken care of, there remains the problem of lexical variation. In natural language it is possible to refer to a specific concept using different words. This leads to search failures due to *vocabulary mismatch*, i.e., the user specifies terms in their query that are different from those that were used to index relevant documents. After all, users of IR systems look for concepts, not strings of characters. If, for example, a user issues a query for *cars*, normally it does not mean that they are interested in every occurrence of the string “c-a-r-s”, but they rather look for documents concerned with the concept of self-propelled land vehicles. This concept can partially—not exhaustively—be described by the words *car*, *automobile*, *pickup*, *minivan*, etc., but it may also be referred to by proper names such as *BMW*, *Hyundai*, or *Chevrolet*.

Attempts have therefore been made to use thesauri to enrich queries with related terms or to control the vocabulary, e.g., by normalizing variant terms. Although similar in some respects, thesauri used in IR systems should not be confused with thesauri intended for creative writing, such as Roget’s Thesaurus, which are unsuitable for retrieval.

Thesauri can either be constructed manually or automatically from a collection of texts. Manual thesaurus construction is a very labor-intensive task and “both an art and a science” [61, p. 166]. Automatic methods exploit statistical relationships between terms and are thus necessarily not able to express all semantic associations that might exist. Thesauri are always domain-dependent, i.e., a thesaurus used for a collection of medical documents will not improve the retrieval of legal documents. This implies that it is absolutely necessary to identify the domain of the collection, also to supply the correct synonyms, hyponyms, and hyperonyms for ambiguous words. Consequently, thesauri can only be used for relatively homogeneous collections, which are rather the exception than the rule today. Furthermore, if they are not very carefully constructed, thesauri are likely to reduce the precision of the IR system by generating hypotheses about the concepts the user looks for which may not be true.

3.3 Syntax and Semantics

NLP researchers tend to consider conventional retrieval systems “inadequate for the obvious reason that they do not do NLP, and so cannot tell a Venetian blind from a blind Venetian” [60]. The logical consequence is trying to make syntactic and semantic relationships explicit, e.g., by building phrase or concept indices. Actual systems implement this approach to various degrees: The *CLARIT* system and experiments based on it (e.g., [19]) aim just for a shallow understanding of the texts by approximating concepts through the analysis of the phrasal structures of the documents. At the other end of the spectrum lies the *FERRET* project [48], in which deep conceptual understanding was tried to be achieved.

While this approach is theoretically convincing, it has failed to produce significantly better results than statistical methods. The use of syntax and semantics is based on the assumption that an indexing for retrieval must directly and explicitly capture the syntactic and semantic relationships contained in the documents, as it is done in manual indexing. The results raise doubts about the usefulness of doing this:

[...] decades of past experiment have shown that complex index descriptions modelled on manual prototypes are far too constraining, while complex terms (e.g., simple phrases with a head-modifier structure) do not work much better than coordinated simple terms. These findings have been confirmed by tests under the current ARPA/NIST Text REtrieval Conference (TREC) evaluation programme, where many alternative specific approaches are being assessed using very large full-text files. [60, p. 13]

A further problem is that semantics is a field which has not yet been fully understood, especially if large quantities of free text are to be processed. Systems which can handle at least some aspects of semantics are still too slow and unstable for industrial-strength IR. Syntax is much better understood than semantics, and *CLARIT*, for example, is a commercial product, but most of the points which make the use of semantics in production systems unwieldy (too difficult, too large, too slow) also apply to syntactic analysis.

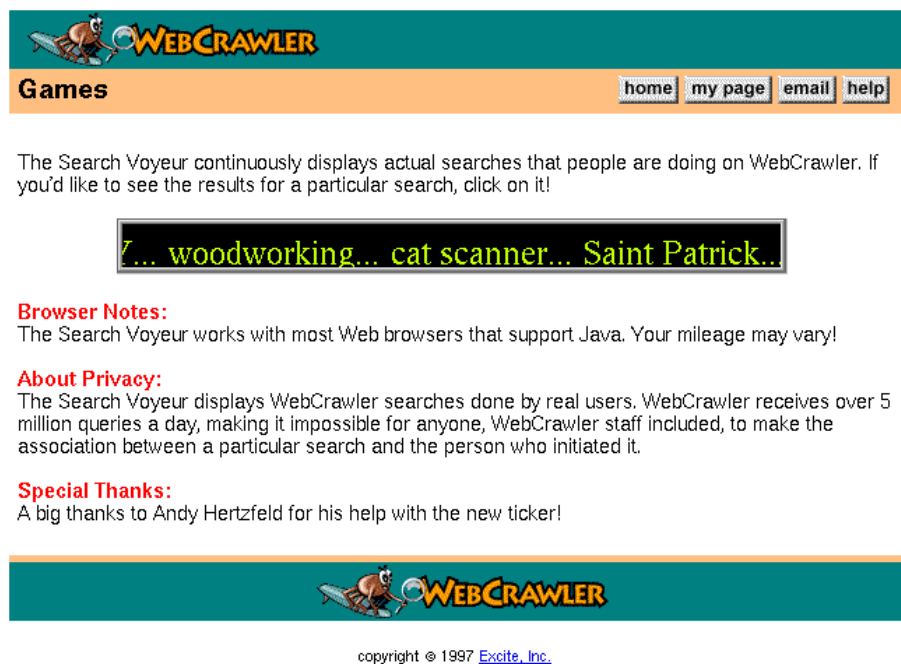
An additional point to consider is that both the users and the data of text retrieval systems are changing: The World Wide Web has made text retrieval an end-user application which is used to retrieve short-lived multimedia documents in many languages out of an extremely large document collection. If one watches the queries users issue to the *WebCrawler* search engine¹ on the *Search Ticker* (see figure 3.1), one can observe that most queries consist of just one or two words, which doesn't leave much room for syntactic and semantic interpretation.

Although there are advances in the use of syntactic and semantic analysis for use in IR, Salton and McGill's opinion from 1983 still seems to be valid:

Various attempts have been made to use simple syntactic analysis systems in actual information retrieval situations. While linguistic methods may eventually prove essential in automatic indexing,

¹ <http://www.webcrawler.com/>

Figure 3.1: WebCrawler
SearchTicker



Games [home](#) [my page](#) [email](#) [help](#)

The Search Voyeur continuously displays actual searches that people are doing on WebCrawler. If you'd like to see the results for a particular search, click on it!

7... woodworking... cat scanner... Saint Patrick...

Browser Notes:
The Search Voyeur works with most Web browsers that support Java. Your mileage may vary!

About Privacy:
The Search Voyeur displays WebCrawler searches done by real users. WebCrawler receives over 5 million queries a day, making it impossible for anyone, WebCrawler staff included, to make the association between a particular search and the person who initiated it.

Special Thanks:
A big thanks to Andy Hertzfeld for his help with the new ticker!

copyright © 1997 [Excite, Inc.](#)

the available evidence indicates that the simplified syntactic analysis systems do not yet provide the answer. The frequency-based phrase-generation methods are simpler to implement and are currently more effective. [54, p. 91]

3.4 Conclusions

While NLP-based approaches to IR are theoretically promising they have yet to prove that they are able to practically improve retrieval results. Thesauri are most useful for controlled-language indexing, which is rapidly disappearing. Syntactic and semantic analysis are still too expensive and not yet able to process unrestricted text. Good results have been achieved with shallow parsing of noun phrases [19], although it does not dramatically improve retrieval effectiveness. It actually seems that “[...] NLP techniques are challenged by the basic methods of statistical IR, which has apparently picked some of the low-hanging fruit off the tree.” [45, p. 99]

The results of the *UPLIFT* and *SPIDER* projects, indicating that splitting compounds can significantly improve Dutch and German text retrieval, look more interesting. In both projects, only full-form dictionary lookup with simple rules for compounding was used. I think that for several reasons this approach might be improved by full rule-based morphologic analysis:

- Rule based analysis can handle a potentially infinite number of forms, which is important if unrestricted texts are to be analyzed.
- For languages which have even more inflectional forms than German or Dutch, *and* compounding, e.g., Finnish, the lexicon would have to be extremely large.
- If compounding cannot be handled by simply concatenating word forms, more powerful rule mechanisms will be necessary.

- It is probably desirable to handle all morphological processes in a uniform way on a linguistic foundation.

Compared to semantics, morphology is relatively well understood in traditional and computational linguistics, and there are implementations of morphologic grammars which are potentially fast and stable enough to be used on a day-to-day basis in a real-world IR system. Furthermore, when syntactic and semantic analysis will eventually be usable, morphologic analysis will be an essential precondition.

This reasoning resulted in the implementation of IRF/1, an experimental retrieval system which uses morphologic analysis to reduce word forms to base forms and to decompose compound words. The design, implementation, and evaluation of IRF/1 will be described in the following chapters.

4

Design and Implementation

If we are to achieve results never before accomplished, we must employ methods never before attempted.

—Francis Bacon

4.1 Overview

The conclusion of chapter 3 was that morphologic analysis with decomposition of compounds is likely to improve retrieval effectiveness for German. To evaluate this hypothesis, an experimental retrieval system called IRF/1 was implemented. Another aspect in the design of IRF/1 was to test the suitability of the CLUE NLP components for use in the text retrieval environment, especially with regard to their ability to process large amounts of text and to process unrestricted text. These components will be described in section 4.2. In fact, IRF/1 is the first non-linguistic application developed at CLUE that embeds the CLUE NLP components.

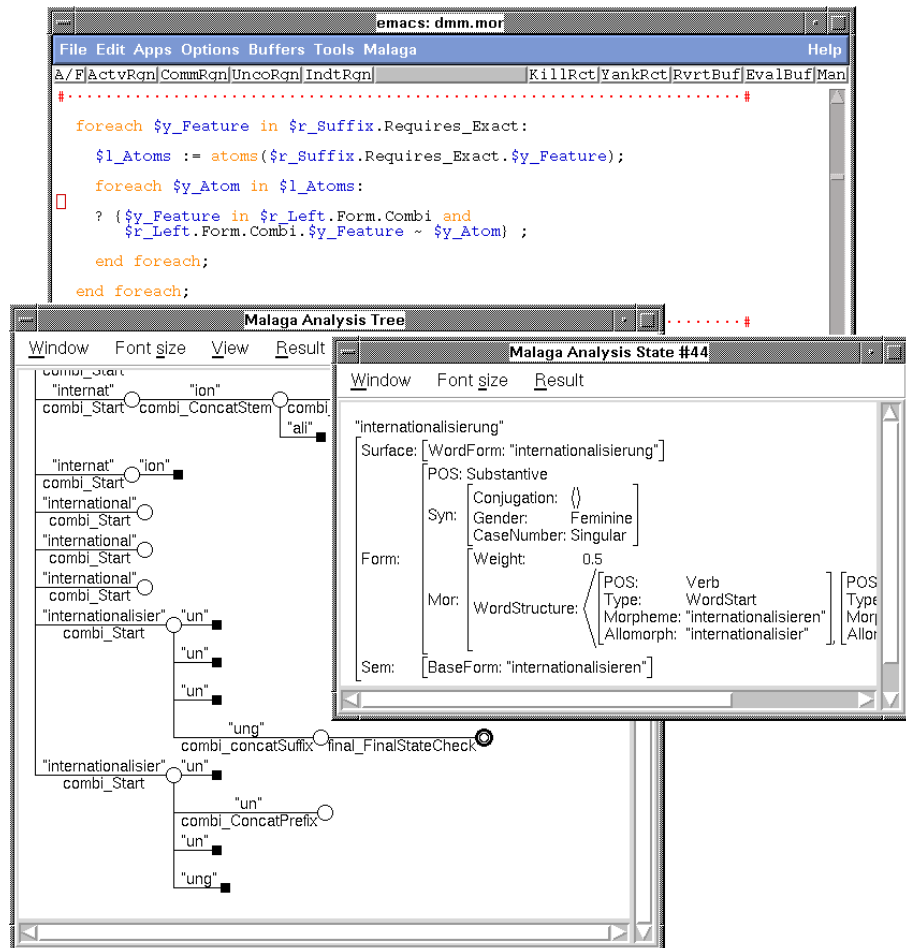
4.2 Design Decisions

The basis for the linguistic components of IRF/1 is the *Malaga* grammar development system [5, 55]. *Malaga* was developed at CLUE and consists of a specialized programming language for natural language grammars, a compiler for that language, a development environment including debugging facilities, a visualization tool (see figure 4.1), and a library for using *Malaga* grammars from C applications. *Malaga* is based on the formalism of *Left-Associative Grammar* [30, 31], which is characterized by formal simplicity and computational efficiency, while being linguistically well-motivated. The available grammars written in *Malaga* currently include morphology grammars for German, Italian and Korean.

IRF/1 is designed to serve as a test bed for different processing approaches. The evaluated methods are described in chapter 5. In some ways, IRF/1 is similar to other current European research projects mentioned above. Table 4.1 contains a point-for-point comparison.

The IRF/1 architecture is highly modular: components can easily be replaced or updated when needed, and different implementations of components can be evaluated. Figure 4.2 shows the basic module architecture of IRF/1.

Figure 4.1: Malaga Development Environment



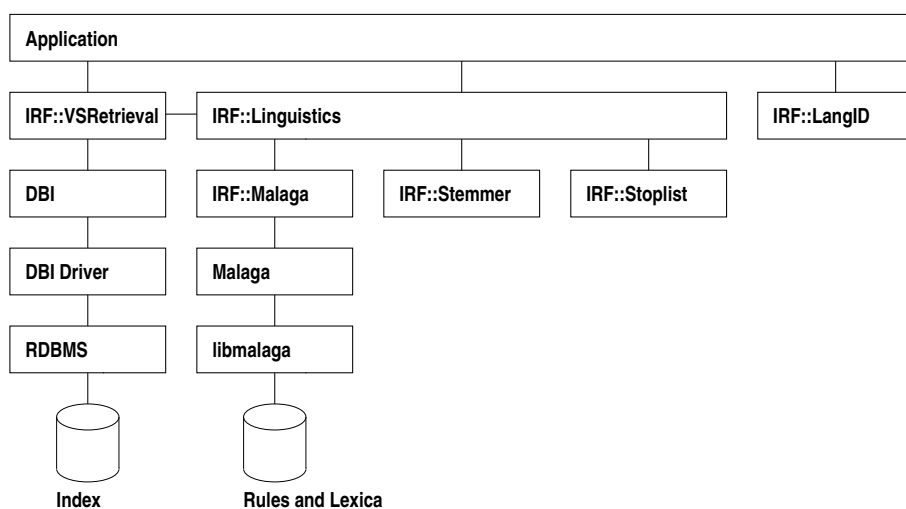
As you can see from table 4.1, a normalization approach was taken for query processing. This means that only processed forms (e.g., base forms) will be stored in the index, which can greatly decrease its size; this is important when document collections become very large. Since morphologic analysis provides information about the part of speech (POS) or word class of a word, it is possible to employ a list of *stop classes*, i.e., those word classes which have only or mostly grammatical functions, or are generally useless for retrieval purposes due to their frequency, like determiners or prepositions. Stop classes are similar to stop words—which can be used at the same time—but allow to control the inclusion or exclusion of large numbers of words with a single list entry. Naturally this further reduces the size of the index.

Recognizing that multilingual applications will be rather the rule than the exception in the future, IRF/1 was designed from the ground up to accommodate multilingual applications. To ensure language independence, IRF/1 automatically detects the language of a document, and accordingly uses the language-

Table 4.1: Comparison of text retrieval research

Area	Systems		
	<i>UPLIFT</i>	<i>SPIDER</i>	IRF/1
NLP	dictionary-based stemming, stemming	dictionary-based stemming, stemming	morphologic analysis, stemming
Phrase indexing	no, rejected	planned	no
Dictionary	CELEX Dutch	CELEX German	Proprietary
Dictionary size (stems)	124 000	51 000	49 000
Dictionary size (word forms)	380 000	360 000	unlimited
Query processing	expansion	normalization	normalization

Figure 4.2: IRF/1 Module architecture overview of the RDBMS-based implementation



specific methods available for that language, or falls back to a generic mode when none are available.

IRF/1 was implemented in Perl 5 [63], a high-performance interpreted language especially suited for text processing. Although Perl can be used to quickly “hack together” a script for some specific task due to its interpretative nature and the fast turn-around times that result from it, it also offers very high-level object-oriented concepts, including closures and multiple inheritance.

Perl also offers modules and packages for namespace separation and convenient code reuse. The easy and seamless integration of C routines removes all possible limits for extensions. In fact, Perl is probably the only language for which such a huge public library of modules exists. The *Comprehensive Perl Archive Network (CPAN)* (a replicated Internet FTP archive¹) contains over 670 modules by hundreds of authors which all follow the conventions of object-oriented Perl programming and are thus ready to be reused with a simple use statement. This demonstrates that large-scale code reuse is actually possible in Perl—something that other object-oriented languages like C++ and Java, which have had much more publicity, have yet to prove.

¹ The master server is <ftp://ftp.funet.fi/pub/languages/perl/CPAN/>; it is also mirrored at <ftp://ftp.uni-erlangen.de/pub/source/Perl/CPAN/>

A further advantage of Perl is its high portability: The interpreter runs on nearly every flavor of *UNIX* and on many proprietary systems, too. Consequently, Perl programs run on all of these platforms without any changes. Experiences with the *Corsica* [66] and *Amalgam* [37] projects have shown that there are still considerable problems with C++ and Java in this respect.

Since the *Malaga* distribution contains a C library as an interface to its internal functions, a Perl module (`Malaga.xs`, `Malaga.pm`) was written that allows the direct analysis of word forms and sentences with *Malaga* grammars from Perl. The functions provided by this module return the *Malaga* feature-value structures as a complex Perl data structure, which allows easy processing using the full power of the Perl programming language. This module is already being widely used at CLUE for various applications besides IR/1.

4.3 Preprocessing

Preprocessing is an important part of all text processing. Nevertheless, it is often neglected. In the preprocessing stage file formats, character sets, and markup can be converted, so that all text, regardless of its source, is in the same format. In later stages all further processing can then be consistently applied to all of the data, without the need to handle exceptions.

But if NLP is involved, preprocessing of the input text becomes even more important, if not critical. Most NLP systems place rather tight restrictions on the type of input they accept; if it does not comply with the requirements, the processing might either abort, or produce unusable output. However, if NLP is just one of several processing steps, “garbage in, garbage out” is not an option since the following steps must be able to rely on the NLP output. Two requirements follow from this:

1. NLP must be robust, and it must be precisely stated what it can handle, and what it cannot.
2. Preprocessing must ensure that the source text be presented to NLP in a form usable for it. For example, NLP programs usually need their input to be *tokenized*, i.e. text elements (usually word forms or sentences) are identified and placed on separate lines of the input.

In the IR setting, it must be absolutely avoided that documents become unretrievable because they were garbled by the NLP, and therefore not correctly indexed. The following list contains some of the things NLP systems have generally problems handling:

- Markup and layout information (bold, italics, capitalized, spaced out, etc.)
- Proper names; this includes names of persons, such as *Le Chevalier des Etoiles* or *Prof. Dr. Hausser, Ph.D.*, geographical names like *Fürth (Bay)* or *Texas City, TX*, and names of companies and institutions as *The Open Software Foundation, Inc.* or *Texas Instruments Holland B. V.*; especially confusing examples are *Be, Inc.* and *Next, Inc.*
- Abbreviations (*etc.*, *Inc.*, *GmbH* (German usage), *Ges. m. b. H.* (Austrian usage), *c/o*, ...)

- Monetary amounts (*1,50 DM, \$3.5 million, £5, ...*)
- Measurements (*40m², 3 sq. mil., 9μF, 30 °C, ...*)
- Dates; there is extremely wide variety in the notation of dates, and they are often ambiguous. Examples are *4.11.97* (German), *1997-11-04* (ISO/IEC 8601 [34]), *11/04/97* (American or British, meaning either November 4, or April 11), or *111600Anov1997* (NATO date-time group, denoting November 4, at 1600 hours local standard time).
- Foreign words or quotations (*Galileo's famous quote: « Oppure se muove! »*)
- Special elements and notations, like chemical formulas (H_2O , $C_4H_4N_2$, FeS_2)
- Non-obvious sentence boundaries
- Unknown words

It is the task of the preprocessor to identify these troublesome points and to ensure that they are correctly handled by following processing steps. With respect to NLP, this can mean:

- supplying “world knowledge” (e.g., by consulting a dictionary of abbreviations or proper names),
- giving hints for the analysis (e.g., by marking words which belong together, or by indicating that they should not be analyzed at all), and
- normalizing variant forms.

For IR applications it is especially important that terms likely to be searched for, like company and product names, are properly identified and possibly normalized. Depending on the domain of the indexed texts, one might also consider to extract factual information, such as dates and numbers, and index them in a normalized form.

Following are two examples for how some of the constructions mentioned above (and more) can occur. Note especially the scope of the genitive 's.

The Del The Del Fuegos, O Positive, and We Saw The Wolf will perform acoustic sets in Amnesty International USA Group 133's Seventh Annual Benefit Concert at 8 p.m. on Friday, March 19, at the First Parish Unitarian Universalist Church in Arlington Center. [6, p. 26]

An industry analyst, Robert B. Morris III in Goldman, Sachs & Co.'s San Francisco office, said ... [6, p. 33]

Although I was aware of the problems outlined above, there was no time for me to implement a sophisticated preprocessor. Therefore, IRF/1 currently uses a rather ad-hoc program that simply strips its input of all unneeded elements. Thanks to the modular architecture of IRF/1, however, it is always possible to replace the current preprocessor with a more powerful one, e.g., the one that is currently being developed at CLUE. IRF/1 does not use the preprocessor/tokenizer developed in [66] because of bugs which can result in untokenized or missing text.

Language identification, however, was considered too important for the whole system—and useful in general—to be left out. Since no free implementation was available, a language identifier was written, which will be described in section 4.4.

4.4 Automatic Language Identification

4.4.1 Overview

Linguistic methods are inherently language-dependent. For a text to be correctly analyzed it is necessary to know in which language it is written. For example, analyzing German text with an English grammar will probably lead to undesirable effects. For small, homogeneous collections the text language can be specified manually, but this is not possible for large text collections like the WWW, which contains documents written in many different languages, most of them without a formal indication of the language.

Furthermore, automatic identification of language is not only necessary to ensure that the correct grammar is applied, but also allows to annotate the indexed documents, so that it is possible to restrict the search to documents in specific languages, as it is offered by Digital Equipment Corporation's *AltaVista* search engine, for example.

While most of the work is being done for spoken language, automatic language identification for written texts also has many other possible uses, which explains why there is research going on at Sun Labs [62] and Xerox [65]. There are at least two commercial language identification tools by large companies [50, 65].

The implementation described in the following sections is based on the algorithms described in [18].

4.4.2 Design and Implementation

When presented with the following 20-character text samples:

```
den anforderungen ih  
r being a successful  
nous republions le d  
messaggi chimici che
```

most people would possibly be able to identify them as German, English, French, and Italian, respectively, even if they did not understand these languages.

This ability can be modeled on the computer by low-order character-level Markov chains (for an in-depth treatment of statistical methods in linguistics see for example [40]). Of course, this does not capture the structure of a language very well, but this is obviously not necessary for language identification; it is more the “look” that is important here.

A Markov chain is a random (stochastic) process in which the probability of the next state depends only on the current state. More formally, a Markov chain defines a stochastic variable whose values are strings from an alphabet Ω , and where the probability of a particular string S is

$$P(S) = P(s_1 \dots s_n) = P(s_1) \prod_{i=2}^n P(s_i | s_{i-1}) \quad (4.1)$$

The conditional probabilities $P(s_i | s_{i-1})$ are called *transition probabilities*.

A simple example of a Markov chain might be used to model the British weather according to [24]. We assume that the weather is observed at intervals of half an hour and that it can be in one of three states (making up the alphabet Ω): s_0 , it's neither raining nor foggy (which is extremely unlikely), s_1 , it's foggy, and s_2 , it's raining. The weather at a point of time t is characterized by a single one of these three states. We can now build a transition matrix:

$$\mathbf{W} = (w_{ij}) = \begin{pmatrix} 0.1 & 0.45 & 0.45 \\ 0.25 & 0.5 & 0.25 \\ 0.25 & 0.25 & 0.5 \end{pmatrix} \quad (4.2)$$

Using this model, we can ask and answer questions about British weather patterns over time. For example, what is the probability (according to the model) that, if it's currently neither raining nor foggy, at the next two checks it's raining, and then foggy?

$$\begin{aligned} P(s_0, s_2, s_2, s_1 | \mathbf{W}) &= P(s_0)P(s_2 | s_0)P(s_2 | s_2)P(s_1 | s_2) \\ &= \pi_0 w_{02} w_{22} w_{21} \\ &= 1.0 \cdot 0.45 \cdot 0.5 \cdot 0.25 \\ &= 0.05625 \end{aligned} \quad (4.3)$$

where π_0 denotes the initial state probability for s_0 , which is 1 in this case because it was actually observed.

A random process where the probability of the next state depends only on the last k states can also be described by Markov chains by using the last k states as the current state. These Markov chains are called Markov chains of order k or n -gram models (where $n = k + 1$; an n -gram is a sequence of characters of length n).

The transition probabilities look like this:

$$P(s_{i+1} \dots s_{i+k} | s_i \dots s_{i+k-1}) = P(s_{i+k} | s_i \dots s_{i+k-1}) \quad (4.4)$$

If we assume that the distributions of strings in the language satisfy the distribution for a k -order Markov chain, we can now use a k -order Markov chain, where k is relatively small (1 or 2, 2 in our implementation), to produce a simple model of language. In this case, the alphabet Ω is the set of characters in the language (e.g., a subset of ISO 8859-1 [33]).

We do this by building a transition matrix similar to the one for the weather example above, only that it has more states since the alphabet is larger, and that it is $k + 1$ -dimensional. However, it can be collapsed into a 2-dimensional table (see equation 4.4). Table 4.2 shows a part of the transition matrix this implementation uses for German.

Table 4.2: Trigram transition matrix

	l	m	n	o	p	q	r	s
de	.009	.089	.287	.032	.032	.032	.352	.071
di	.032	.032	.014	.032	.032	.032	.040	.008

If we have a table like this for every language we want to be able to identify, we can calculate the probability for a string S to be generated by a particular Markov chain A like this:

$$P(S|A) = \prod_{s_1 \dots s_{k+1} \in S} T(s_1 \dots s_k, S) P(s_{k+1} | s_1 \dots s_k | A) \quad (4.5)$$

where $T(s_1 \dots s_k, S)$ is the number of times the $k + 1$ gram $s_1 \dots s_{k+1}$ occurs in the string S .

To avoid problems of numeric underflow, we compare logarithms of these conditional probabilities. This gives us:

$$\log P(S|A) = \sum_{s_1 \dots s_{k+1} \in S} T(s_1 \dots s_k, S) \log P(s_{k+1} | s_1 \dots s_k | A) \quad (4.6)$$

By computing this value for each of our language models and selecting the largest, we can pick the language model which is most likely to have generated the observed string.

How do we get the transition probabilities (also known as *model parameters*) for a language? We can learn them from sample texts (this is also referred to as training). By counting the number of occurrences of n -grams in the text and then estimating the probabilities for the n -grams we have observed, i.e. $P(s_{k+1} | s_1 \dots s_k | A)$ (from above).

The most obvious estimation method is the maximum likelihood estimator

$$P(s_{k+1} | s_1 \dots s_k | A) = \frac{T(s_1 \dots s_{k+1}, S_A)}{T(s_1 \dots s_k, S_A)} \quad (4.7)$$

where S_A is the training string for language A . This, however, is not completely satisfactory for our application because our training data is relatively limited, so that it will necessarily happen that n -grams which were not in the training data appear in the data to classify. The maximum likelihood estimator gives n -grams which do not appear in the sample a probability of 0, which results in $P(S|A) = 0$. This would result in a completely wrong classification if this n -gram actually appeared in the training data of one language because every string containing this n -gram would then be automatically judged to be from this language.

What we use instead is the Bayesian estimator which minimizes the mean squared error for $P(S|A)$ instead of maximizing the probability. The details are described in [18]; the final form is the expression

$$\hat{p} = \frac{T(s_1 \dots s_{k+1}, S_A) + 1}{T(s_1 \dots s_k, S_A) + m} \quad (4.8)$$

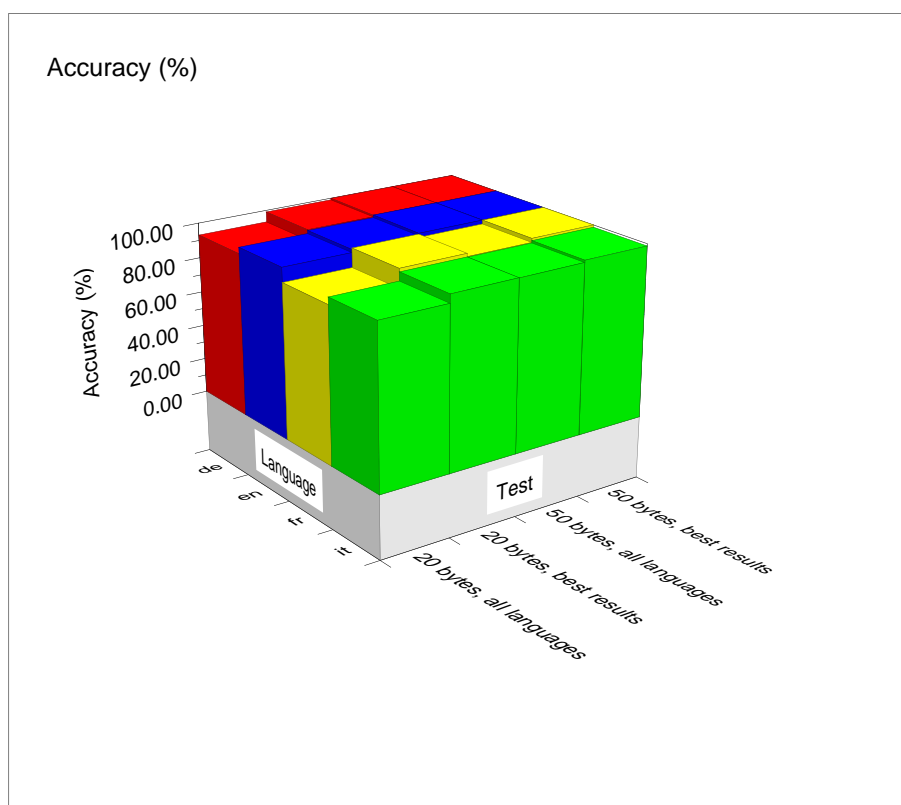
also known as Laplace's sample size correction. This method is used in the implementation described in this thesis.

4.4.3 Performance

In [18] the following conditions are identified as relevant to the performance of a language classifier:

1. Selection of test strings
2. Amount of training data
3. Length of strings to be identified
4. Number of languages to be identified
5. Correlation between domain and language (quality of training data for the intended identification task)

Figure 4.3: Identification Accuracy



We can confirm the relevance of these points. The quality and size of the training data becomes more important with more languages; this problem didn't occur in [18] since selections from a parallel English-Spanish corpus for training and testing were used and English and Spanish were the only languages.

In [18] both the training and testing data were carefully selected and manually checked; there wasn't enough time to do the same for this implementation, so the data was selected in a rather ad-hoc manner from corpora available at CLUE, and from on-line sources like newspapers and press releases. Nevertheless, this implementation compares quite favorably to the original one as well as to the commercial ones. Up to now, training has been performed for

German, English, French, and Italian with 50K of training data for each language.

For 20 byte strings and optimal language pairs, an average accuracy of 98.73% is achieved, which rises to 99.69% for 50 byte strings. For strings longer than 60 bytes no errors could be observed for any of the languages, thus achieving 100% accuracy (see figure 4.3). Classification speed is approximately 340 20-byte strings per seconds when deciding between two languages, but the current implementation was not optimized for speed.

4.4.4 Possible Enhancements

To avoid false classifications due to insufficient training or test data a threshold for accepting a classification could be introduced. No work has been done in this direction yet, since language identification is only one component of IRF/1.

4.5 Indexing

4.5.1 Morphologic Analysis

Two basic indexing methods were implemented: stemming and morphologic analysis. Since the stemmer that I originally wanted to use (Text::German by Norbert Fuhr, available from CPAN) had problems with some of the input, a new stemmer (Text::German::Stem) was written. This stemmer is rather simple but its overall quality is not worse than that of Text::German (although it is different). Text::German has some very limited linguistic knowledge, e.g. about base forms of verbs, but this often leads to spurious analyses, like *nichen* as base form of *nicht* (“not”).

Given a word form, a morphologic analyzer, unlike a stemmer, which only produces a more or less linguistically motivated stem, should return its base form, properties of the base, such as the word class, gender, etc., as well as information on the specific form, such as case, number, tense, mood, or, in the case of a compound, its constituent words.

To do this, two things are needed: a lexicon and grammar rules. The format of the lexicon depends on the grammatical formalism and its specific implementation. In the case of *Malaga*, which is based on the LAMORPH approach to morphologic analysis [30], the lexicon contains only morphemes, which are compiled into an allomorph lexicon by the application of so-called *allomorph rules*. The allomorph lexicon is then used by the *combination rules* to concatenate allomorphs into word forms according to the principles of Left-Associative Grammar [30].

As of this writing, there are three morphologic grammars for *Malaga* (described in section 4.2), and a fourth one (EMM) is being developed in parallel to this thesis:

- DMM for German [46]
- IMM for Italian [64]
- Komoran for Korean [42]
- EMM for English [43]

Due to the lack of a large lexicon, Komoran can currently only be regarded as a toy grammar. IRF/1 was therefore designed to use DMM and IMM, and to easily accommodate EMM and Komor.

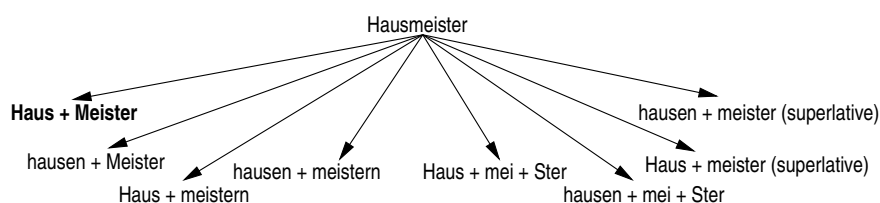
Morphologic Analysis for IR

For a morphologic analyzer to be usable in a real-world application, such as an IR system, its coverage of the language is critical. This means that the lexicon must contain the most frequently used words, either of the language in general, or of the domain it is to be applied to, and that the grammar rules must describe a sufficient part of the morphologic processes.

The trouble with real-world texts is that they tend to contain words that are not even in the most comprehensive dictionary. These are then coupled or modified (or simply misspelled) in creative ways that are not described in grammar books. If automatic morphologic analysis is to be used on such texts, the system is required to handle unknown words robustly. What this exactly means depends on the application: Sometimes a slightly overgenerating grammar might be sufficient, or one might wish that the system constructs a hypothesis, or some sort of automatic lexical acquisition may be desired. In yet other cases it might be considered better if no—possibly wrong—attempt is made at the analysis of unknown word forms.

Another problem is morphologic ambiguity: If there is no syntactic information available, many word forms are morphologically ambiguous, which may lead to the selection of incorrect base forms. For example, the DMM analysis of *Hausmeister* (“janitor”) without morphologic filtering produces eight analyses (see figure 4.4). Using the morphologic filtering feature of DMM this can be reduced to one analysis, and in this case it is the correct one. Morphologic filtering reduces the need for syntactic analysis by applying a number of techniques like the weighting of morphologic processes. It works very well but there has not yet been any formal evaluation. So, it remains to be seen if the results are good enough, or if additional syntactic analysis is necessary; if so, a shallow analysis might be sufficient. Nevertheless, IRF/1 currently relies only on the morphologic filtering feature of DMM for German word forms.

Figure 4.4: Ambiguity in German compounds (correct analysis in boldface)



The IMM grammar, however does not have an integrated disambiguation mechanism. Due to the different morphologic properties of Italian, if one were to be devised, it might have to be along totally different lines. An example for morphologic ambiguity in Italian are the two analyses for *giornalista* (“journalist”) displayed in figure 4.5.

Although it is etymologically correct to derive *giornalista* from *giorno* (“day”), the first analysis is the one that is desired (one might argue, though, that *giornalista* should be derived from *giornale* (“journal”). This type of ambiguity can be resolved by simply counting the number of applied morphologic processes (here in the SIN_CG slot), and selecting the analysis where the fewest

Figure 4.5: IMM analysis for *giornalista*

"giornalista"

1:	<table style="border-collapse: collapse;"> <tr><td style="padding-right: 10px;">CG:</td><td>nominale</td></tr> <tr><td style="padding-right: 10px;">NUM:</td><td>Sg</td></tr> <tr><td style="padding-right: 10px;">SIN_CG:</td><td>{ nominale, flessivo }</td></tr> <tr><td style="padding-right: 10px;">LXM:</td><td>{ "giornalista" }</td></tr> <tr><td style="padding-right: 10px;">ELEMENTS:</td><td>{ "giornalist", "a" }</td></tr> </table>	CG:	nominale	NUM:	Sg	SIN_CG:	{ nominale, flessivo }	LXM:	{ "giornalista" }	ELEMENTS:	{ "giornalist", "a" }
CG:	nominale										
NUM:	Sg										
SIN_CG:	{ nominale, flessivo }										
LXM:	{ "giornalista" }										
ELEMENTS:	{ "giornalist", "a" }										
2:	<table style="border-collapse: collapse;"> <tr><td style="padding-right: 10px;">CG:</td><td>nominale</td></tr> <tr><td style="padding-right: 10px;">NUM:</td><td>Sg</td></tr> <tr><td style="padding-right: 10px;">SIN_CG:</td><td>{ sostantivo , suffisso , suffisso , flessivo }</td></tr> <tr><td style="padding-right: 10px;">LXM:</td><td>{ "giorno" }</td></tr> <tr><td style="padding-right: 10px;">ELEMENTS:</td><td>{ "giorn", "al", "ist", "a" }</td></tr> </table>	CG:	nominale	NUM:	Sg	SIN_CG:	{ sostantivo , suffisso , suffisso , flessivo }	LXM:	{ "giorno" }	ELEMENTS:	{ "giorn", "al", "ist", "a" }
CG:	nominale										
NUM:	Sg										
SIN_CG:	{ sostantivo , suffisso , suffisso , flessivo }										
LXM:	{ "giorno" }										
ELEMENTS:	{ "giorn", "al", "ist", "a" }										

processes were applied. This is admittedly a bit simplistic, but it could be improved after a more thorough analysis of morphologic ambiguity in Italian.

In contrast to IMM, where disambiguation was not considered in the design, EMM is intended to be integrated into a tagger called *Toccata*, which is being designed from the ground up to use several (statistical) disambiguation methods. It remains to be seen how this can be integrated into IRF/1. Until the completion of EMM an implementation of the Porter stemmer² can be used for English.

Figure 4.6: DMM analysis for *Hausmeister*

"Hausmeister"	<table style="border-collapse: collapse;"> <tr><td style="padding-right: 10px;">AnalysisType:</td><td>Parsed</td></tr> <tr><td style="padding-right: 10px;">Surface:</td><td>[WordForm: "hausmeister"]</td></tr> <tr><td style="padding-right: 10px;">Form:</td><td> <table style="border-collapse: collapse;"> <tr><td style="padding-right: 10px;">POS:</td><td>Substantive</td></tr> <tr><td style="padding-right: 10px;">Syn:</td><td> <table style="border-collapse: collapse;"> <tr><td style="padding-right: 10px;">Gender:</td><td>Masculine</td></tr> <tr><td style="padding-right: 10px;">CaseNumber:</td><td>NomSg&DatSg&AccSg&NomPl&GenPl&AccPl</td></tr> </table> </td></tr> <tr><td style="padding-right: 10px;">Weight:</td><td>0.8</td></tr> <tr><td style="padding-right: 10px;">Mor:</td><td> <table style="border-collapse: collapse;"> <tr><td style="padding-right: 10px;">Lexemes:</td><td>{ [Morpheme: "haus"], [Morpheme: "meister"] }</td></tr> <tr><td style="padding-right: 10px;">Allomorph:</td><td>{ "haus", "meister" }</td></tr> </table> </td></tr> <tr><td style="padding-right: 10px;">WordStructure:</td><td>...</td></tr> </table> </td></tr> <tr><td style="padding-right: 10px;">Sem:</td><td>[BaseForm: "hausmeister"]</td></tr> </table>	AnalysisType:	Parsed	Surface:	[WordForm: "hausmeister"]	Form:	<table style="border-collapse: collapse;"> <tr><td style="padding-right: 10px;">POS:</td><td>Substantive</td></tr> <tr><td style="padding-right: 10px;">Syn:</td><td> <table style="border-collapse: collapse;"> <tr><td style="padding-right: 10px;">Gender:</td><td>Masculine</td></tr> <tr><td style="padding-right: 10px;">CaseNumber:</td><td>NomSg&DatSg&AccSg&NomPl&GenPl&AccPl</td></tr> </table> </td></tr> <tr><td style="padding-right: 10px;">Weight:</td><td>0.8</td></tr> <tr><td style="padding-right: 10px;">Mor:</td><td> <table style="border-collapse: collapse;"> <tr><td style="padding-right: 10px;">Lexemes:</td><td>{ [Morpheme: "haus"], [Morpheme: "meister"] }</td></tr> <tr><td style="padding-right: 10px;">Allomorph:</td><td>{ "haus", "meister" }</td></tr> </table> </td></tr> <tr><td style="padding-right: 10px;">WordStructure:</td><td>...</td></tr> </table>	POS:	Substantive	Syn:	<table style="border-collapse: collapse;"> <tr><td style="padding-right: 10px;">Gender:</td><td>Masculine</td></tr> <tr><td style="padding-right: 10px;">CaseNumber:</td><td>NomSg&DatSg&AccSg&NomPl&GenPl&AccPl</td></tr> </table>	Gender:	Masculine	CaseNumber:	NomSg&DatSg&AccSg&NomPl&GenPl&AccPl	Weight:	0.8	Mor:	<table style="border-collapse: collapse;"> <tr><td style="padding-right: 10px;">Lexemes:</td><td>{ [Morpheme: "haus"], [Morpheme: "meister"] }</td></tr> <tr><td style="padding-right: 10px;">Allomorph:</td><td>{ "haus", "meister" }</td></tr> </table>	Lexemes:	{ [Morpheme: "haus"], [Morpheme: "meister"] }	Allomorph:	{ "haus", "meister" }	WordStructure:	...	Sem:	[BaseForm: "hausmeister"]
AnalysisType:	Parsed																										
Surface:	[WordForm: "hausmeister"]																										
Form:	<table style="border-collapse: collapse;"> <tr><td style="padding-right: 10px;">POS:</td><td>Substantive</td></tr> <tr><td style="padding-right: 10px;">Syn:</td><td> <table style="border-collapse: collapse;"> <tr><td style="padding-right: 10px;">Gender:</td><td>Masculine</td></tr> <tr><td style="padding-right: 10px;">CaseNumber:</td><td>NomSg&DatSg&AccSg&NomPl&GenPl&AccPl</td></tr> </table> </td></tr> <tr><td style="padding-right: 10px;">Weight:</td><td>0.8</td></tr> <tr><td style="padding-right: 10px;">Mor:</td><td> <table style="border-collapse: collapse;"> <tr><td style="padding-right: 10px;">Lexemes:</td><td>{ [Morpheme: "haus"], [Morpheme: "meister"] }</td></tr> <tr><td style="padding-right: 10px;">Allomorph:</td><td>{ "haus", "meister" }</td></tr> </table> </td></tr> <tr><td style="padding-right: 10px;">WordStructure:</td><td>...</td></tr> </table>	POS:	Substantive	Syn:	<table style="border-collapse: collapse;"> <tr><td style="padding-right: 10px;">Gender:</td><td>Masculine</td></tr> <tr><td style="padding-right: 10px;">CaseNumber:</td><td>NomSg&DatSg&AccSg&NomPl&GenPl&AccPl</td></tr> </table>	Gender:	Masculine	CaseNumber:	NomSg&DatSg&AccSg&NomPl&GenPl&AccPl	Weight:	0.8	Mor:	<table style="border-collapse: collapse;"> <tr><td style="padding-right: 10px;">Lexemes:</td><td>{ [Morpheme: "haus"], [Morpheme: "meister"] }</td></tr> <tr><td style="padding-right: 10px;">Allomorph:</td><td>{ "haus", "meister" }</td></tr> </table>	Lexemes:	{ [Morpheme: "haus"], [Morpheme: "meister"] }	Allomorph:	{ "haus", "meister" }	WordStructure:	...								
POS:	Substantive																										
Syn:	<table style="border-collapse: collapse;"> <tr><td style="padding-right: 10px;">Gender:</td><td>Masculine</td></tr> <tr><td style="padding-right: 10px;">CaseNumber:</td><td>NomSg&DatSg&AccSg&NomPl&GenPl&AccPl</td></tr> </table>	Gender:	Masculine	CaseNumber:	NomSg&DatSg&AccSg&NomPl&GenPl&AccPl																						
Gender:	Masculine																										
CaseNumber:	NomSg&DatSg&AccSg&NomPl&GenPl&AccPl																										
Weight:	0.8																										
Mor:	<table style="border-collapse: collapse;"> <tr><td style="padding-right: 10px;">Lexemes:</td><td>{ [Morpheme: "haus"], [Morpheme: "meister"] }</td></tr> <tr><td style="padding-right: 10px;">Allomorph:</td><td>{ "haus", "meister" }</td></tr> </table>	Lexemes:	{ [Morpheme: "haus"], [Morpheme: "meister"] }	Allomorph:	{ "haus", "meister" }																						
Lexemes:	{ [Morpheme: "haus"], [Morpheme: "meister"] }																										
Allomorph:	{ "haus", "meister" }																										
WordStructure:	...																										
Sem:	[BaseForm: "hausmeister"]																										
accept																											

Figures 4.6 and 4.7 show sample analyses of the DMM and IMM grammars to show their variation in both structure and contents. Despite attempts at standardization, the formats of the analyses of the grammars unfortunately still differ greatly from one another. What is more, the formats change frequently, which requires a close tracking of the development. This is even worse because the application needs at least some knowledge of the analysis format to work efficiently with it.

² The Perl module Text::English by Ian Phillipps, available from CPAN.

Figure 4.7: IMM analysis for
risurgimento

"risurgimento"

1:	CG:	sostantivo
	GEN:	Mas
	NUM:	Sg
	SIN_CG:	{prefisso, verbo, suffisso, flessivo}
	LXM:	{"ri", "surgere"}
	ELEMENTS:	{"ri", "surg", "iment", "o"}

In IRF/1 several measures help to overcome this problem. There is an object-oriented wrapper (called IRF::Malaga) around the *Malaga* module for Perl. This serves several purposes: It allows to abstract from the inner workings of *Malaga*, especially by allowing to (virtually) use several grammars at the same time. This is accomplished by lazily loading the grammar only when it is needed, and then keeping it active until another grammar is requested. This approach is easier to program (because no inter-process communication (IPC) and no server process are needed) and less memory-intensive than the original *Malaga/MX*³ approach, where all grammars were always active. It has the obvious drawback that there is a delay when the active grammar is switched. This is not a problem when indexing for IRF/1, because it occurs relatively seldom, since (1) the grammar can only be switched between documents, and (2) all documents that are indexed in one batch are usually in the same language. The other advantage of using IRF::Malaga is that it makes changing the interface to the grammars easier, because the details are hidden from the programmer.

Furthermore, all morphology and stemming routines are only accessed through the IRF::Linguistics module, which uses the IRF::Malaga, IRF::Stemmer, and IRF::Stoplist modules and provides a uniform interface regardless of the underlying methods. If, for example, the analysis format of DMM changed again, only changes in this module would become necessary. Similarly, if a new language were to be added, code would only need to be added here.

4.5.2 Storage

General

Three types of data need to be stored in an IR system:

1. the index,
2. the documents, and
3. information about the documents (document metadata)

The central component of any IR system is the index. The index is the basis for the fast retrieval of relevant documents. It is also what differentiates an

³ *Malaga/MX* was a multi-user, multi-grammar *Malaga* server I wrote around July 1996.

IR system from simple search utilities like the *UNIX* `grep` command, which linearly search documents for text strings (although there are experimental IR systems which use `grep` internally, e.g. [2]). The most common data structure for full-text retrieval systems is the inverted file index as described in section 2.3.2. Conceptually, an inverted file index consists of a record for each term that appears in the document collection. A term's record contains pointers to all occurrences of the term in the document collection. The pointers typically reference a document and possibly the location in the document or a weight.

The full text of the indexed documents should also be accessible through the IR system, and must therefore be stored somewhere. In a system which has only abstracts (of paper documents) indexed, these will be normally stored 'within' the system, while the full documents must be physically fetched from their shelves. This type of IR system is becoming less important as the full text of more and more documents is available on-line, making it viable to access the documents themselves directly from the IR system. If the documents are relatively short (e.g. newswire stories) or not intended to be modified (i.e. archived documents), it may still be sensible to store them in the IR system.

However, in a decentralized networked environment, and with documents that are frequently modified at their original locations, it will be more appropriate to simply provide a *storage identifier* which enables the retrieval of the full document. This is essentially how search engines for the WWW work: They provide the user with hyperlinks to documents relevant to their query. The full text of the document (or the images, or whatever media the document may also contain) is not stored in the search engine (except for the index terms), but resides only at its original location, from which the user can retrieve it.⁴

The storage identifiers would be part of the document metadata, which may also include the title, the document language, the document size, the number of terms it contains, its format, or a document representation like an abstract or a thumbnail image.

In IRF/1, the full text of the documents is stored outside of the system, as a file on a local or remote disk. Consequently, only the index and the document metadata has to be stored and handled by IRF/1.

Requirements of IRF/1

The following quotation, describing the reasoning for the reimplementa-tion of the *INQUERY* system also serves quite well to describe the ideas guiding the implementation of IRF/1:

Typically, an IR system that depends on an inverted file index will use custom data management software built from scratch to support the index. An advantage of this approach is that the software is designed specifically to meet the requirements of the particular information retrieval strategy used in the system. A disadvantage is that building such software is difficult and tedious, particularly if it must provide sophisticated features such as concurrency control or recovery. [8]

Building a storage component was not an option because there was not enough time available; consequently, reuse of existing software was the only choice.

⁴ If the environment is as fast-paced and decentralized as the WWW is, it is certain that some "dead links" are among the list of relevant documents when documents disappear between indexing runs. However, it would be even less practical if each search engine would hold its own, most certainly outdated, copy of all documents it has indexed.

However, on what software should the storage component be based? Initially I considered building upon an existing IR system, e.g. the classic *SMART* system [54]. The selection criteria were:

- free availability,
- usable on HP-UX,
- source code in a common language available, and
- modular structure and sufficient documentation.

The first point is obvious—there is simply no budget for a thesis. The second point must be fulfilled so that the system runs on the HP workstations at CLUE and on the author’s personal machine. The last two points are required so that the necessary modifications to accommodate the morphologic analysis could be made with a reasonable effort and within a reasonable time frame. *SMART* would have had the advantage of being a system explicitly designed for conducting IR experiments. It depends, however, on non-standard system functionality only available on Sun systems. This would have made a port much too expensive and impossible to complete in the limited time. Other evaluated IR systems did not meet the requirements much better. This doesn’t mean that they are bad, only that they do not lend themselves to easy modification necessary for an experimental system.

A complete IR system was therefore not available as a basis for IRF/1. Consequently, a storage model for IRF/1 had to be devised which would not need to be implemented from scratch, but which could reuse as much existing software as possible, while still satisfying the requirements of IRF/1.

Data Structures for Text Retrieval

Traditionally, IR systems use specialized data structures such as tries. However, most of these structures are character-based, while the word forms of natural languages are composed of allomorphs. Futrelle and Zhang point out that character-based structures may not be really appropriate to store and search natural-language texts:

One of the problems of the character-based method is that they go to great lengths to make it possible to do rapid searches for patterns such as “c.t” where “.” is a wild card representing any single character. In English, this would return the items, “cat”, “cot” and “cut” which do not form any natural set of interest. Wild cards fail in this example by returning too much and in other cases by returning too little (no simple variant of “mouse” returns “mice” nor does “is” return “are”. [23])

They further observe that regular expressions are typically used to find variant word forms, such as “develop.*” to find the word forms of *develop*, which would become unnecessary if the word forms were morphologically analyzed before stored in an index. Systems using these structures also lack two important features of databases, namely *data independence* and the ability to store structured information (at least not using uniform access methods). Data independence means that applications are independent of the physical data organization. An example from [22] is that when an index is added or removed

for a column in a relational database, no changes are needed in the applications which use this database. The relational data model of tables and relations [14, 13] and the queries used to access the data abstract from the physical storage and access methods actually used.

Off-the-shelf relational database management systems (RDBMS) have further advantages besides using proven technology with a solid theoretical background. They offer flexible structures, so that attributes can be easily added and removed, and they allow the easy addition, deletion, and updating of entries, something which is very difficult with other approaches, where the addition of new documents to a collection usually imply a reindexing of the whole collection:

Rather than update existing inverted lists when adding new documents, many IR systems simply rebuild the inverted file by adding the new documents to the existing collection and indexing the entire collection from scratch. This technique is expensive in terms of time and disk space, resulting in update costs proportional to the size of the total collection after the addition. [7]

RDBMSs usually also support multi-user access and transactions to ensure data integrity; both features are not common for IR systems. Due to the generality of the relational model, when an RDBMS is available, it can be used for many different applications, reducing acquisition and maintenance costs. Finally, although most RDBMS support only subsets of the SQL standard [35], database systems have a relatively high degree of interchangeability thanks to the standardization.

Despite these obvious advantages of RDBMS there is disagreement on whether they can be used to efficiently implement an IR system. The disadvantages named mainly include that RDBMS are usually not optimized for this specific application, which results in storage overhead, processing overhead, and therefore suboptimal performance.

Chris Buckley reports the following about Edward Fox's implementation of *SMART*, which preceded his own:

A large number of people in recent years have suggested putting an information retrieval system on top of one of the existing commercial database systems. There are many advantages to be gained from this, including uniform mechanism for accessing data, concurrency control, and protection features. The disadvantages are the overhead in speed and space. The previous version of *SMART* (Fox's) was based on such a system; the current version is not. [...] The relational system was very efficient for experimental design. It was extremely flexible and it allowed easy, uniform viewing and manipulation of the data input and results.

Unfortunately, these easily designed experiments still needed to be run. They were tremendously slow. Operations which ideally should take about 5 seconds, were taking from 3 minutes to 45 minutes. One experimental run on a medium size database (77 queries and 12000 documents) could take several *days* to complete. The slowness of the system hampered efforts of the experimenters to perform as many experiments as they would like to. [9, p. 34]

Buckley identifies the following causes for the "painful slowness" [9, p. 34f]:

- The free version of *INGRES* was 2–3 times slower than commercial systems of that time
- Data structures commonly used in IR (e.g. trees) were difficult to represent in a relational system
- The overhead for exchanging data between the RDBMS and the application was too high for the massive amounts of data used in IR processes.

The main advantage of custom storage solutions is therefore that they can be optimized for the specific application they were designed for, potentially yielding very high performance. On the other hand, there come a number of disadvantages with this solution. The maintenance cost is high, because the storage software has to be maintained as well as the application, yet another system has to be administered in addition to the RDBMS a site may already be running, since the storage software of the IR system is only usable for this specific purpose. The performance optimization necessarily results in an inflexible structure, often making the later addition, deletion, and updating of entries difficult, if not impossible. Due to its customized nature, the storage manager will not be easily interchangeable with another product. Finally, all the “hard” parts of data management, like multi-user and transaction support, have to be written again.

A completely different approach is not to care at all how the data is actually stored on disk but to use a *persistent object store* or an *object-oriented database management system (OODBMS)* for the on-disk storage of the data structures needed by the IR system. A persistent object store provides a programmer with transparent persistence for their program objects, i.e. they are automatically stored to disk and retrieved when referenced by the program. This completely frees the programmer from handling the disk I/O themselves; the inverted file becomes simply a collection of objects for each term or even for each occurrence. Examples of such systems are described in [23, 15]. However, since there has not yet been done as much theoretical work on OODBMSs as on RDBMSs some issues are not yet fully understood. Consequentially, “[...] it will be some time before we understand fully how to extract performance from OODBs that can match mature relational database systems.” [23]

A very interesting approach has therefore been taken in the conversion of the *INQUERY* system. Originally it used a custom B-tree package for its inverted file indices. As we have already seen, this can be advantageous because the storage system can be specifically adapted to the needs and the particular information retrieval strategy of the IR system. It was recognized, however, that “[...] building such software is difficult and tedious, particularly if it must provide sophisticated features such as concurrency control or recovery.” [8] The developers therefore decided to replace the B-tree package with the *Mneme* persistent object store [49]: “The result is a system that reaps the benefits of using an existing data management facility without sacrificing performance or functionality.” [8] *INQUERY* uses the object store directly by allocating an object (basically a chunk of bits, identified by a number for *Mneme*) for each inverted list that was formerly stored in the B-tree file. The performance of *INQUERY* even improved relative to the original version, which shows that it is not necessary to use custom software to get good performance out of an IR system.

Conclusions

An increasing tendency to use off-the-shelf storage management software for the implementation of IR systems can be noted in the literature (e.g., [49, 8, 7, 15, 1, 16, 22, 17]). Although custom storage management software provides high performance, its inherent inflexibility and high development costs are no longer considered acceptable. However, it is not yet clear what kind of storage management software should be used instead of custom solutions.

For IR/1 two approaches were realized: An object store approach influenced by *INQUERY* and an RDBMS-based approach, which are described in the rest of this chapter.

First Implementation: Object Store

Since no easy-to-use, freely available object store was found, a simple solution based on the *Berkeley DB* package⁵ was developed. It consists of the two modules `IRF::Index` and `IRF::IndexUtils`, and provides a restricted object store which only handles one class of objects right now, namely `IndexEntry` objects. This is further hidden behind the `Index` object.

When requested by an `Index` object, an `IndexEntry` object can return itself in a linearized form, which is then stored in the `DB` database. As one can see, `IRF::Index` provides a very high-level interface, making it easy to use and extend. If desired, the underlying storage manager could be replaced by a commercial product like Object Design, Inc.'s *ObjectStore* without changes to the interface.

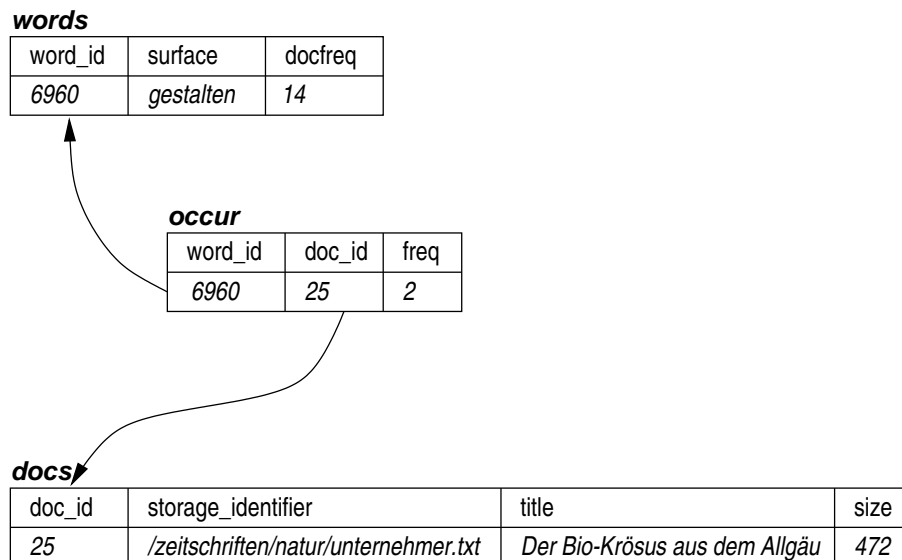
Second Implementation: RDBMS

Because the CLUE Workbench (see section 1.3) is planned to use an RDBMS for uniform storage and access to all types of data, it would be useful if IR/1 could be integrated into the Workbench and use the RDBMS for the storage of its indices. Consequently, a version of IR/1 was prepared which can use any RDBMS supporting SQL. The inverted file index is stored in a database consisting of three tables, as outlined in figure 4.8. Note that this is a simplified figure and that the actual tables may contain more fields.

The `words` table maps every distinct word form which appears in the document collection to a unique identifier (`word_id`). This is standard practice in corpus retrieval applications (see [66]) to avoid the overhead of storing the surface of every word at every occurrence. This table also contains the *document frequency* for each word, i.e. the number of documents in the collection in which a word occurs. The `docs` table contains the document metadata for each document of the collection, like title, storage identifier, language, format, etc. This table is normally accessed by document ID (`doc_id`), but additional criteria, like the language of the document, can be used. The number of distinct terms which occur in the document (also referred to as the length of the document) are also stored in this table (in the `size` column). The `occur` table finally contains the index proper. Each record in this table corresponds to one occurrence of a word (referenced by `word_id`) in a document (referenced by `doc_id`). It also contains the frequency of occurrence of this word in this document (the *within-document frequency*) in the `freq` column. The document

⁵ Available from <http://mongoose.bostic.com/db/>. *Berkeley DB* may be freely redistributed and used under non-commercial conditions.

Figure 4.8: Tables for the Index Database



frequency, the length of each document, and the within-document frequencies are needed for the term weighting necessary for the ranked retrieval model described in section 4.6

Given the tables shown in figure 4.8, the following SQL query would retrieve the titles of all documents containing a word form of *gestalten*:

```

SELECT docs.title
FROM words, docs, occur
WHERE words.surface = 'gestalten' AND
        occur.word_id = words.word_id AND
        docs.doc_id = occur.doc_id
    
```

Currently IRF/1 uses the *mSQL* RDBMS⁶ by Hughes Technologies Pty. Ltd. However, since the Perl DBI module is used, which provides a database-independent interface, any other database system that supports SQL could easily be used instead.

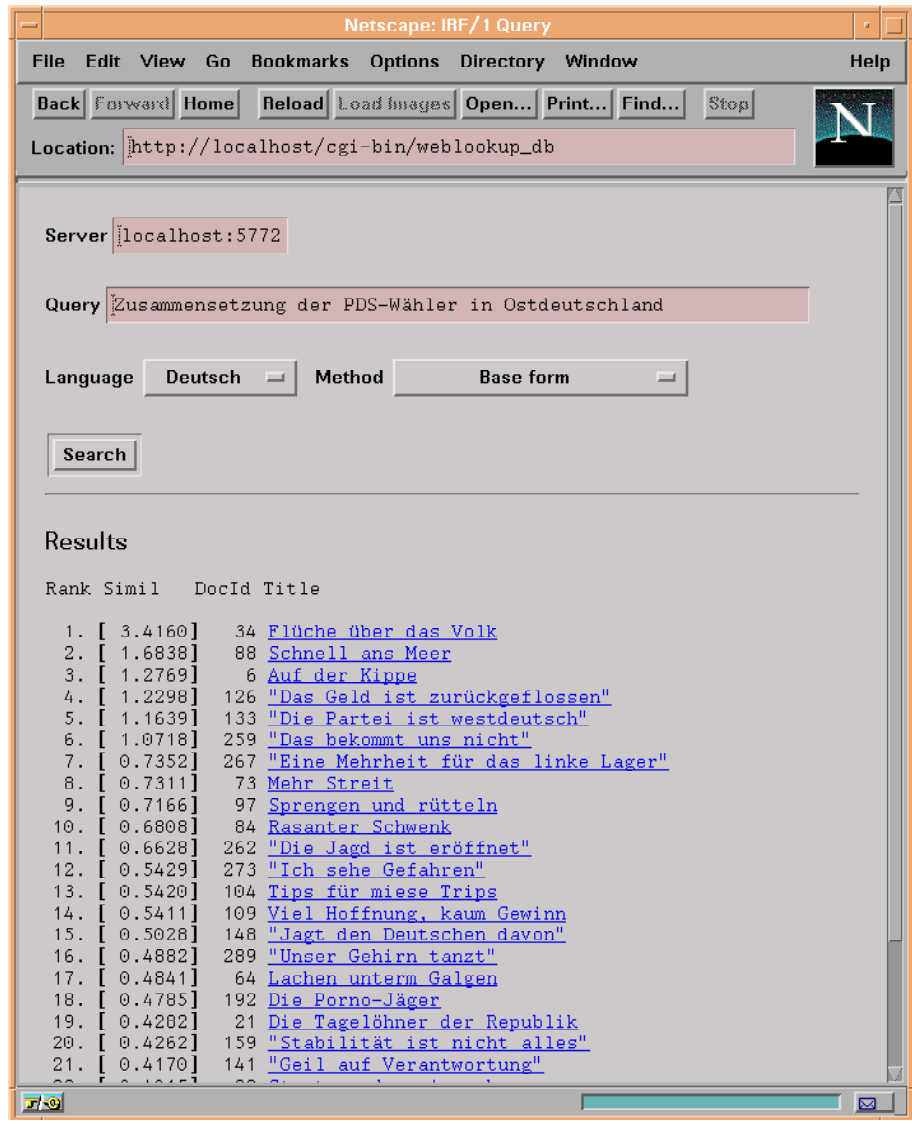
The RDBMS-based architecture proved much more flexible than the object store of the first implementation. The relational system was very efficient for experimental design. I have already cited Buckley's comment on the *INGRES*-based implementation of *SMART* above, and his observation that it "[...] was extremely flexible and it allowed easy uniform viewing and manipulation of the data input and results" [9, p. 34] also applies to the RDBMS-based implementation of IRF/1. Since IRF/1 could not be tested on very large collections anyway (due to the lack of such collections for German) the potential performance penalty was not of concern. For the collections and queries it was tested with, IRF/1 was more than sufficiently fast. Only this approach was therefore made completely operational and used for the evaluation.

4.6 Retrieval

The retrieval of documents is done by the IRF::VSRetrieval module. This module implements a retrieval strategy based on the *vector space model* [54, 26] with *term weighting*. The index is stored in three tables of a relational database, as described above.

⁶ Available from <http://www.hughes.com.au/>. *mSQL* can be used freely for academic purposes.

Figure 4.9: IRF/1 Web Interface



The first retrieval systems were based on Boolean logic (they are therefore often called *Boolean systems*). Queries consisting of a variety of terms are constructed using the Boolean operators AND, OR, and NOT. These operations are implemented by using set intersection, set union, and set difference procedures, respectively. For example, to retrieve documents on information retrieval a query like “information AND retrieval” might be used. The following procedure might be used to identify the corresponding documents:

1. Retrieve the document IDs associated with the term “information” from the index. This set of IDs is set A.
2. Retrieve the document IDs associated with the term “retrieval” from the index. This set of IDs is set B.
3. Determine the intersection of set A and set B, that is, the document IDs which are contained in both set A and set B. This is set C.
4. Retrieve the titles etc. of the documents identified by the IDs in set C and display them for the user.

Although Boolean systems allow to formulate very precise queries, there are a number of problems with them. First, if queries get more complex they become difficult to formulate for untrained users, resulting in poor retrieval results. As Harman notes, “[. . .] end-users are likely to be familiar with the terminology of the data set they are searching, but lack the training and practice necessary to get consistently good results from a Boolean system because of the complex query syntax required by these systems.” [26, p. 363] Second, the retrieval results are not ordered or ranked according to their possible relevance for the query. This means that the set of retrieved documents has to be reduced by intersection until the number of documents is small enough to be inspected by the user. A *ranking approach* seems to be easier to use, especially for the end users mentioned by Harman. This approach allows the user to simply give a natural language sentence or phrase as a query and get a list of documents ranked in the order of their relevance to the query. Harman [26, p. 363f] lists the following advantages of this approach:

- All terms in the query are used for retrieval, with the results being ranked based on cooccurrence of query terms, as modified by statistical term weighting.
- Users are not required to learn a formal query syntax, and some results are provided even if a query term is incorrect (not in the data, misspelled, etc.).
- It also works well for complex queries that may be difficult to express in Boolean logic.

There are different models for building a ranking retrieval system, but the most important is probably the *vector space model*. In the vector space model a document collection is regarded as an n -dimensional space. Each dimension represents one of the terms assigned to the documents of the collection (see figure 4.10). Each document can therefore be represented by a vector (t_1, t_2, \dots, t_n) . In the simplest case t_i is 1 if term i is present, and 0 if it is absent in the document. A query can be represented in the same manner.

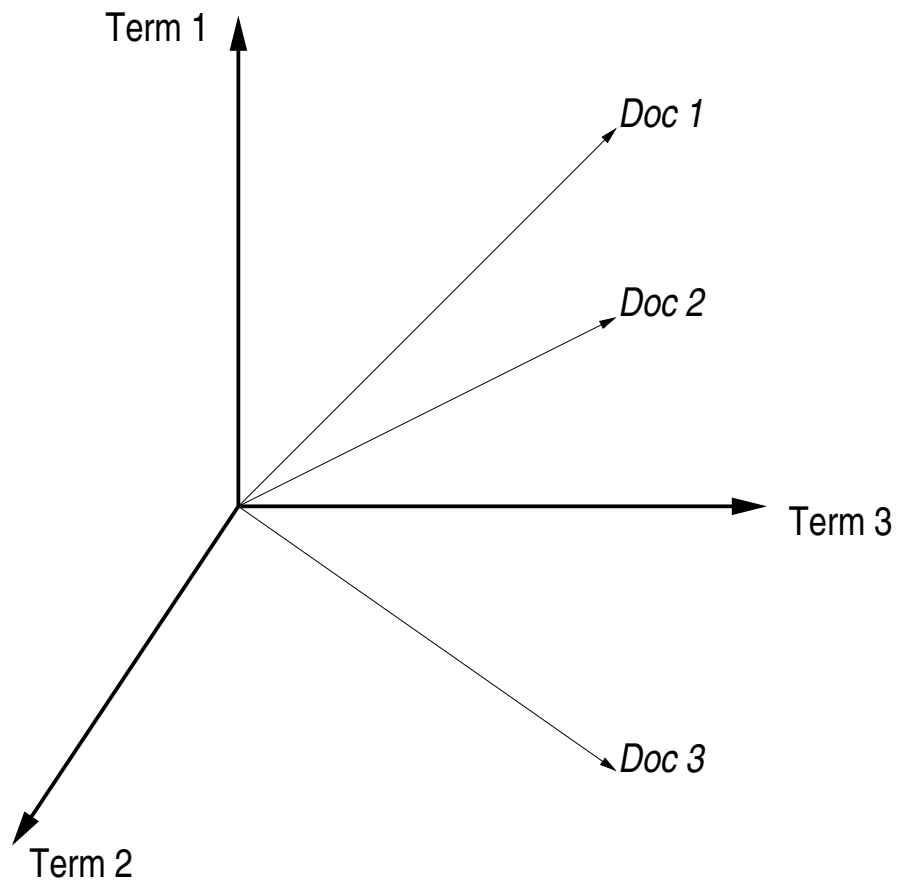
For example, if we have a document collection with seven unique terms, say, *improvement, information, linguistics, overhead, retrieval, storage, systems*, then the query

“linguistics for information storage and retrieval systems”

would be represented by the vector $\vec{q} = (0, 1, 1, 0, 1, 1, 1)$, where an element is 1 if the corresponding term occurs in the query, and 0 if it does not. So, the first 0 indicates that the term *improvement* is not included in the query, whereas the first 1 indicates that the term *information* is present in the query, and so on. The documents in the collection can be represented in the same way, for example:

$$\begin{aligned}\vec{d}_1 &= (1, 1, 0, 1, 0, 1, 0) \\ \vec{d}_2 &= (0, 1, 1, 1, 0, 0, 1) \\ \vec{d}_3 &= (0, 0, 1, 0, 1, 1, 1)\end{aligned}$$

Figure 4.10: Three-term document space



To determine which document matches the query best the angle between the query vector and each of the document vectors is measured. The smaller the angle between two vectors, the more similar they are considered. Usually the cosine of the angle between the two vectors, or just the scalar product of the two vectors is calculated (this will be explained in more detail below). If the scalar (or inner) product between the query vector and each of the document vectors is calculated for the example we arrive at the following ranking:

$$\begin{aligned}\vec{q} \cdot \vec{d}_1 &= (0, 1, 1, 0, 1, 1, 1) \cdot (1, 1, 0, 1, 0, 1, 0) = 2 \\ \vec{q} \cdot \vec{d}_2 &= (0, 1, 1, 0, 1, 1, 1) \cdot (0, 1, 1, 1, 0, 0, 1) = 3 \\ \vec{q} \cdot \vec{d}_3 &= (0, 1, 1, 0, 1, 1, 1) \cdot (0, 0, 1, 0, 1, 1, 1) = 4\end{aligned}$$

\vec{d}_3 matches the query best, while \vec{d}_1 is the worst match. To improve the retrieval quality, it is possible to perform the same operation using weighted vectors, i.e. vectors where each element is not just 0 or 1 but a number which indicates the importance of this particular term. For the example this might look as follows:

$$\begin{aligned}\vec{q} \cdot \vec{d}_1 &= (0, 1, 1, 0, 1, 1, 1) \cdot (1, 2, 0, 1, 0, 2, 0) = 4 \\ \vec{q} \cdot \vec{d}_2 &= (0, 1, 1, 0, 1, 1, 1) \cdot (0, 2, 4, 1, 0, 0, 1) = 7 \\ \vec{q} \cdot \vec{d}_3 &= (0, 1, 1, 0, 1, 1, 1) \cdot (0, 0, 1, 0, 5, 3, 2) = 11\end{aligned}$$

The weight might be the frequency of the term in the document or a different measure, such as the scarcity of a term in the collection, or some user-defined

term weight. Term weighting is known to usually provide “substantial improvement in the ranking” [26, p. 365].

Although it is possible to implement a system in the way described in this example (the “Sequential Access” mode of *SMART* seems to work this way; see [9] for details) it would be a very inefficient system because all document vectors in the collection would have to be compared with the query vector. It is more efficient to base the retrieval system on a standard inverted file enriched with the information needed for term weighting. This is done in the *SIRE* system [54, p. 118ff], [26, p. 384f]. This mode of operation is also available in *SMART* [9, p. 13] under the name of “Inverted Access for Vector Queries.” This method was also chosen for IRF/1. Conceptually it works as follows:

1. For each query term, the document IDs of the documents containing this term, along with the weight of that term, are retrieved from the index. This is equivalent to a Boolean query where all terms are connected with the OR operator. In fact, a Boolean system can be used for this task, and is actually used by IRF/1 since an RDBMS is based on Boolean logic.
2. For each document, a vector of the length of the query can then be constructed. These vectors are compared to the query vector and the rank of each document is calculated.

This method is much more efficient than the one described in the example above: instead of all documents only those which contain at least one term of the query have to be compared, and the vectors have only as many elements as the query has terms, and thus only those terms have to be looked at which have an actual influence on the similarity of a document. Furthermore, if no user-defined weights have been assigned to query terms, the query vector contains only elements which are 1. The query vector can therefore be dropped from any multiplication which is needed to determine a document’s rank.

Because there are a number of different methods available, I have intentionally avoided to specify concrete methods to determine term weights and to calculate the similarity of documents to queries up to this point. These methods will be discussed in the rest of this section. Before this, however, I want to point out that in IRF/1 term weights are not stored in the index but calculated on the fly. This has the advantage that the weighting function can be changed, and that new documents can be added without the need to update all index entries.

The idea of term weighting is based on the observation that not all words in a collection occur with the same frequency, i.e. they are not randomly distributed. As a result of this, classes of words are distinguishable by their occurrence frequencies. Specifically, when the distinct words in a body of text are arranged in decreasing order of their frequency, the occurrence characteristics of the vocabulary can be described by *Zipf’s Law* (see e.g. [54]):

$$\text{Frequency} \cdot \text{Rank} \simeq \text{const.} \quad (4.9)$$

This means that the frequency of a given word multiplied by its rank in the list is approximately equal to the product of frequency and rank for some other word. Furthermore, different frequency classes of words differ in the amount of content they bear. For example, the top-ranked words are almost all function words, which bear no or almost no content. For information retrieval purposes,

a useful index term has two functions: (i) it must be related to the information contained in the document, so that it helps to retrieve the document (called the *recall function* in [54, p. 62]), and (ii) it should help to distinguish a document or a group of documents from the rest of the collection to prevent retrieval of all documents, whether wanted or not (the *precision function*). Words that occur extremely often in a document are likely to be function words, and are thus not representative for the content of a document. Similarly, a term like *computer* is likely to be an unsuitable index term in collection of computer science documents, because it is very likely to occur in every document in the collection—it will not help to distinguish one document from another.

These observations suggest the use relative frequency measures to identify terms which are relatively frequent in a few documents in the collection, but have a relatively low frequency over the whole collection. Several term weighting functions have been devised to emphasize those terms which satisfy the above criteria. The best-known functions are the *inverse document frequency (IDF)*, the *signal-noise ratio*, and the *term discrimination value*.⁷ Of these I will only describe the IDF measure in detail, because it is used in IRF/1; it has proved to yield very good results in numerous experiments, many of which are referenced in [26].

The inverse document frequency measure was originally devised by Sparck Jones in [59]. It is based on the assumption that the importance of a term t is proportional to its frequency in each document d (without stopwords), that is $freq_{td}$, the *within-document frequency*, and inversely proportional to the total number of documents in which it occurs, the so-called *document frequency* df_t . The IDF is traditionally specified as

$$IDF_d = \log_2 \frac{N}{df_t} + 1 \quad (4.10)$$

where N is the number of documents in the collection. The logarithm is used to flatten the curve somewhat for relatively low values of df_t , for which the IDF values would otherwise grow exponentially. In IRF/1 the calculation of the IDF is implemented as

$$IDF_d = \ln \frac{N}{df_t} + 1 \quad (4.11)$$

because the natural logarithm is provided as a built-in function by most programming languages, which is usually more efficient. Since the results of \ln and \log_2 are approximately of the same size, this does not invalidate the function.

To account for the fact that the importance of a term increases with its frequency in a given document, the IDF should be combined with the within-document frequency. It should however be normalized to reduce the effects of high-frequency terms and to compensate for document length. For IRF/1 the following function is used (from [26, p. 375], \log_2 again replaced with \ln):

$$nfreq_{td} = \frac{\ln(freq_{td}) + 1}{\ln length_d} \quad (4.12)$$

⁷ All of these functions are described in [54, p. 59ff]; [26] gives very helpful guidelines for the selection of ranking techniques.

where $length_d$ is the number of unique terms in document d . The normalized within-document frequency is then multiplied with the IDF to produce a composite term weighting function. When a document vector is compared with the query vector this function is used to calculate the weight of each term in the document vector. The two vectors are compared by measuring the angle between the two vectors. This can be done by calculating the cosine of the two vectors:

$$\cos(\vec{q}, \vec{d}) = \frac{\sum_{i=1}^n q_i d_i}{\sqrt{\sum_{i=1}^n q_i^2 \cdot \sum_{i=1}^n d_i^2}} \quad (4.13)$$

where n is the number of components of the vectors, q_i and d_i are the components of \vec{q} and \vec{d} , and q and d are the absolute values or lengths of \vec{q} and \vec{d} , generally:

$$|\vec{a}| = a = \sqrt{\sum_{i=1}^n a_i^2} \quad (4.14)$$

The greater $\cos(\vec{q}, \vec{d})$, the more similar the document is to the query, and the higher is its rank. The retrieved documents are then sorted according to their rank and presented to the user. As an extension *relevance feedback* can be implemented, where the query is reevaluated after the user has reduced or increased the weight of query terms based on the initial search results.

Come, give us a taste of your quality.

—William Shakespeare, *Hamlet*

5.1 Introduction

The evaluation of information retrieval systems has been a concern of IR research from the start. In this chapter I will describe why information retrieval systems have to be evaluated, what should be evaluated, and how it can be done. Finally, I will discuss the evaluation of IRF/1.

The task of an information retrieval system is to help its users to find information (e.g. in the form of text documents) relevant to their information needs. The utility of an information retrieval system for its users therefore depends on how accurately the retrieved information matches the expectations of the users, how quickly the information is delivered, and how easy the system is to use. Or, as Salton and McGill put it: “Few customers will want a system incapable of retrieving what they want and of rejecting what they do not want. Nor will they want a system that is difficult to handle, slow in furnishing responses, or expensive to use.” [54, p. 158] Consequently, information retrieval systems must be evaluated before they are put in operation to avoid costly failures. But evaluation is equally important for research systems, even if they will never be publicly accessible, to see if the new methods which were experimentally introduced are promising enough to be further pursued.

There are basically two types of evaluations: tests for *efficiency*, and tests for *effectiveness*. The effectiveness of an information retrieval system depends on its ability to provide its users with the information they need, while the efficiency is determined by the time and resources needed to perform a specified task.

In most areas of computer science it is possible to consider only efficiency, because the task of the system is exactly specified, and the complete and correct solution is an absolute precondition. Take a C compiler for example: If it claims to be compliant with the ISO C standard [36], it can be checked against the specifications of the standard to see if it works as specified and generates correct code. If it does, it has solved its task correctly, and only the efficiency with which it does its job has to be measured.

While it is relatively easy to generate correct and fast machine code from a correct C program, the ultimate goal of information retrieval—to retrieve all and only the documents from a document collection which are relevant to a query—is currently not possible to achieve. That is, there is no perfect solution, and thus the difference between IR systems is primarily in the quality of the retrieval, whereas it is expected that a C compiler works, so that the question is mainly how efficiently it works.

In this respect the evaluation of information retrieval systems poses challenges similar to the evaluation of natural language processing systems, like morphologic analyzers (see e.g. [32]). In fact, NLP could be considered an IR task: Out of the set of possible analyses, the relevant (i.e., correct) analyses have to be retrieved (see [44] for an example where the quality of morphological analysis and disambiguation is measured in recall and precision). Similar to information retrieval, where—as I will describe below—it is difficult to determine the relevance of retrieved documents, it is often not easy to determine if an analysis of an NLP system is correct.

The quality of an information retrieval system obviously depends on both its efficiency and its effectiveness, and both have to be evaluated. However, the retrieval effectiveness is often of greater importance than the efficiency of the system. This is especially true for experimental systems since they are not yet optimized for speed as production systems would be. Therefore I will be mainly concerned with the evaluation of retrieval effectiveness in this chapter.

5.2 Evaluation Criteria for IR Systems

The next question is then, what to evaluate? As early as 1966, Cleverdon [12] (cited after [53, p. 112]) listed six main measurable quantities:

1. The *coverage* of the collection, that is, the extent to which the system includes relevant matter;
2. the *time lag*, that is, the average interval between the time the search request is made and the time an answer is given;
3. the *form* of presentation of the output;
4. the *effort* involved on the part of the user in obtaining answers to his search requests;
5. the *recall* of the system, that is, the proportion of relevant material actually retrieved in answer to a search request;
6. the *precision* of the system, that is, the proportion of retrieved material that is actually relevant.

The first four of these six criteria are relatively easy to assess. Recall and precision, however, pose a problem because they rely on the notion of *relevance*.

5.3 Relevance

In information retrieval, the notion of *relevance* is usually interpreted as a logical property between two texts, the query and a document. A document is considered relevant if it contains material which is appropriate to the requirements stated in the query. This could be described as an “objective view” [54, p. 163] of relevance. A subjective view of relevance would not only have to consider the content of a document, but it would also have to take into account the knowledge of the user, and what documents they already know about at the time of the query. Although this interpretation of relevance is probably closer to its meaning in everyday language, this is not (yet) possible, since the system cannot know what a user already knows about a topic.

However, even when taking an objective view on relevance, the relevance of a document with regard to some query must be somehow determined. This problem will be discussed after an overview of the standard effectiveness measures.

5.4 Standard Effectiveness Measures

Let E be a set of documents, the *document collection*, and let $A, B \subset E$, where A is the set of relevant documents with respect to some query, and B being the set of actually retrieved documents in response to that query.

These definitions can be conveniently arranged in a ‘contingency table’¹ like table 5.1.

Table 5.1: ‘Contingency table’

	Relevant	Nonrelevant	
Retrieved	$A \cap B$	$\bar{A} \cap B$	B
Not retrieved	$A \cap \bar{B}$	$\bar{A} \cap \bar{B}$	\bar{B}
	A	\bar{A}	

Many effectiveness measures can be derived from this table, the most important being recall and precision. The *precision* P of a retrieval system for some query can be defined as:

$$P = \frac{|A \cap B|}{|B|} \quad (5.1)$$

That is, precision is the proportion of retrieved documents that are relevant. The *recall* R of a retrieval system for some query can then be defined as:

$$R = \frac{|A \cap B|}{|A|} \quad (5.2)$$

¹ This table is commonly being referred to as *contingency table* (see e.g. [53, p. 114], [54, p. 175]) although it does not have anything except the form in common with a contingency table as used for χ^2 tests.

That is, recall is the proportion of relevant documents that are retrieved. Recall and precision are related in such a way that the higher the recall, the lower the precision, and vice versa. The goal is, of course, to maximize both recall and precision at the same time. A third measure, fallout, is less often used but can also be useful:

Assuming that retrieval effectiveness increases with the number of relevant items obtained in answer to a query, and decreases with the number of nonrelevant items retrieved, a measure appears to be needed which reflects the performance for the nonrelevant documents in the same way as recall measures the performance of the relevant. [54, p. 174]

The *fallout* F of a retrieval system for some query can then be defined as:

$$F = \frac{|\bar{A} \cap B|}{|\bar{A}|} \quad (5.3)$$

That is, fallout is the proportion of retrieved documents that are not relevant. Precision, recall, and fallout are further dependent of the *generality factor* G , which is a measure of the density of relevant documents in the collection (i.e. the average number of relevant documents per query). The relationship between these four parameters is:

$$P = \frac{R \cdot G}{(R \cdot G) + F \cdot (1 - G)} \text{ where } G = \frac{|A|}{|E|} \quad (5.4)$$

For every query submitted to a retrieval system, precision and recall can thus be computed. This is the most commonly used measure of retrieval effectiveness. The problem with recall and precision, however, is that they are based on relevance. Van Rijsbergen wrote in 1979 with regard to this problem:

There has been much debate in the past as to whether precision and recall are in fact the appropriate quantities to use as measures of effectiveness. [...] However, all the alternatives still require the determination of relevance in some way. [53, p. 113]

Salton and McGill, four years later, in 1983, pointed out that precision and recall “[...] present the greatest difficulties both conceptually and in practice. An immediate problem in determining the recall and precision is the interpretation of *relevance*.” [54, p. 163] And thirteen years after van Rijsbergen, Frakes still has to write:

Most IR experimentation has focused on retrieval effectiveness—usually based on document relevance judgments. This has been a problem since relevance judgments are subjective and unreliable. [...] The seriousness of the problem is the subject of debate, with many IR researchers arguing that the relevance judgment reliability problem is not sufficient to invalidate the experiments that use relevance judgments. [20, p. 10]

Some judgment of relevance is obviously necessary if one aims at retrieving *relevant* documents, although this is probably the hardest part in the evaluation of IR systems. Thus, to determine the precision and recall values for an

information retrieval system, a *test collection* with associated *test queries* and corresponding relevance assessments for each query is needed. The construction of test collections is the subject of the next section; I will also discuss some approaches which try to reduce the number of manual relevance judgments which need to be made.

5.5 Test Collections

5.5.1 Overview

To be able to reuse relevance judgments, it is desirable to design and build a standard collection of documents and user queries, with associated relevance judgments. This has the further advantage that different information retrieval systems can be compared. [53, p. 113] describes the usual way to construct a test collection:

These questions [the test queries] are usually elicited from bona fide users, that is, users in a particular discipline who have an information need. The relevance assessments are made by a panel of experts in that discipline. So we now have the situation where a number of questions exist for which the 'correct' responses are known. It is a general assumption in the field of IR that should a retrieval strategy fare well under a large number of *experimental* conditions then it is likely to perform well in an *operational* situation where relevance is *not* known in advance.

Table 5.2 lists some of the most widely known and used test collections with their respective properties.

Table 5.2: Test collections (adapted from [27, p. 337]). The abbreviations mean: N Docs, number of documents; Avg T/Doc, average number of terms per document; N Req, number of requests; Avg T/Rq, average number of terms per request; Avg R/Rq, average number of relevant documents per request.

Collection	N Docs	Avg T/Doc*	N Rq	Avg T/Rq*	Avg R/Rq
Older Test Collections					
Cranfield	1,398	53.1	225	9.2	7.2
ADI	82	27.1	35	14.6	9.5
MEDLARS	1,033	51.6	30	10.1	23.2
TIME	423	570	24	16.0	8.7
CACM	3,204	24.5	64	10.8	15.3
CISI	1,460	46.5	112	28.3	49.8
NPL	11,429	20.0	100	7.2	22.4
INSPEC	12,684	32.5	84	15.6	33.0
Newer Test Collections					
OSHUMED	348,566	~250	101	~10	17/19.4 [†]
Cystic Fibrosis	1,239	49.7	100	6.8	6.4–31.9 [‡]
FSupp	11,953	1,823	44	17	35
Fed	410,883	1,235	44	17	56
TREC-1	741,856	444.4	50	83	277
TREC-2	741,856	444.4	50	105	210
TREC-3	741,856	444.4	50	60	196
TREC-4	567,529	842.0	50	10	130

* These numbers should be viewed as approximations for comparison only, as varying the stoplists and varying the stemmers will cause these numbers to change.

[†] There are two levels of relevance for this collection, *definitely relevant* and *possibly relevant*.

[‡] There are 6 levels of relevance for this collection ranging from *specific* to *comprehensive*.

Since a test collection should be as large as possible, and since human experts are needed to judge the relevance of hundreds of documents with respect to hundreds of queries, it is prohibitively expensive (both in time and money) to create test collections in this way for all but the largest organizations, where the necessary funds are available. It is therefore not surprising that other ways to build a test collection or to determine relevance have been searched for.

5.5.2 Reducing the Need for Human Relevance Judgments

Pooling Method

For the massive TREC² collections (see table 5.2), an approach known as the *pooling method* [28] was used. For the pooling method to build a test collection, after documents and queries have been collected, all queries are presented to many different retrieval systems. The best n documents from each query are put into a pool of documents. It is then manually determined for each query which documents in its pool are relevant, and which are not.

This approach has the advantage that the number of relevance judgments that need to be made by domain experts are limited to only those documents which make it into the pool retrieved for a query. The method is based on the assumption that if many different retrieval systems are used, the probability of having almost all relevant documents retrieved will be relatively high, since different systems have different characteristics and thus retrieve different sets of documents for a given query. However, the cost of relevance judgments in this method of building test collections can still be considerable since the pools of documents can be quite large. For the TREC collection, the 100 or 200 best documents from each system are admitted to the pool for each of the 50 queries. All of these documents have to be manually checked, resulting in the average numbers of relevant documents per query listed in table 5.2. Another problem is that many different retrieval systems have to be installed, administered, and used in order to arrive at representative pools. If a retrieval system that is later tested on this collection retrieves different documents, new relevance judgments have to be made, and, if some of the documents are relevant documents which had not been contained in the pool (because all the systems used in building the collection failed to retrieve them), new documents have to be added.

Evaluation Using Seed Documents

Sheridan, Ballerini, and Schäuble [57] of ETH Zurich therefore propose two different methods, which they also use for their *SPIDER* retrieval system.

In the *seeding method*, relatively few documents which were determined to be somehow unique are selected from the collection (the seed documents), and queries are constructed to find these documents. The evaluation is based on some measure of the system's success in finding the seed document for each query. In an application to video retrieval users were instructed to find the seed documents, and the time used to find them was used to compare the different retrieval systems. For a different task, the retrieval of OCR'd library catalog cards, around 2,500 words from a sample of 400 cards were identified which were unique to one card. These words were then used as queries, and the cards in which they occurred were the seed documents. The evaluation criterion in

² Text REtrieval Conference, an open information retrieval competition organized annually since 1993 by the U.S. National Institute of Standards and Technology.

this case was the percentage of seed documents returned in the top three rank positions.

The advantage of this approach is that it requires much less resources than the task of building a test collection using either the traditional method or the pooling method, since only unique seed documents have to be identified, and queries constructed to retrieve these documents. However, a large number of queries should be used to compensate for the fact that for each query only one document (the seed document) is being sought and evaluated. Furthermore, while the seeding method simulates a realistic task (the user has the desired document already in mind, e.g., because they have seen it once and want to find it again), it is different from what is traditionally evaluated, namely a user which has an information need and tries to find information relevant to that need.

The ETH Multilingual Test Collection

For the latter task a multilingual (German and Italian) test collection was constructed at ETH Zurich [57]. This collection is based on news stories by SDA (Schweizerische Depeschenagentur). To reduce the number of relevance judgments needed, 65 completely unpredicted world events were identified to be used as query topics. Since these events were unpredicted, there can be no documents before they occurred. A very strict notion of relevance is used which considers only stories about this particular event relevant (e.g., for the Oklahoma City bombing on April 19, 1995 only stories pertaining to this event are relevant; reports on other terrorist attacks are not relevant). Thus, all documents dated earlier than the event can automatically be considered irrelevant. Because there can also be relevant reports after the event occurred, the news stories of up to three days after the event were also included, while all later documents were discarded. For the documents in these three days after the event manual relevance judgments are needed, which still amounted to about 65,000 judgments in the described case.

This method results in a separate document collection for each query, consisting of:

- the news stories before the event, which are all irrelevant to the query
- the report of the event itself
- the news stories in the 3-days period after the occurrence of the event, which have manually made relevance judgments.

If appropriate material is available, this is a very interesting approach at constructing a test collection because it provides the quality of relevance judgments of the traditional approach (called “the naïve approach” in [57]) with a drastically reduced amount of manual work.

Relative Recall

A different approach was taken for the evaluation of the *IRENA* system [2]. Based on the assumption that all of a collection is in some respects relevant to every query (a single-domain collection about pop music was used), the authors derive a new measure called *relative recall* (*RR*) from equation 5.2:

$$RR = \frac{|A \cap B|}{|E|} \quad (5.5)$$

They describe the properties of RR as follows: “This new measure does not give reliable results for individual queries, but it can be still used for comparing recall between two or more queries.” [2, p. 10] However, it seems that several normalization factors have to be used to approximate the real recall, some of which seem to be rather ad-hoc, so that further investigation will probably be necessary before this measure can be widely employed.

5.5.3 Conclusions

Although the methods that have been devised to make the manual relevance judgment of all documents of a test collection unnecessary seem to arrive at fairly reliable results, the design and construction of new test collections remains difficult. For retrieval experiments on English texts this problem is not very grave due to the relatively large number of freely available standard test collections, which free the evaluators from building their own collections, except perhaps for experiments in special domains or text types.

For languages other than English, however, there are no standard collections yet. TREC now has Spanish and Chinese evaluations (called *tracks*), but researchers developing information retrieval systems for other languages often find it impossible to test their systems on large amounts of data due to a lack of resources.

This is the reason why it was only possible to evaluate IRF/1 with a relatively small collection. The methods and results of the evaluation will be described in the following sections.

5.6 Evaluation of IRF/1

The goal of this thesis is to investigate whether the application of linguistic methods to full-text retrieval will improve the retrieval effectiveness, especially for German, which has a much richer morphology than English, for which most retrieval experiments are made. In this section I will first describe the expected behavior for different indexing methods. Then I will give an overview of the test collection used for the evaluation experiments, which are described afterwards. The results of the evaluation conclude this section.

5.6.1 Expected Behavior

Four different indexing methods were to be evaluated (the abbreviations in parentheses will be used in the following tables and graphics):

1. Stemming (STEM)
2. Base form reduction by morphologic analysis (NODCMP)
3. Splitting of compound words into the base forms of their elements by morphologic analysis (DCMP)
4. Base form reduction and splitting of compound words into the base forms of their elements by morphologic analysis (BOTH)

To illustrate the index terms generated by the different methods, let us look at the phrase *Entscheidung des Bundesverfassungsgerichts* (“decision of the federal constitutional court”). First, *des* (“of the”) is filtered out as a stop word, leaving *Entscheidung* and *Bundesverfassungsgerichts*. Table 5.3 shows the index terms that are generated by the different methods.

Table 5.3: Index terms generated by different methods

	Entscheidung	Bundesverfassungsgerichts
STEM	entscheidung	bundesverfassungsgericht
NODCMP	entscheiden	bundesverfassungsgericht
DCMP	entscheiden	bund, verfassung, gericht
BOTH	entscheiden	bundesverfassungsgericht, bund, verfassung, gericht

This example was chosen to illustrate various morphologic phenomena of German: verb-to-noun derivation by *-ung*, inflection (genitive *-(e)s*), and composition including the insertion of a linking morpheme (called *Fugen-s*) between *Verfassung* and *Gericht* (*Verfassungs* is not a form of the inflectional paradigm of *Verfassung*). It should be noted, however, that there are many more morphologic phenomena.

The expected behavior of stemming (STEM) is to either produce high recall and low precision through *overstemming errors* (i.e., semantically distinct word forms are mapped to one stem) or high precision and low recall through *understemming errors* (i.e., the failure to map semantically related word forms to one stem).

Morphologic analysis without decomposition (NODCMP) was expected to produce consistently better recall and precision than STEM, because over- and understemming errors are less likely to happen, and because changes of the stem can be handled.

Decomposition of compounds (DCMP) was expected to considerably enhance recall, especially in cases where a concept is expressed both by a phrase and a (possibly ad-hoc) compound, as is often done for variation. An example might be *Mehrwertsteuererhöhung* and *Erhöhung der Mehrwertsteuer* (“increase of the sales tax”). Both a drop in precision and a rise in precision were considered possible. The former because the possibly more specific compound is lost, the latter because relevant documents would be ranked higher in cases like the one described above. The BOTH method was thought to perform similar but to prevent the possible drop in precision of the DCMP method by keeping the base form of compounds.

These were the assumptions the experiments described in the following were to verify or falsify.

5.6.2 Test Collection

Because no German test collection was readily available, I had to construct my own. Articles from the German news magazine *DER SPIEGEL* were used for this purpose. The articles are from the winter of 1995/96, when they were collected from the magazine’s Web site for a corpus project at CLUE. The topics range from German and foreign politics over science and economics to culture and movie reviews. The summary or abstract at the beginning of each article had already been identified and marked up for the corpus project. For the test collection the summaries were separated from the body of the article. Only the body text was then used for indexing. The summaries were used to

construct the test queries and were therefore not indexed to avoid “easy hits.” Table 5.4 shows the general statistics of this collection.

Table 5.4: General statistics for the test collection

Total number of documents	299
Total number of word forms (tokens)	402,022
Average document length (in word forms)	1344.56

Since document terms are reduced at indexing time instead of expanding query terms at retrieval time, the term statistics depend on the indexing method used; they are listed in table 5.5.

Table 5.5: Term statistics for the test collection

	STEM	NODCMP	DCMP	BOTH
Total number of index terms	36,360	39,889	22,951	41,303
Total number of term occurrences	145,007	142,150	142,937	174,247
Average number of terms per document	484.97	475.42	478.05	582.77

Ten queries were then devised on the basis of the previously extracted summaries. The relevant documents were selected manually, which is still possible, though tedious, for 299 documents. For larger collections one of the methods described in section 5.5.2 would have to be used.

5.6.3 Measuring Procedures

It was decided to measure the retrieval effectiveness of IRF/1 in recall and precision (described in section 5.4) because these are the measures most widely used. The measuring procedures follow the lines of [54, p. 157ff] and [53, p. 112ff], which are the standard procedures for evaluations of *SMART*, and which are also used at TREC. As already mentioned, the relevance of the documents for each query was determined manually. This is the most accurate method, and recall can be calculated with nearly absolute certainty.

In the form as defined in section 5.4, recall and precision are directly applicable only to Boolean systems, because they are defined in terms of sets. Ranking retrieval systems like IRF/1, however, yield an ordered (ranked) list of documents instead of an unordered set. A pair of recall-precision values can therefore be computed following the retrieval of each document in the ranked order. In most cases, the output of ranking retrieval systems is only *partially ordered*, i.e., several documents may have the same rank, and the order of documents within a rank is random. Special provisions have then to be made concerning the position of relevant documents inside of a rank. For this small test collection, however, the scalar product of the query and document vectors proved to be a better similarity function than the cosine of the vectors (see definitions in section 4.6). This is probably because of the normalizing effect of the cosine, which might be too strong for small collections. The use of the scalar product had the additional advantage for the evaluation that it produces a *simple ordering* of the output (i.e., no two documents share a single rank).³

Table 5.6 might serve as an example. It shows the results of an actual evaluation query. There are four relevant documents for this query, which are marked by “✓”.

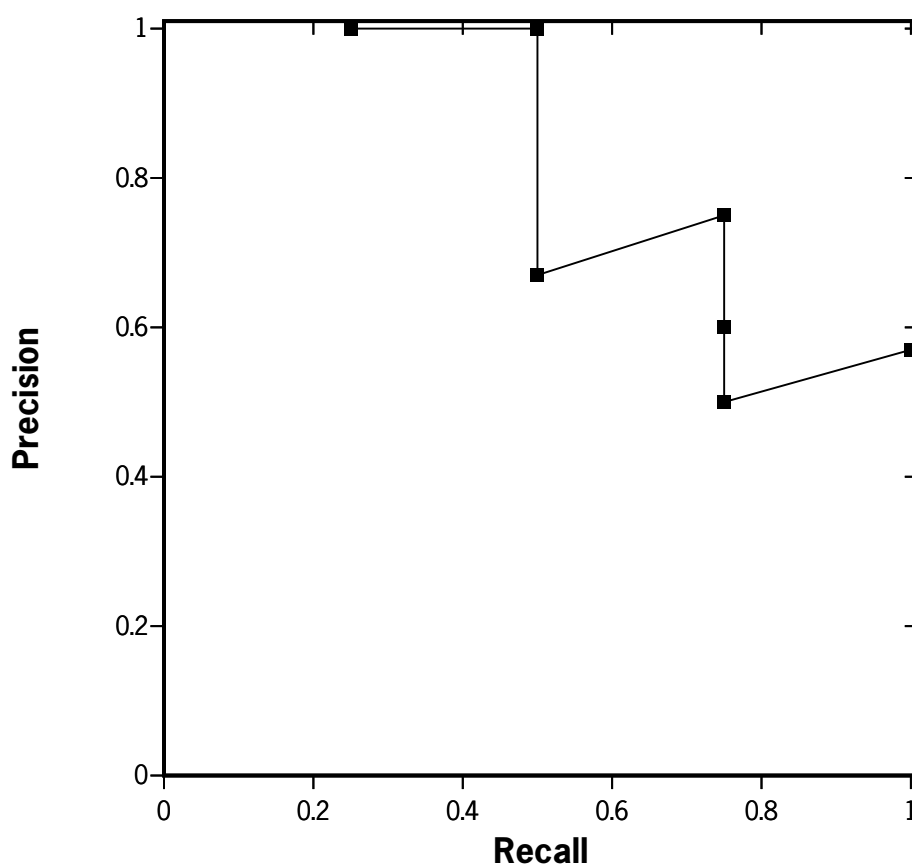
³ Actually, only the probability is very high that under the circumstances of this evaluation no ties occur in the ranking.

Table 5.6: Recall and precision results for a sample query

Rank	Weight	Document ID	Rel	Recall	Precision
1	2.9242	7	✓	0.25	1.00
2	1.0283	179	✓	0.50	1.00
3	0.6818	264		0.50	0.67
4	0.6511	217	✓	0.75	0.75
5	0.6332	188		0.75	0.60
6	0.6287	111		0.75	0.50
7	0.5502	8	✓	1.00	0.57

Given a set of recall-precision pairs, as shown in table 5.6, a *recall-precision graph* can be constructed by plotting the precision against the recall. Figure 5.1 shows the graph corresponding to the recall and precision values from table 5.6. It should be noted that the lines connecting the points only serve to make the plot easier to read. They do not allow the interpolation of values, since only the measured recall and precision values are defined.

Figure 5.1: Recall-precision graph for table 5.6

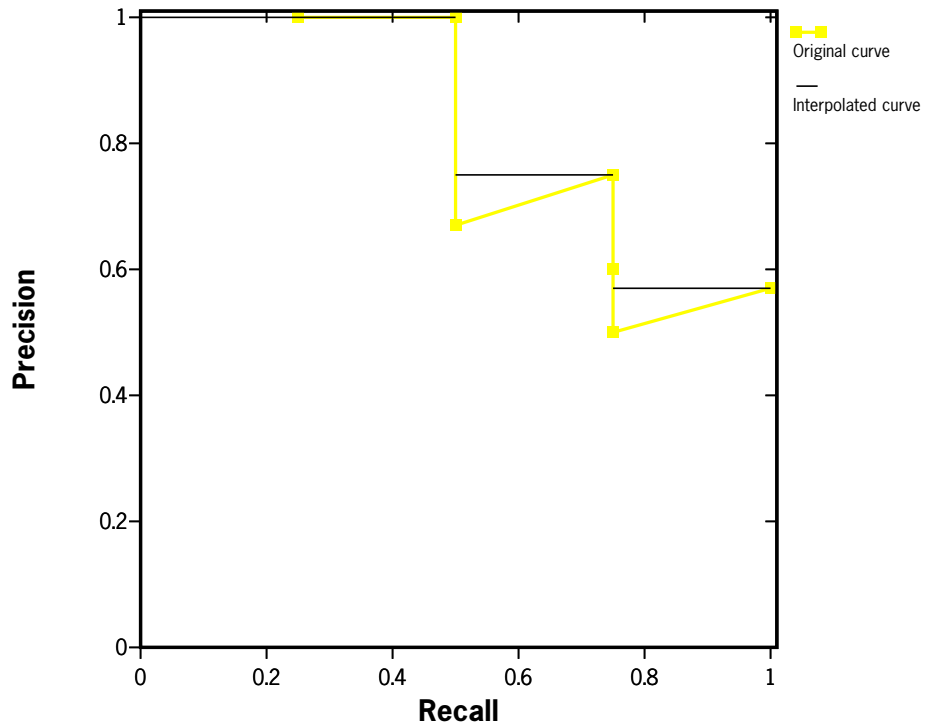


Normally, one is not interested in the performance for a single query but in the average performance of the system. To obtain average performance characteristics, a number of curves such as the one from 5.1, each valid for a single query, have to be processed. This makes it necessary to interpolate the curves so that there is a unique precision value fore each recall point. The standard method for doing this is to proceed as described in [54, p. 167f]: the original graph is replaced by an interpolated version which consists exclusively of horizontal line segments.⁴ This is done by starting at the highest recall value and drawing a horizontal line leftward from each peak point of precision, up

⁴ Not of horizontal and vertical segments, as is often incorrectly stated, see e.g. [39, p. 15].

to a point where a higher precision point is encountered. The resulting curve, representing the best performance a user can achieve, is shown in 5.2.

Figure 5.2: Interpolated recall-precision graph for table 5.6



Given a set of different recall-precision curves such as the one in figure 5.2, each corresponding to a different query, average performance values can now be obtained by computing the arithmetic means of recall and precision over n test queries. Using the symbols from section 5.4, this is:

$$\bar{R} = \frac{1}{n} \cdot \sum_{i=1}^n \frac{|A_i \cap B_i|}{|A_i|} \quad (5.6)$$

$$\bar{P} = \frac{1}{n} \cdot \sum_{i=1}^n \frac{|A_i \cap B_i|}{|B_i|} \quad (5.7)$$

Since the recall and precision values for the individual test queries are unambiguously defined as shown in figure 5.2, the averages as defined in equations 5.6 and 5.7 are also uniquely determined. This enables the calculation of *average precision values at fixed recall levels*, which is the standard way of displaying retrieval performance. A typical interval size for recall values is 0.1. For each test query the precision values are determined for the 11 levels of recall from 0 to 1, and equation 5.7 is used to obtain average precision values over all queries at each of the 11 recall levels. In the resulting curve, the left end corresponds to narrow, i.e. specific, queries, resulting in high precision and low recall. The right end of the curve represents broad, i.e. rather general queries which typically retrieve large numbers of documents with low precision.

Recall-precision curves are used to evaluate the performance of retrieval systems. Typically the recall and precision values for two or more systems, or for one system operating under different conditions are calculated, and superimposed on the same graph to determine which system is superior. The curve

closest to the upper right-hand corner, where recall and precision are maximized, indicates the best performance.

5.6.4 Evaluation Results

Figure 5.3: Average recall-precision graph for IRF/1.

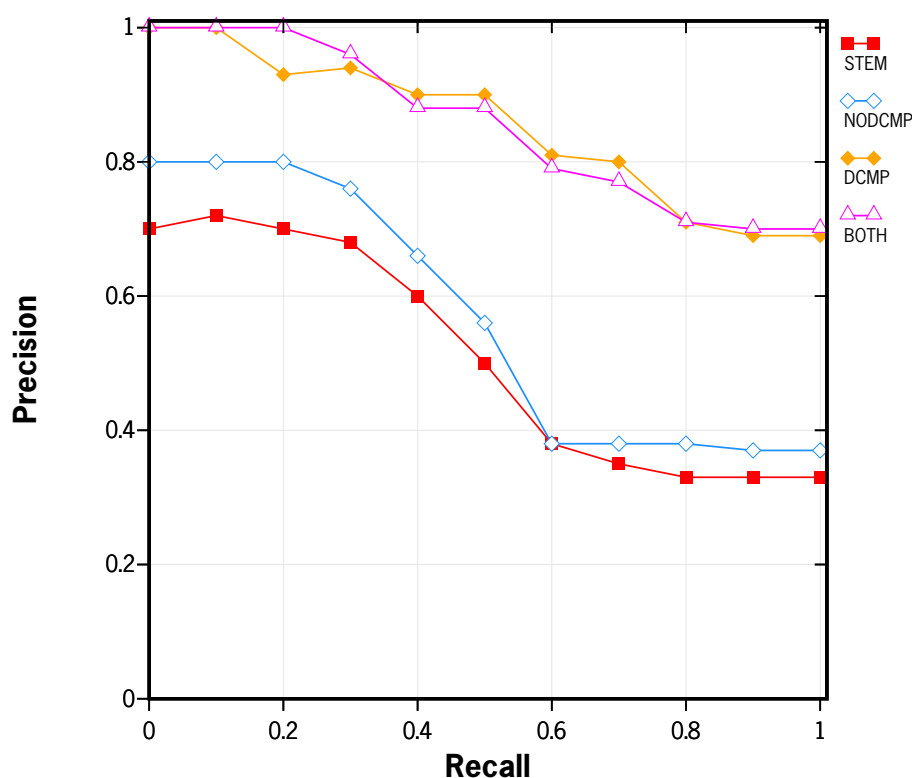


Table 5.7 and figure 5.3 show the results of the evaluation for the four indexing/retrieval methods implemented in IRF/1. How do these results relate to the expected behavior described in section 5.6.1? As expected, stemming (STEM), the traditional method, establishes the base line. The use of morphologic analysis (NODCMP) to provide linguistically motivated base forms instead of the “stems” of a stemmer results in some improvement in precision. This was also expected; one might have expected a more significant improvement, though, especially compared to the relatively crude stemmer which was used. A more sophisticated stemmer might even further reduce the differences. Sophisticated stemmers, however, also come close to full morphologic analysis in their complexity and maintenance costs, if they do not even exceed it due to their rather ad-hoc nature.

The performance improvement gained by the decomposition of compounds (DCMP) in addition to base form reduction is far beyond what was expected. Although the analysis of compounds was expected to be beneficial for the retrieval of German, it proves to be essential to achieve high recall and precision. This observation was also made in the *UPLIFT* project for Dutch, which also has very productive compounding:

The query expansion methods that do more than just conflate the morphological variants and expand the query with new (but semantically related) concepts, namely the methods based on compound analysis and synonym expansion, gave the best Recall results. [39, p. 34]

What is even more interesting is that the expected deterioration of precision due to the splitting of compounds did not occur, and that the addition of the

Table 5.7: Average recall-precision results for IRF/1. The percentages in the Improvement column indicate the improvement in precision from the worst to the best result at the specific recall level.

Recall	Average precision for 10 queries				Improvement (%)
	STEM	NODCMP	DCMP	BOTH	
0.0	0.7000	0.8000	1.0000	1.0000	42.9
0.1	0.7200	0.8000	1.0000	1.0000	38.9
0.2	0.7024	0.8000	0.9333	1.0000	42.4
0.3	0.6774	0.7600	0.9417	0.9600	41.7
0.4	0.6022	0.6600	0.9050	0.8767	50.3
0.5	0.5022	0.5600	0.9050	0.8767	80.2
0.6	0.3776	0.3822	0.8077	0.7878	114.0
0.7	0.3500	0.3808	0.8035	0.7694	129.6
0.8	0.3321	0.3780	0.7096	0.7126	114.6
0.9	0.3321	0.3677	0.6884	0.6992	110.5
1.0	0.3321	0.3677	0.6884	0.6992	110.5

unsplit base forms of compounds (BOTH) did not result in any further increase in precision.

Due to the small scale of the experiments, the individual numbers should not be overestimated, but there is certainly a recognizable trend indicating that the use of morphologic analysis for German text retrieval—and especially the analysis and decomposition of compounds—yields significantly better retrieval performance than traditional stemming. The higher initial development effort necessary for morphologic analysis seems to eventually pay off when the more complex task of analyzing compound words is required.

The question posed in the beginning was whether linguistic methods can help to improve performance and user-friendliness of full-text retrieval systems. I have shown in chapter 3 that currently only improvements on the level of morphology can be considered. Stemming is the standard method here. While its usefulness is still disputed for English, experiments for other languages have shown it to be useful for more inflected languages. In the *SPIDER* and *UP-LIFT* projects, dictionary-based stemming and decomposition of compounds were found useful for Dutch and German. The results of chapter 5 confirm this observation for German. They clearly indicate that linguistic knowledge on the morphologic level can improve retrieval performance for German, especially if compound word forms are decomposed into their constituents. This results in high retrieval precision, even at high recall levels. Because the user does not have to care how to formulate their query this is definitely also an advance in user-friendliness.

To the user, compounds may often intuitively seem to better express a concept, but they are difficult to handle without linguistic knowledge. Full-fledged natural language processing on a linguistic basis is often regarded as “overkill,” too expensive, or too slow. The evaluation results, however, suggest otherwise. As stated in chapter 5, the higher initial cost for laying down a solid, linguistically motivated foundation, is eventually compensated by more comprehensive analyses and better scalability. While an inflectional stemmer might be developed faster and more easily than full morphologic analysis, it cannot simply be extended to handle derivation and compounding. Any attempt to do so without a linguistic basis will, with a very high probability, result in an unmaintainable “kludge,” necessitating a complete rewrite. With full morphologic analysis, however, the analysis of compounds is obtained at no additional cost. Furthermore, the latest version of *Malaga* brings a dramatic increase in speed and, at the same time, lower memory requirements, making it more attractive than ever to embed full linguistic NLP into applications. *DMM*, the German morphology grammar, also proved that it is robust enough to be suited for the analysis of real-world texts. Although no formal evaluation of *DMM* was conducted, the recently improved disambiguation routines and the new hypothesis module for unknown word forms seem to be very effective.

One can therefore conclude that the evaluation results for IRF/1 are very promising. Further work should aim at validating the results of this thesis with larger test collections. Since IRF/1 was designed to provide linguistically supported indexing and retrieval for any language for which a *Malaga* grammar exists, it would also be of great interest to investigate how linguistics might improve retrieval performance for other languages, or to research into cross-language retrieval. When available, the use of syntactic and semantic analysis could also be evaluated. Thanks to its modular architecture and the flexibility of the vector space model, IRF/1 could also easily be extended with relevance feedback and term reweighting methods to further improve retrieval performance.

Bibliography

- [1] Aoki, Paul M. 1991. "Implementation of Extended Indexes in POSTGRES." *SIGIR Forum*. 25(1): 2–9.
- [2] Arampatzis, Avgerinos, Theofilos Tsoiris, and C.H.A. Koster. n.d. *IRENA: Information Retrieval Engine based on Natural language Analysis*. Technical Report. CTI, University of Patras/CSI, University of Nijmegen, Patras/Nijmegen.
- [3] Baayen, R. H., R. Piepenbrock, and H. van Rijn, eds. 1993. *The CELEX Lexical Database*. CD-ROM. Linguistic Data Consortium, Philadelphia, PA.
- [4] Berghel, Hal. 1997. "Cyberspace 2000: Dealing with Information Overload." *Communications of the ACM*. 40(2): 19–24.
- [5] Beutel, Björn. 1997. *Malaga 4.0*. Manual. Abteilung für Computerlinguistik, Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen.
- [6] Boguraev, Branimir, and James Pustejovsky. 1996. "Issues in Text-based Lexicon Acquisition." In *Corpus Processing for Lexical Acquisition*, edited by B. Boguraev and J. Pustejovsky. Cambridge, MA: MIT Press.
- [7] Brown, Eric W., James P. Callan, and W. Bruce Croft. 1994. "Fast Incremental Indexing for Full-Text Information Retrieval." *Proceedings of the 20th Conference on Very Large Data Bases (Santiago, Chile, 1994)*: 192–202.
- [8] Brown, Eric W., James P. Callan, W. Bruce Croft, and J. Eliot B. Moss. 1994. "Supporting Full-Text Information Retrieval with a Persistent Object Store." *Proceedings of the 4th International Conference on Extending Database Technology*: 365–378.
- [9] Buckley, Chris. 1985. *Implementation of the SMART Information Retrieval System*. Technical Report TR85-686. Cornell University, Ithaca, NY.

- [10] Choueka, Yaacov, and A. Zampoli. 1992. *Responso: An Operational Full-Text Retrieval System with Linguistic Components for Large Corpora: Computational Lexicology and Lexicography: a Volume in Honor of B. Quemada*. Pisa: Giardini Press.
- [11] Church, Kenneth Ward. 1995. "One Term or Two?" *Proceedings of the 18th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval 1995*: 310–318.
- [12] Cleverdon, C. W., J. Mills, and E. M. Keen. 1966. *Factors Determining the Performance of Indexing Systems: Vol. 1—Design*. Technical Report. Aslib Cranfield Research Project, Cranfield.
- [13] Codd, Edgar F. 1990. *Relational Model for Data Management: Version 2*. Reading, MA: Addison-Wesley.
- [14] Codd, Edgar F. . "A Relational Model for Large Shared Data Banks." *Communications of the ACM*. 13(6).
- [15] Cutting, Doug, Jan Pedersen, and Per-Kristian Halvorsen. 1991. "An Object-Oriented Architecture for Text Retrieval." *Proceedings of RIAO'91*.
- [16] DeFazio, Samuel, Amjad Daoud, Lisa Ann Smith, Jagannathan Srinivasan, Bruce Croft, and Jamie Callan. 1995. "Integrating IR and RDBMS Using Cooperative Indexing." *Proceedings of the Eighteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Seattle, WA.*: 84–92.
- [17] DeBloch, Stefan, and Nelson Mendonça Mattos. 1997. "Integrating SQL Databases with Content-Specific Search Engines." *VLDB'97, Proceedings of 23th International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*: 528–537.
- [18] Dunning, Ted. 1994. *Statistical Identification of Language*. Technical Report CRL MCCS-94-273. Computing Research Lab, New Mexico State University, Las Cruces, NM.
- [19] Evans, David A., and Chengxiang Zhai. 1996. "Noun-Phrase Analysis in Unrestricted Text for Information Retrieval." *34th Annual Meeting of the Association for Computational Linguistics—Proceedings of the Conference*: 17–24.
- [20] Frakes, William B. 1992. "Introduction to Information Storage and Retrieval Systems." In *Information Retrieval*, edited by W. B. Frakes and R. Baeza-Yates. Englewood Cliffs, NJ: Prentice Hall: 1–12.
- [21] Frakes, William B. 1992. "Stemming Algorithms." In *Information Retrieval*, edited by W. B. Frakes, and R. Baeza-Yates. Englewood Cliffs, NJ: Prentice Hall: 131–160.
- [22] Fuhr, Norbert. 1996. "Object-oriented and Database Concepts for the Design of Networked Information Retrieval Systems." *Proceedings of the Fifth International Conference on Information and Knowledge Management (CIKM 1996)*: 164–172.

- [23] Futrelle, Robert P., and Xiaolan Zhang. 1994. "Large-Scale Persistent Object Systems for Corpus Linguistics and Information Retrieval." *Proceedings of the First Annual Conference on the Theory and Practice of Digital Libraries*.
- [24] Goscinny, René, and Albert Uderzo. 1966. *Astérix chez les Bretons*. Paris: Dargaud.
- [25] Harman, Donna. 1991. "How Effective is Suffixing?" *Journal of the American Society for Information Science*. 42(1): 7–15.
- [26] Harman, Donna. 1992. "Ranking algorithms." In *Information Retrieval*, edited by W. B. Frakes, and R. Baeza-Yates. Englewood Cliffs, NJ: Prentice Hall: 363–392.
- [27] Harman, Donna. 1996. "Panel: Building and Using Test Collections." *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval SIGIR 96*: 335–337.
- [28] Harman, Donna, ed. 1993. *The First Text REtrieval Conference (TREC-1)*. Special Publication 500-207. National Institute of Standards and Technology, Gaithersburg, MD.
- [29] Harman, Donna, Ricardo Baeza-Yates, Edward Fox, and W. Lee. 1992. "Inverted Files." In *Information Retrieval*, edited by W. B. Frakes, and R. Baeza-Yates. Englewood Cliffs, NJ: Prentice Hall: 28–43.
- [30] Hausser, Roland R. 1989. *Computation of Language: An Essay on Syntax, Semantics and Pragmatics in Natural Man Machine Communication*. Berlin: Springer Verlag.
- [31] Hausser, Roland R. 1992. "Complexity in Left-Associative Grammar." *Theoretical Computer Science*. 106(2): 283–308.
- [32] Hausser, Roland R., ed. 1995. *Linguistische Verifikation: Dokumentation zur Ersten Morpholympics 1994*. Tübingen: Niemeyer.
- [33] ISO 8859-1:1987. *Information processing — 8-bit single-byte coded graphic character sets — Part 1: Latin alphabet No. 1*.
- [34] ISO/IEC 8601:1988. *Data elements and interchange formats — Information exchange — Representation of dates and times*.
- [35] ISO/IEC 9075:1992. *Information technology — Database languages — SQL*.
- [36] ISO/IEC 9899:1990. *Programming languages — C*.
- [37] Knorr, Oliver. 1997. *Entwicklung einer Java-Schnittstelle für die linguistische Arbeitsumgebung Malaga*. Pre-master's thesis, Institut für Mathematische Maschinen und Datenverarbeitung, Friedrich-Alexander-Universität Erlangen-Nürnberg.
- [38] Kraaij, Wessel, and Renée Pohlmann. 1996. "Viewing Stemming as Recall Enhancement." *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval SIGIR 96*: 40–48.

- [39] Kraaij, Wessel, and Renée Pohlmann. n.d. *Using Linguistic Knowledge in Information Retrieval*. Technical Report. Utrecht University, Utrecht.
- [40] Krenn, Brigitte, and Christer Samuelsson. 1997. *The Linguist's Guide to Statistics*. Online document: http://www.coli.uni-sb.de/~christer/stat_cl.ps. Lehrstuhl für Computerlinguistik, Universität des Saarlandes, Saarbrücken.
- [41] Krovetz, Robert. 1993. "Viewing Morphology as an Inference Process." *Proceedings of the Sixteenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval 1993*: 191–202.
- [42] Lee, Kiyong. 1995. "Recursion Problems in Concatenation: A Case of Korean Morphology." *Proceedings of PACLIC 10, the 10th Pacific-Asian Conference on Language, Information and Computation*.
- [43] Leidner, Jochen. 1998. *Linksassoziative morphologische Analyse des Englischen mit stochastischer Disambiguierung*. Master's thesis, Abteilung für Computerlinguistik, Friedrich-Alexander-Universität Erlangen-Nürnberg.
- [44] Levinger, Moshe, Uzzi Ornan, and Alon Itai. 1995. "Learning Morpho-Lexical Probabilities from an Untagged Corpus with an Application to Hebrew." *Computational Linguistics*. 21(3): 383–404.
- [45] Lewis, David D., and Karen Sparck Jones. 1996. "Natural language processing for information retrieval." *Communications of the ACM*. 39(1): 92–101.
- [46] Lorenz, Oliver. 1996. *Automatische Wortformererkennung für das Deutsche im Rahmen von Malaga*. Master's thesis, Abteilung für Computerlinguistik, Friedrich-Alexander-Universität Erlangen-Nürnberg.
- [47] Lovins, Janet B. 1968. "Development of a Stemming Algorithm." *Mechanical Translation and Computational Linguistics*. 11: 22–31.
- [48] Mauldin, Michael L. 1991. *Conceptual Information Retrieval. A Case Study in Adaptive Partial Parsing*. Boston: Kluwer.
- [49] Moss, J. Eliot B. 1990. "Design of the Mneme Persistent Object Store." *ACM Trans. on Information Systems*. 8(2): 103–139.
- [50] Novell Advanced Technology Division. 1997. *Novell ATD Collexion Language Identifier*. Online document: <http://www.novell.com/atd/colx/alt/lanf/lanrec.html>. Novell ATD, San Jose, CA.
- [51] Popovič, Mirko, and Peter Willet. 1992. "The Effectiveness of Stemming for Natural-Language Access to Slovene Textual Data." *Journal of the American Society for Information Science*. 43(5): 384–390.
- [52] Porter, M. F. 1980. "An Algorithm for Suffix Stripping." *Program*. 14(3): 130–137.
- [53] van Rijsbergen, C. J. 1979. *Information Retrieval*. London: Butterworths.
- [54] Salton, Gerard, and Michael McGill. 1983. *Introduction to Modern Information Retrieval*. New York: McGraw-Hill.

- [55] Schulze, Markus. Forthcoming. "Morphologie, Syntax und Semantik im Rahmen der linksassoziativen Grammatik." *Proceedings der GLDV-Jahrestagung 1997, Leipzig*.
- [56] Sheridan, Páraic, and Jean Paul Ballerini. 1996. "Experiments in Multilingual Information Retrieval using the SPIDER System." *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval SIGIR 96*: 58–65.
- [57] Sheridan, Páraic, Jean Paul Ballerini, and Peter Schäuble. 1996. "Building a Large Multilingual Test Collection from Comparable News Documents." *Proceedings of Workshop on Cross-linguistic Information Retrieval, held in conjunction with the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 96*.
- [58] Smeaton, Alan F. 1996. "An Overview of Information Retrieval." Agosti, Maristella, and Alan Smeaton. 1996. *Information Retrieval and Hypertext*. Boston, MA: Kluwer: 3–25.
- [59] Sparck Jones, Karen. 1972. "A Statistical Interpretation of Term Specificity and Its Application in Retrieval." *Journal of Documentation*. 28(1): 11–20.
- [60] Sparck Jones, Karen. 1997. "The way forward in information retrieval." *elsnews*. 6(3): 12–13.
- [61] Srinivasa, Padmini. 1992. "Thesaurus Construction." In *Information Retrieval*, edited by W. B. Frakes, and R. Baeza-Yates. Englewood Cliffs, NJ: Prentice Hall: 161–218.
- [62] Sun Labs International Linguistic Application Group. 1997. *Language Identification Demo*. Online document: <http://www.sunlabs.com/research/ila/demo/>. Sun Microsystems Laboratories, Chelmsford, MA.
- [63] Wall, Larry, Tom Christiansen, and Randal L. Schwartz. 1996. *Programming Perl*. Sebastopol, CA: O'Reilly.
- [64] Wetzel, Christian. 1996. *Erstellung einer Morphologie für Italienisch in Malaga*. Pre-master's thesis, Institut für Mathematische Maschinen und Datenverarbeitung, Friedrich-Alexander-Universität Erlangen-Nürnberg.
- [65] Xerox Research Centre Europe. 1997. *RXRC Language Identifier*. Online document: <http://www.xrce.xerox.com/research/mltt/Tools/guesser.html>. Xerox Research Centre Europe, Grenoble.
- [66] Zierl, Marco. 1997. *Entwicklung und Implementierung eines Datenbanksystems zur Speicherung und Verarbeitung von Textkorpora*. Master's thesis, Abteilung für Computerlinguistik, Friedrich-Alexander-Universität Erlangen-Nürnberg.