Title	Indexing multilingual information on the web
Author(s)	Yip, Chi Lap; Kao, Ben
Citation	Proceedings - Ieee Computer Society's International Computer Software And Applications Conference, 1998, p. 576-581
Issued Date	1998
URL	http://hdl.handle.net/10722/45605
Rights	Creative Commons: Attribution 3.0 Hong Kong License

Indexing multilingual information on the web

葉志立 YIP Chi Lap, Ben KAO
Department of Computer Science, The University of Hong Kong
{clyip,kao}@cs.hku.hk

Abstract

The web connects people speaking more than twenty languages in more than one hundred countries. Search engines, which provide users starting points to navigate and retrieve resources on the web, should thus be able to handle documents in many languages. Moreover, with information being added and changed every minute on the web, search engines should discover new index terms time-efficiently. This paper introduces an abstraction of viewing multilingual documents and a statistical analysis method so that search engines can index multilingual documents in a generic, efficient, and effective manner with minimal requirements of language-specific information.

1. Introduction

The "Information Superhighway" (Internet) enables a computer user to be connected to virtually endless number of sites on the global network. The World-Wide-Web uses the Internet to transmit hypermedia documents between computer users located around the world. According to the Internet Society [1], there are more than 150 countries with Internet access. The Internet community is really a multi-racial one with netizens speaking more than twenty different languages.

In order to fully utilize the power of the Web as a gigantic information source, it is essential to have a starting point, and that is what search engines provide. Search engines work by traversing the Web via the hyperlinks that connect the Web pages, performing text analysis on the pages they encounter, and indexing the pages based on the key terms (or index terms) they contain. A user seeking information from the Web would formulate his information goal with a few index terms composing a query. A search engine, on receiving a query, would match the query against its document index. One can thus see that how successful a search engine is depends on its ability of extracting index terms from the documents. Traditional search engines, such as WebCrawler, have little problem in index term extraction on documents that are written in Latin-based languages such as English. They assume that potential index terms are character sequences delimited by punctuations and whitespaces. A stoplist, containing words such as prepositions and connectives, is used to remove those terms that are not effective in differentiating the documents. However, extending search engines to cover documents written in other languages, such as Chinese and Japanese, is not trivial. For example, there are no hard and fast rules of segmenting a piece of Chinese document into meaningful terms without using substantial language-specific information. Successful Chinese segmentation algorithms often rely heavily on a "good" dictionary — an up-to-date set of terms. Unfortunately, such dictionaries are hard to come by. This is especially true in a dynamic environment such as the Web on which new index terms (such as people's names) are constantly created.

Dictionaries not only help segmentation and retrieval of documents in Oriental languages, they are also useful for phrase extraction in Latin-based documents. Thus, the problem of maintaining a *timely* dictionary becomes important. Given the myriad of documents that a search engine processes, manual update of dictionaries can never meet the timeliness requirement. Automatic index term extraction and dictionary maintenance are thus essential.

The problems of term extraction and dictionary maintenance are not new. There are quite a number of related studies on, say, the Chinese language. However, algorithms for term extraction and dictionary maintenance usually require much language-specific information, notably a language and grammar model, and some language-specific heuristics [4]. Implementing these algorithms in a Web search engine is not very appropriate for two reasons. First, computational linguistics approaches are very computationally expensive, and thus may not be fast enough to handle the large numbers of documents search engines process. Second, if the indexing engine is to cover the Web world-wide, we need to implement a language model for each possible language — a daunting task given that each model is a complex system in its own right. Moreover, not all languages are well studied and have their linguistics models available.

The goal of this paper is to design a multilingual index-

ing engine that is suitable for use in a Web environment. Such an indexing engine must satisfy four requirements:

1 it uses **minimal language-specific information**, so that the implementer (who only speaks in one or a few languages) does not need to study and collect the different features of all the languages;
2 it is **generic**, so that extending the index engine to cover a new language is straightforward;
3 it is **time-efficient**, so that the index engine can handle large numbers of documents on the Web;
4 it is **effective**, so that the index terms extracted faithfully represent the concepts or topics relevant to the documents.

Incidentally, the problem of handling multilingual information and finding index terms not only appears in search engines but also in other areas of information retrieval, such as digital libraries. Being a networked information system with contributions worldwide, a digital library has to handle documents in multiple languages. Since cataloging is a very important aspect in libraries traditional or digital, successful handling of multilingual documents is very essential for digital libraries as well.

To design an indexing system that can achieve the four requirements, we first identify four issues and difficulties in handling multilingual information in Section 2. They are document language identification, document segmentation, dictionary maintenance and new index term identification. After establishing in Section 3 that we only need to determine the coded character set of a document rather than its language to do indexing, a view of documents using an abstraction is introduced in Section 4. Then, a purely statistical, language-independent method using suffix trees for finding new index terms is described in Section 5. Following that, examples of applications of the abstraction and analysis stages will be introduced in Section 6. As a proof of concept, we have done some experiments and implemented a search engine, which are reported in Section 7. Finally, we summarize our study and describe our related future work in Section 8.

2. Handling multilingual information

Existing search engines and directory services on the Web, such as WebCrawler, Infoseek or Excite are mostly designed without the multilingual issues in mind. Queries in languages such as Chinese or Japanese would often solicit nonsense results. Some others, such as AltaVista, do differentiate between documents of different languages and allow queries for documents of a specific language. However, it seems that the correct index terms cannot always be identified. For example, to search for documents in Chinese language about the H5N1 poultry virus ("bird flu") incident in Hong Kong, we submitted two queries to AltaVista's advanced search form on 12 January 1998. The first one uses the keyword "H5N1" and the second one a common and perhaps the only Chinese term for bird flu "禽流感", widely used in Hong Kong Chinese newspapers. We are confident that both terms are rather new on the Web and have only rarely, if ever, appeared in the Web before May 1997. Both searches were limited to Chinese Web documents only. For the H5N1 query, 19 document links were returned, among which 14 of them were relevant. Indeed, most of them are relevant news reports from Chinese newspaper sites and contain both the terms "H5N1" and "禽流感". However, when the Chinese keyword "禽流感" was used alone, we got 119 hits, but about 110 of the links have nothing to do with the H5N1 virus or the bird flu.

So, what are the difficulties in handling multilingual documents? Here, we identify four important ones:

Document language identification Search engines that process documents using language-specific information need to identify the language a document is using before it can proceed. A failure in the language identification would greatly affect the indexing correctness.

Document segmentation A document is separated into segments of text that contain index terms. As discussed in the introduction, this is usually not a problem for Latin-based languages such as English or French that use spaces to delimit words. However, for languages such as Chinese and Japanese where no delimiters are used, segmentation is a nontrivial problem.

Maintenance of index term dictionaries For dictionary-based indexing methods, index terms and stoplists need to be maintained. New terms have to be added and outdated terms deleted in a regular basis to keep up with the changes of the documents.

New index term identification Manual or automatic methods should be used to find out new index terms such as the names of people or new abbreviations.

In the following sections, we discuss how the problems above can be tackled in a Web environment.

3. Document language vs. coded character set

For structured documents such as those written using SGML or HTML, it is often easy to determine their languages by just looking at its `lang` attribute. However, for unstructured documents, language determination is often difficult. Yet, from the view of a document indexing program, what is important is not the language, but the character set the document is encoded in. Since every document is but a stream of bits, the coded character set determines how those bits represent characters. For example, if a document is in ASCII, we know that each byte represents a character

and that the byte range 00 to 20 hex corresponds to control characters. If it is in Big5, we know that a character can either be one or two bytes long, depending on the value of the first byte. In short, the coded character set determines what is a character, or what constitutes a basic unit from which a term is composed, whatever the language the document is in.

There are two major implications that some indexing can be done once the coded character set is known. First, we can index documents even if we do not know the exact language it is written in. Some information about the language can often be inferred from the coded character set used. For example, if the coded character set ISO 8859-1 (often called "Latin-1") is used, although we do not know which language among the 34 countries listed in the standard [2] the document is written, we can assume that whitespace characters are used to separate words in the document. This piece of information alone is enough to help segmenting documents for finding index terms.

Second, a system does not have to keep a lot of language-specific information to do indexing. It is very useful in a multilingual environment, since it would be rather difficult, if not impossible, to collect information, such as grammar and word dictionaries, about all the languages in the world. Also, since language-specific information is not used, attributes such as language directionality (left-to-right as in English or right-to-left as in Arabic) would not affect the indexing process. Of course, it would be useful if we have the dictionaries and grammars, but they are not mandatory and we can do without them.

4. An abstraction

Coded character sets determine how we interpret the bits of a document seen as a sequence characters. This is nothing new. However, a document can also be treated as a sequence of other things — subword units (such as 'ter', 'un', 'th'), words, syntactic units (noun phrases, adjectives), words derived from word stems (both "smile" and "smiling" have the same word stem), stanzas, sections, runs of alphabets and nonalphabets, runs of digits and nondigits, and others. More abstractly, we can treat a document as a sequence of symbols. A symbol represents one of the "things" we mentioned above. Depending on what we choose the symbols to be, different sequences of symbols can be associated with the same document. Let us illustrate the idea with an example.

Figure 1(a) shows a quotation by Thomas Hewitt Key[1]. Suppose we have an English word dictionary and assign a symbol to every word in the quotation that is found in the dictionary, we obtain the 10-symbol sequence corresponding to the framed words in Figure 1(b). This sequence can be represented by the sequence of Greek letters in Figure 1(c), with α representing "What", β representing "is", and so on. Note that in this case, language-dependent information, that is, a dictionary of the English language, is needed. Moreover, the dictionary has to be complete. That is, it must have recorded all the possible index terms. Otherwise, some terms cannot be successfully matched and will be missed from the index.

(a) What is mind?No matter.What is matter?Never mind.
(b) What is mind?No matter.What is matter?Never mind.
(c) $\alpha\ \beta\ \gamma\ \delta\ \epsilon\ \alpha\ \beta\ \epsilon\ \zeta\ \gamma$
(d) What is mind?No matter.What is matter?Never mind.
(e) $\psi\ \omega\ \psi\ \omega\ \psi\ \omega\ \psi\ \omega\ \psi\ \omega\ \psi\ \omega\ \psi\ \omega\ \psi\ \omega\ \psi\ \omega\ \psi\ \omega$
(f) What is mind?No matter.What is matter?Never mind.
(g) $\alpha\ \beta\ \gamma\ \delta\ \epsilon\ \alpha\ \beta\ \epsilon\ \zeta\ \gamma$

Figure 1. Sentence as sequence of symbols

Now suppose we do not have a complete dictionary. We have to use features of the coded character sets to do the segmentation. Since the type of character, such as whether it is an alphabet or not, is often one such attribute, we can treat a run of alphabets as one kind of symbol (ψ) and a run of nonalphabets another (ω). This way, we obtain the segmentation shown in Figure 1(d). The corresponding symbol sequence is shown in Figure 1(e). Although the symbol sequences are different, the symbol boundaries coincide with that in Figure 1(b). To find segments of text containing possible index terms, we can first remove all the ω's, which corresponds to the nonalphabet runs, from the symbol string. Then, the corresponding text for the remaining 10 ψ's, shown in Figure 1(f), are used for indexing. After the ω removal, we can reassign the symbols so that they identify with the text the ψ's represent. For example, since the text of the first ψ, "What", does not identify with that of the second, "is", they are assigned different symbols α and β. And since the text of the first and the sixth ψ's are identical, they are assigned the same symbol α. After this symbol reassignment, we obtain the symbol sequence shown in Figure 1(g), which happens to be identical to that in Figure 1(c). Of course, such an ideal segmentation is not always achievable, especially for languages such as Chinese and Japanese where no delimiter is used to separate terms. However, by just considering a document's coded character set, one is able to segment it into basic units for index term extraction. Thus, this procedure is *generic* in the sense that documents in new or unknown languages can easily be accommodated. Dictionaries are not always required, and a *minimal amount of language-specific information* is used.

[1]Thomas Hewitt Key (1799-1875), a Professor of Mathematics who left University of Virginia in 1827.

5. Finding index terms

As we have discussed in the introduction, a dictionary of index terms is very useful in document segmentation and key phrase extraction. This is especially so for languages such as Chinese and Japanese, whose term boundaries are not delimited by special characters such as whitespaces. Unfortunately, a static dictionary can never be complete due to the highly dynamic nature of the Web. New information and index terms are constantly created. (Who would have thought that "tamagotch" is a valid term of something that people are interested before the toy is invented?) It is thus very important that a search engine be able to recognize or to deduce new index terms from the documents it processes. Timely recognition is essential for search engines to keep their key term indices abreast with the current state of the Web.

Seeing a document as a string of symbols, we can discover index terms by analyzing the symbol strings statistically. Since a symbol can correspond to different text entities, an index term here can correspond not only to words but also phrases and other text sequences. This analysis is *generic* in the sense that all is required is a choice of what a symbol corresponds to. To do the analysis *efficiently*, we introduce a data structure for finding substring frequencies — the suffix tree [5].

Given a string, its suffix tree is a level-compressed dictionary trie of all the suffixes of that string. For example, the suffixes of the string "banana" are "banana", "anana", "nana", "ana", "na", "a", and the null string. Its suffix tree is shown in Figure 2(a).

Observing that the prefix of a suffix of a string is a substring, we can see that each node of a suffix tree corresponds to one or more substring of the string it is built from. For example, the unshaded node in Figure 2(a) corresponds to the substrings "an" and "ana", and are prefixes of the suffix "anana" of the string "banana".

A suffix tree can be built in $O(n)$ time, where n is the length of the string. It utilizes $O(n)$ space and there are offline [3] and online [6] algorithms for its construction. Since all the substrings of a string are represented in its corresponding suffix tree, it can be nicely used to collect the occurrence frequencies of symbol subsequences in our symbol sequence analysis stage. Indeed, the number of leaves of a subtree rooted by a node is the occurrence frequency of the strings represented by that node. As an example, the subtree of the node corresponding to "a" in Figure 2(a) has three leaf nodes. Thus, "a" is the prefix of three suffixes of "banana". In other words, the substring "a" appears three times in the string "banana". As another example, the node corresponding to the substring "nana" is a leaf itself. Thus, it appears only once in "banana". Similarly, since the subtree under the node corresponding to the substring "an" (both "an" and "ana" correspond to the unshaded node) has two leaves, so "an" appears twice in "banana". Note that when two nodes represent the same substring, as we have now for "ana" (both the unshaded node and its right child), only the parent should be considered. Some authors eliminate this special case by appending a unique end-of-string symbol to the original string the suffix tree is built from.

Now, we see that suffix trees can be used to analyze for patterns from a symbol sequence. Combined with the symbol abstraction technique, it would then give us a powerful framework for extracting index terms. We not only can extract commonly-occurring character sequences but also word sequences, phrases, or grammatical structures by assigning different meanings to the symbols. Let us look at a phrase extraction example. Suppose a suffix tree for the symbol sequence "$\alpha\beta\gamma\delta\epsilon\alpha\beta\epsilon\zeta\gamma$", which comes from the text shown in Figure 1(a), is built. Using the tree, we know that "$\alpha\beta$" appears twice in the string. Thus, the text corresponding to "$\alpha\beta$", namely "What" and "What is", appear twice in the original text. Similarly, the node "$\zeta\gamma$" is a leaf in the tree, and thus "Never" and "Never mind" appear only once. Because each symbol in this example represents an English word, the analysis of symbol sequence occurrence statistics actually corresponds to the analysis of word sequences of the original document. Given the many different ways a document can be converted into a symbol sequence, analysis of the symbol sequences statistics is a powerful way for finding new index terms.

6. Applications

The use of the symbol abstraction together with the analysis using suffix tree is useful for document indexing and phrase extraction in a multilingual environment. We present a few application examples in this section.

6.1. Finding common character sequences

In documents of an unknown language, common character sequences often designate index terms, stop words or delimiters of other entities. Efficient identification of common character sequences of a piece of text helps its segmentation. Here is an algorithm for finding common character sequences:

1 Identify each character in the document with a symbol.
2 Build a suffix tree for the symbol string.
  Using the algorithm in [6], this takes time linear to the length of the symbol string.
3 Find the number of leaves under every node of the tree.
4 Select a threshold, traverse the tree, and collect those nodes (except the root node) that has more leaf nodes under it than the threshold.

Threshold selection can be done using heuristics. For example, if we are processing a document coded in character sets for Latin-based languages, we can set it to be a thousandth of the file length since that roughly corresponds to words of 5 to 7 characters long appearing in about 1/150 of all the words in the document.

Subtrees can be pruned when we traverse to a node whose number of leaf nodes under it is less than the threshold, since the number of leaf nodes under a child node can never be greater than that under the parent.

5 Eliminate nodes that correspond to symbol strings that are prefixes or suffixes of another.

This can be done by sorting the symbol strings corresponding to the collected nodes and examining the adjacent entries.

6.2. Finding new index terms

Now, suppose we have a partial dictionary of index terms and a stop list and want to find new index terms. Obviously, these terms cannot be any of the existing entries in the dictionary or stop list, otherwise they would not be new. They must correspond to document text that could not successfully match the dictionary and the stop list. Since both the dictionary and the stop list are but lists of words, we can make things conceptually simpler by considering them to be in a single dictionary. What we need to do is to purge all the dictionary-matching entries of the document text and statistically analyze the remaining text fragments. Assuming that new index terms appear moderately frequently in those fragments, suffix tree again can be used for the frequency counting analysis. Here is an algorithm:

1 Tree $T =$ null tree

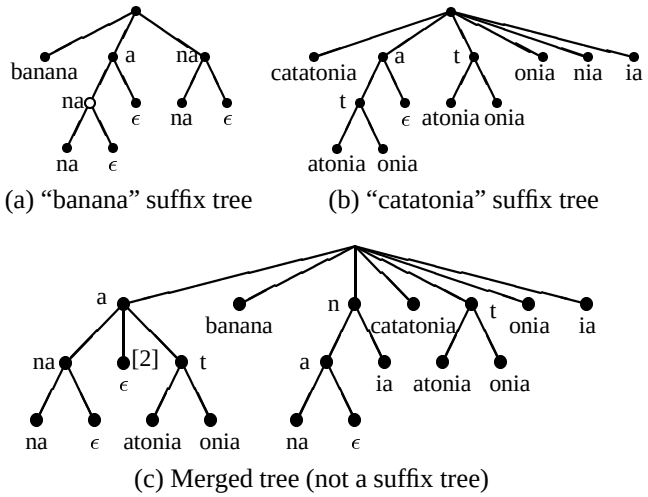
2 For each document,

2a Treat fragments of the document that match a dictionary entry as the symbol ω and non-matching segments as the symbols ψ

2b For each ψ, $T \leftarrow$ TreeMerge(T,SuffixTree(TextOf(ψ)))

Standard tree merge algorithm can be used, taking care that some of the leaf nodes represent a count of more than one after the merge. Note that after the tree merge, T is not a suffix tree in general. For example, Figure 2(a) and (b) show two suffix trees for the strings "banana" and "catatonia". Merging them results in the level-compressed trie in Figure 2(c). Since the text for the middle child node corresponding to the node "a" appear in both the "banana" and "catatonia" trees, it should be counted twice in the merged tree, and is thus marked by a "[2]" there. So, when finding the "number of leaf nodes" the node "a" has in the merged trie, the answer will be six rather than five.

3 Frequently appearing strings that does not match any dictionary entries can be found by counting the number of leaf nodes under every node of the trie T in the same fashion as in steps 3 to 5 of Section 6.1



(a) "banana" suffix tree (b) "catatonia" suffix tree

(c) Merged tree (not a suffix tree)

Figure 2. Merging suffix trees

7. Experiments and results

As a proof of concept and to test for the effectiveness of our procedures, we carried out a series of experiments to see how well our algorithms can find index terms or stop words of an unknown language with minimal use of language-specific information. First, a number of newspaper editorials in Chinese using Big5 code were obtained from the Web. From the coded values of characters, we separate runs of ideographic characters from nonideographic runs and treat them as two kinds of symbols. Ideographic runs so separated are usually Chinese phrases in our documents.

To see the effectiveness of statistical analysis using suffix trees, we compared the number of meaningful index terms found before and after the analysis. Before the suffix tree analysis, we found the possible index terms by applying the heuristics that very short ideographic runs, such as those not more than four characters long, are possibly index terms. We thus recorded, in Table 1, the number of ideographic runs of any length, the number of ideographic runs not more than four characters long, and the number of such meaningful ideographic runs. By meaningful we mean those segments that make sense to an ordinary Chinese person and can be used for indexing or in a stop list. This includes, among other things, proper names, connectives, adjectives, adverbs and phrases.

After recording the short ideographic runs, we found the set of index terms by building and merging suffix trees us-

Doc. date (mmdd1996)		0922	1022	1122	1222
Characters count		874	1185	1092	1116
no. of ideographic run		67	114	99	82
Ideographic runs, len≤4	C	5	26	18	8
	M	3	21	13	6
	P	60%	81%	72%	75%
Character sequences appearing more than thrice	C	6	17	8	7
	M	5	17	7	5
	P	83%	100%	88%	71%

C=count, M=meaningful, $P = 100\% \times M/C$

Table 1. Experimental result

ing the algorithm in Section 6.2. The suffix trees built correspond to the character sequences of the ideographic runs of any length. Strings of any length that occur at least three times in the merged tree (three corresponds to about 1/350 of average file size) were then examined to see whether they are meaningful or not. The result is shown in Table 1.

From this pilot study, we found that statistical analysis using suffix trees is helpful in finding effective index terms from the text segments. For example, for the editorial on October 22, only 26 text segments out of the 114 were of length no more than four, and 21 of them are meaningful. However, after analyzing all the 114 segments by building and merging suffix trees, 17 index terms were found and all are meaningful. Although the number of index terms found is less than that found by using heuristics, the accuracy is much higher. Indeed, for the September 22 editorial, suffix tree analysis can give more index terms than that found using heuristics. Statistical analysis using suffix tree has the added advantage of not imposing any length limitation on the index terms found. This is in sharp contrast to our short-ideographic run heuristics where every index term cannot exceed a certain length. The efficiency of the suffix tree, together with the effectiveness of the terms found, makes statistical analysis very useful.

We have also implemented an experimental search engine (`http://www.cs.hku.hk/~catfind/`) that indices Chinese news editorials using symbol abstraction and statistical analysis but not Chinese term dictionaries. Although we are still collecting queries to evaluate its effectiveness, we found that it can discover index terms that have never appeared before, such as people's names (李麗珊, 張偉良) and translated names of objects (他媽哥池 (tamagotch)).

8. Summary and future work

The Web spans across more than a hundred countries with users speaking more than twenty languages. Successful web search engines thus have to handle multilingual information in a *generic* way such that documents in new languages can be *efficiently* accommodated with *minimal language-specific information*. Also, as the Web is dynamic and constantly evolving, automatic methods for discovering index terms *effectively* from documents in a *timely* manner are needed.

To achieve these goals, we introduced an important abstraction of viewing documents: they are but sequences of symbols, which can represent characters, subword units, words, or runs of alphabets and nonalphabets. Documents so viewed can then be analyzed for frequently-occurring patterns using a data structure called suffix tree. As seen in Section 6, this analysis, combined with the abstraction, is powerful and can be used to find frequent phrases, extract new index terms, and help document segmentation even if only the coded character sets the documents use is known. Experimental work using Chinese language newspaper editorials has confirmed that even simple frequency analysis of commonly occurring symbol sequences are useful in finding new index terms.

We have implemented an experimental search engine (`http://www.cs.hku.hk/~catfind/`) that indices Chinese news editorials using the principle of symbol abstraction and statistical analysis, without the use of a Chinese term dictionary. It is found to be able to discover names of people or objects quite well.

Since in essence we are discovering patterns in documents, we believe that this symbol abstraction and statistical analysis method has potential in other areas than multilingual information retrieval. One such application we are currently investigating is on the recognition of chord progressions in musical pieces.

References

[1] Internet society, `http://info.isoc.org/`. Homepage.

[2] International Organization for Standardization. *ISO 8859-1:1987: Information processing — 8-bit single-byte coded graphic character sets — Part 1: Latin alphabet No. 1*, 15 February 1987.

[3] E. M. McCreight. A space-economical suffix tree construction algorithm. *Journal of the Association of Machinery*, 23(2):262–272, Apr. 1976.

[4] M. Sintichakis and P. Constantopoulos. A method for monolingual thesauri merging. In N. J. Belkin, A. D. Narasimhalu, and P. Willett, editors, *Proceedings of the 20th Annual Interantional ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 129–138. The Association for Computing Machinery, July 1997.

[5] G. A. Stephen. *String Searching Algorithms*. World Scientific, 1994. ISBN 981-02-1829-X.

[6] E. Ukkonen. Constructing suffix trees on-line in linear time. In J. van Leeuwen, editor, *Algorithms, Software, Architecture: Information Processing 92*, volume 1, pages 484–492. Elsevier Science B.V., 1992.