AUTOMATIC INFORMATION RETRIEVAL
BASED ON
GRAPH-THEORETICAL CONCEPTS

AUTOMATIC INFORMATION RETRIEVAL
BASED ON
GRAPH-THEORETICAL CONCEPTS

By

Tetsuro ITO

March, 1978

ACKNOWLEDGMENTS

The author wishes to express his thanks to Professor Makoto Kizawa who has guided him into the field of information retrieval studies, and has given the continuous advice and encouragement through the conduct of this thesis. Similar acknowledgment is also due to Assistant Professor Junichi Toyoda.

He expresses the special appreciation to Professor Masamichi Shimura (presently with Tokyo Institute of Technology) for his instructive criticisms during his residence in Prof. Kizawa's Laboratory.

He would also like to thank Professor Kokichi Tanaka who has conducted the author into the field of information sciences and given excellent suggestions.

The author is indebted to Assistant Professor Shinichi Tamura and Dr. Masaharu Mizumoto for the ideas in their enlightening papers. Assistant Professor Tadahiro Kitahashi has willingly given the useful advices from the beginning of this study.

It is with author's pleasure to express his gratitude to his friends Drs. Yoshinori Ezawa (presently with Kansai University), Kazuyoshi Mikami (presently with Mitsubishi Electric Co. Ltd.) and

Shoji Tominaga (presently with Osaka Electro-Communication University) for their helpful suggestion and cooperation.

The author would like to express his thanks to Messrs. Hiroshi Makino, Kohei Hanatate, Hideo Kudo and the staffs and students of Prof. Kizawa's Laboratory, and his special thanks to Dr. Riichiro Mizoguchi (presently with Osaka Electro-Communication University) for his useful discussions.

Final acknowledgment is due to the author's family for their invaluable help from a spiritual point of view.

PREFACE

The explosion of stored document information and the wide spread utilization of document data banks require that the computer aided information retrieval systems should be designed to permit automatic processing of a large amount of documents and various information queries of non-expert users. The success of automatic data banks and retrieval systems depends upon the accurate and consistent description of the document data, which includes ill-structured or ambiguous properties of natural language. It is then important for the processing of a large amount of documents to make *fundamental* studies of analyzing conceptually the natural language document sentences, and retrieving effectively the relevant document information to the various types of submitted queries. Many of the existing retrieval systems utilizing conventional library techniques of manual keyword indexing, however, have been inefficient to permit such a flexible treatment of the stored document information.

This thesis discusses, leaving information storage and retrieval tasks under manual keyword indexing to the librarians, the automatic method of obtaining document themes through the relevancy computation between the stored information and the given queries. It is noted that the analysis, classification or description of textual information for easy relevancy judging is scheduled so as to handle a large amount of document data covering wide variety of specified topics, and to control the future extension and changes of the document collection. An automatic analysis and matching method of textual data is then presented based on a predicate network for the document sentences.

After the drawbacks of conventional information retrieval (IR) systems are indicated in Chapter 1, the methods of generating a word usage dictionary and decomposing document sentences into predicative expressions are discussed in Chapters 2 and 3. The relevancy of the documents to the information queries is obtained by computing, via the dictionary, the similarity between predicative expressions of the document sentences and the queries. Since the automatic systems ignoring the prevalence of ill-structured information have seriously failed to live up to their technological promise, the specification of ambiguous property of document sentences might be important to the contextual information analysis for natural language IR systems.

The method of organizing a structured file is discussed in Chapter 4 for the fast retrieval of the required information. The document information retrieved from the structured file is arranged according to the magnitude of relevancy coefficient to the submitted queries. When a data bank of natural language documents is searched for answering effectively the submitted queries, the specification of ambiguous data becomes one of the important factors for the well-structuring of the document file.

Some of the other fundamental problems in IR tasks are stated in Chapter 5, which is the concluding chapter. Finally in the Appendix several algorithms, together with a new data structure for expressing in the computer an undirected finite graph, are formulated for extracting graph-theoretical concepts which are used in this thesis.

Toyonaka, Osaka, JAPAN Tetsuro Ito

Contents

CHAPTER 1

Introduction

The basic function of an automatic information retrieval system is discussed in comparison with the manual keyword indexing systems. It is suggested that IR systems should be designed so that it can treat the problem of which documents are conceptually related to the given query. The notions of an automatic dictionary construction and predicate network representation for natural language text processing, and a structured file organization for fast retrieval of required documents are introduced using graph-theoretical concepts.

1.1 Information Retrieval in Laboratory Environment

The problems of retrieving document information in a laboratory environment are classified into the following two types.

1) Selection of the documents which include the specified bibliographic elements used for recording such as the title of a book, author's name, publisher and place of publication, etc.

2) Selection of the documents whose information contents refer to the subject fields of the submitted queries.

A descriptive cataloging technique employed in a conventional library system provides the main file searching tool for the first requirement. Another cataloging operation, i.e., subject cataloging, which is defined as the systematic indexing of the documents by the class numbers or keywords for representing the subject themes, has also been used in IR systems to answer the second problem. Some of the classification schemes for subject cataloging, e.g., Universal Decimal Classification (UDC), colon classification, or subject catalog in alphabetic order, are well known to the librarians [3, 35]. The set of index terms covering these schemes is called an authority list of a controlled vocabulary of terminology.

From a historical consideration shown above, the authority list designed in the manner to cover a wide variety of specialized themes seems to give an unsatisfactory solution to the content analysis of professional journals, technological assessment reports, etc. The manual analysis of such professional materials, however, needs time consuming and costly perusal process of document classification or subject heading assignment, and most of the research workers use non-authorized vocabularies to index their documents and search requests. Thus it is hard for information retrieval to use the subject cataloging schedules of historical types. These defects come from the fact that the strictly restricted or well-structured identifiers of discrete information are used for the subject analysis of documents including complex subject entities. Since any document in general has many aspects of information contents showing high relevancy to various search queries, the success of information retrieval systems depends upon the fundamental study of analyzing contextually the document data and obtaining effectively the relevancy between the documents and the queries.

This thesis presents a method of representing document sentences in the network form of predicative primitives so that the relevancy computation between two sentences could be automatically carried out. The subject themes of documents are then interpreted by the "relevancy" between the sentences of the documents and the search requests. Such approaches to text processing will become more promising as both the technology for handling a fast-accessible memory with a large capacity and the method of knowledge representation studied mostly in the field of artificial intelligence become more available.

Any sentence in the predicate network is expressed by the combination of predicate patterns and word classes to show how the words are used in the given document sentences. The predicate network is, by its simple structure of representing conceptual relations between sentences, useful for the processing of large textual data for document retrieval. Further its flexibility of expressing content information would contribute to the high retrieval performance of IR systems.

The structured file organization, from which the required documents are retrieved by arranging in the order of magnitude of relevancy to the query, is also proposed for the effective searches of a document file. Since the performance effectiveness of an IR system is heuristically interpreted based on the user's experience, the arranged output of the document list is useful for a direct evaluation of ranked positions of relevant documents. Also, the relevancy between two documents should be properly formulated for the effective search of the structured file.

1.2 Historical Survey

During the last decade many experimental or practical information retrieval systems have been developed in various controlled environments. But nowadays only limited systems employing manual or semi-automated keyword indexing schedules are put into practice for handling "real world" information. The SMART document retrieval system [46-52], among others, designed at Harvard University, and operated at both Harvard University and Cornell University by Salton, has provided a lot of experimental tools for an automatic document processing method, and has influenced on many other retrieval systems. This system can take documents and search requests in English, i.e., natural language, by fully automated manner, and can retrieve, utilizing feedback information from the user population, the documents believed to be most relevant to the submitted queries. Further several evaluation parameters for the measurement of the system performance have been introduced for the appropriate comparison of different types of analyses and search procedures [50]. Some of the theoretical studies have also been carried out for language analyses including subject indexing and thesaurus construction [1, 2, 7, 8, 16, 24, 36, 48, 59, 67], and for file organizing and searching methods [11, 12, 32, 49, 56, 62].

The earlier study of computer aided information retrieval was devoted to computerizing the already existing library systems [3, 35], where documents were indexed using descriptive and subject catalogs, and the required documents were selected by the set-theoretical matching between descriptors. The preparation of an accurate and consistent cataloging schedule by human endeavor, however, is too poorly accomplished and too time consuming. As the data processing ability of computers increases, the researches have advanced into the statistical study of indexing [8, 36, 46], abstracting [16] or analyzing the documents by the computer aids: The techniques based on the syntactic property of sentences [2, 58] were also designed for automatic or semiautomatic subject analyses. These ideas, however, are of no use when the retrieval system is intended to work on the more general document data base than the precisely defined one, for the prevalence of ill-structured or ambiguous properties of natural language document sentences has prevented the IR system from the successful automation of storage and retrieval tasks. Present-day studies of language analysis [6, 10, 17, 22, 53, 54, 55, 65] have been prosecuted by the formulation of a semantic network model for representing conceptual comprehension process of question answering, sentence paraphrasing, story understanding, etc.

Keyword indexing techniques (utilizing purely manual, machine or man-machine combination means) intended to retrieve the bibliographic information identical to the query indices have also been used for organizing a document file. Burkhard [11], Jardine [32], Salton [49] and Shapiro [56] have intended to solve the problem of searching effectively the document file by constructing the groups of the related documents. Filing methods now have been studied in relation to the data base technology for dealing with the data availability, privacy, security or independency [12, 13, 14, 37, 44].

When these new techniques are intended to be applied to the retrieval of document information, they should be modified so that a large amount of specialized textual materials will be systematically processed. The study of the evaluation method of system performances is also an important problem. Ricchio and Keen [50] have presented, in this connection, some of evaluation parameters in addition to the usual recall and precision factors. Lancaster has evaluated thoroughly in [35] the operational and economical performance of IR systems.

1.3 Graph-Theoretical Approaches to Information Retrieval

Fig. 1.1 outlines the formal tasks of information retrieval systems using relevancy computation between the documents and the search queries (the IR tasks generally are explained as the storage and retrieval process of bibliographic data), where notation a (actual data) denotes the set of data usable as an input to the detailed process p, and notation n denotes the description of another needed information for the execution of p. As shown in Section 1.1, the success of the information retrieval system depends upon how well the document information is analyzed and then organized to obtain the conceptual *relevancy* between document sentences.

This thesis describes a graph-theoretical scheme of analyzing document data using a similarity weighted graph G_W. The nodes of G_W correspond to the natural language units (words, sentences, documents, etc.) and the weighted lines to similarity values between those units. Similarity weighted graphs were often used for analyzing various problems related to IR studies [2, 7, 24, 58] because of its concise expressing power of complicated relations in multivariate data.

It has been emphasized that the intensional characterization of the words rather than the extensional one is needed to the semantic

Process			Description
1 INPUT	1.1 Accumulation		a) input texts, stop words and function words p) storing of document information in the matched position n) (*Human*) → (3.1) → (3.2)
	1.2 Query Analysis		a) information query p) same as the above n) (3.1) → (3.2) → (2.1)
2 OUTPUT	2.1 Editing		a) document list with similarity values for a query p) listing of matched information according to a given format n) output convention is an ordering of documents according to similarity values.
3 MAIN	3.1 Language Analysis	3.1.1 Syntax Anl.	a) application pointers p) setting of predicate patterns n) (3.1.2)
		3.1.2 Semantic Anl.	a) predicate patterns p) setting of application pointers n) (3.1.1)
	3.2 Matching		a) predicate network p) searching into the document space for the similar patterns to a query, by referring word usages n) matching results with similarity values
4 REFINEMENT	4.1 Evaluation		a) output listing p) computing of the retrieval factors n) (*Human*) → (4.2)
	4.2 Feed-back		a) evaluation results p) settling of a new information query n) (1.2)

Fig. 1.1. Modeling of IR system.

processing of natural language sentences. Here, since the IR tasks are explained as the relevancy computation process, any word is characterized by relating it to the synonym word classes in which each class corresponding to a word application area consists of the words having similar usage experiences within the given document sentences. Thus the similarity, i.e., line weight of G_W, between two words x and y is obtained by their usage experiences as follows: Let $\mathcal{F}=\{z: Fxz\}$ and $\mathcal{F}'=\{z: Fyz\}$ be classes (or class abstracts in Quine's notation [43]) for predicate F, which purport to designate the sets of all words z such that Fxz and Fyz, respectively. Then the similarity between x and y via F is given by the concretion of a relation abstract $\mathcal{R}_F=\{(x,y)$: x and y have the similar usage experiences with respect to $\mathcal{F}$ and $\mathcal{F}'\}$ with (x,y) as: $f_{R_F}(x,y)$ for $x\mathcal{R}_F y$ $(f: X\times X \rightarrow [0, 1])$. A set {G} of undirected graphs is derived by considering a line of G_W having a specified weight to be a line of G.

Chapter 2 discusses, by assuming graph G_W for a word set is given, methods of extracting synonymous word classes and ambiguous words based on cliques and cut-nodes. Two types of complexities caused by the ambiguous properties of the words are introduced to define multiple meaning words. The notion of multiple meaning is important to analyze formally the natural language document data consisting of a mix of well-structured and ill-structured information. The clustering schemes using cliques to obtain word classes have been believed to be invalid when an input graph is large in node size. The computing speed for generating cliques increases in the order k^n (k is an integer) for n nodes. It is noted, however, the speed of a clique finding algorithm mainly depends upon the line density rather than the node size. Thus there is no problem from a computational viewpoint to use cliques for analyzing graphs with a sparse matrix representation.

The contextual analysis of document sentences for obtaining similarity between two words (or sentences, documents, etc.) is attempted in Chapter 3 using a network representation of predicate patterns and word classes, called a predicate network. An automatic method of constructing a predicate network is an iterative procedure such that predicate patterns for relating words (which appeared as the argument values of the predicate patterns) are settled based on word classes, and conversely words are grouped based on the similarities of their usage experiences on the predicate network at the previous iteration. Thus both a predicative expression of a sentence and a word usage dictionary are needed for automatic natural language information retrieval.

Chapter 4 aimed at organizing hierarchically related document files for the effective retrieval of required information covers a method of arranging the document data in a decreasing order of similarities between documents. The result of file searching is ranked output of the document list, which contributes to the performance efficiency of an automatic IR system. Also well-structuring of ambiguous documents is an important factor for the fast retrieval of required results. The document files are shown to be placed on a storage hierarchy which is one of the most desirable formulations for storing a large amount of document data.

In Chapter 5, other important problems in the IR fields are briefly outlined, and in the Appendix a new data structure for expressing graphs G_W or G is presented using a labeled similarity or adjacency matrix, and several algorithms for extracting graph-theoretical concepts used in this thesis are formulated based on a matrix rearrangement procedure.

CHAPTER 2

Dictionary Construction for Natural Language Text Processing

A graph-theoretical method of constructing a word usage dictionary is presented for an automatic content analysis of document sentences. First an algorithm based on the fuzzy transitive inequality is formulated for defining a graph from which all the synonymous word pairs and ambiguous words of the given data are extracted. It is shown that both synonym classes and ambiguous words are specified by using cliques of the graph. Second two types of complexities caused by the ambiguous property of the data are defined to find multiple meaning words. The notion of multiple meaning is used for obtaining application areas of any word within the given data set. Experimental construction of a dictionary operated on a set of verbs in the scientific documents is also analyzed.

2.1 Introduction

Natural language information retrieval systems have firstly been developed at the aim of performing an automatic content analysis of texts and search queries for determining the relevancy between them [50]. Early computer aided information retrieval systems, most of which were developed from manual library systems, utilized the already established calssification schedules to decide which categories would fit the given items most reasonably, and to search the file for the required bibliographic information. The assignment of subject identifiers to the documents, however, are not suitable to discriminate the underlying subjects of the stored items. The problem of automatic content description and analysis of written texts would be solved by establishing a computational method of representing syntax and semantics of natural language document sentences.

Various types of semantic dictionaries (including a semantic network) have been concerned with the contextual analysis of natural language. As explained in Chapter 1 (that the formal information retrieval task is the relevancy or similarity computation process between document sentences), the dictionary which correlates the synonyms with one another based on the usage experiences in the given documents is necessary in information analysis. The synonymous property of the words, however, causes logically the ambiguous property which brings the complexities or irregularities of the syntactic and semantic relations of natural language. The specification of ambiguous words and, if possible, their proper application areas or senses in a given context is therefore an important problem for the automation of natural language information retrieval. The improper specification of ambiguities is shown to cause an incorrect recognition of the word senses. Further the efficiency of file search algorithms is affected by the existence of ambiguous documents.

The establishment of an automatic method of constructing synonym dictionaries has been attempted by various research workers. Borko [8] and Anderson [1] used the principles of factor and latent analyses to find synonym classes from a similarity matrix of the given data. Their techniques are only applicable to the analysis of a word set rather small in size because of their complicated computations and manipulations of the matrix. Other methods [2, 24, 59] based on graph-theoretical concepts have also been presented for finding synonym word clusters. The graph was introduced as a model of describing a similarity relation on the data of multidimensional space. These graph-theoretical techniques have defined graphs by setting a threshold level or a nearest neighbor rule to the similarity matrix. Dictionaries generated by using such automatic methods, however, do not successfully handle ambiguity and synonymy in natural language, and then seem not to be appropriate for the semantic processing of natural language texts.

The graph-theoretical method developed here [27, 29], given a usage similarity of every word pair, generates a dictionary to obtain word application areas for the contextual analysis of textual data. Note that the specification of word application areas is required even when the words are disambigulated contextually in a specified sentence. First an algorithm based on a fuzzy transitive inequality is presented for determining a synonym graph G of a given word set. The synonym graph G, from which synonymous word classes and ambiguous words are extracted based on the cliques, is produced by corresponding a word and a synonym indication between words to a node and a line of G, respectively. Next a method of finding multiple meaning words is proposed to treat the complexities or irregularities of the semantic relations which are caused by the ambiguous property of the words. The word application areas are, therefore, obtained by specifying the synonym word classes of characteristic non-multiple meaning words. The computational experiment operated on a set of verbs drawn from ABSTRACTs of 100 documents in IEEE Transactions is made to show the validity of the proposed methods.

2.2 Basic Concepts and Their Expressions

In this section, let us describe some definitions and notations for the generation of a word usage dictionary. The notion of fuzzy expression of a relation is first introduced to measure the similarity of the usage experiences for any pair of words in a sentence collection.

A fuzzy expression of any relation in a given word set X is defined as follows [68, 69]:

Definition 2.1. Let $\mathcal{R}=\{(x,y): x\mathcal{R}y\}$ be a relation abstract [43] which purports to designate a relation of all word pairs (x,y) such that xRy. Then the fuzzy expression of $\mathcal{R}$ is a function f_R such that:

$$f_R(x,y) \quad \text{for} \quad x\mathcal{R}y, \qquad f_R: X\times X \to [0, 1] \; (=L) \qquad (2\text{-}1)$$

where L is a complete lattice ordered semigroup[†].

We write $a \cup b$ for the least upper-bound of a and b ($a,b \in L$) and $a*b$ for the composition. When $\mathcal{R}$ is an "ordinary" relation, it is written in the same notation as R. Hereafter the function $f_R(x,y)$ is used to show the similarity of the usage situations for x and y.

The natural language coding of semantic entities could, if it had been devised logically and scientifically, have embodied the well-structured rule of "one name - one sense." Though such uniformity would only be practicable within certain limitation, more than one word is likely to be used to denote the specific entity in the given discourse. These words, called synonyms, should be treated together as a semantic unit for the processing of natural language. The synonymy,

[†] Complete lattice L is called a complete lattice ordered semigroup when L is a semigroup with identity under the operation *, which satisfies $a*\bigcup_i a_i = \bigcup_i (a*a_i)$ and $(\bigcup_i a_i)*a = \bigcup_i (a_i*a)$ for $a, a_i \in L$ [21]. We can employ ×(product) or *min* in place of * for example.

as we shall see, leads logically the other elasticity, i.e., ambiguity, of natural language. The next is the definition of a fuzzy synonym relation.

Definition 2.2. A synoymous word pair set $\{(x,y)\}$ (=E) in any similarity relation $\mathcal{R}$ is defined as a collection of the pair of words x and y satisfying (2-2), (2-3) and (2-4) for $(x,z),(z,y) \in E \cup \breve{E}$.
Reflexive law:

$$\bigcup_z (f_{R \cup \breve{R}}(x,z)) \leq f_R(x,x). \qquad (2\text{-}2)$$

Symmetric law:

$$f_R(x,y) = f_R(y,x). \qquad (2\text{-}3)$$

Transitive law:

$$\bigcup_z (f_R(x,z) * f_R(z,y)) \leq f_R(x,y). \qquad (2\text{-}4)$$

Here x, y and z are in X, and $\breve{R}$ is the converse of R.

Thus the word pairs almost equal in similarity constitute a synonym class.

The other complication, ambiguous property, is responsible for the difficulty of a formal treatment of natural language words. Quine [42] indicated that if word y is synonymous with some word x in one sense of y and with another word z in another sense of y, then y may be called to be ambiguous. Here, by a fuzzy expression, the ambiguity is defined as follows:

Definition 2.3. Let (x,y) and (y,z) be synonymous word pairs in $\mathcal{R}$. Then y is said to be an ambiguous word with respect to $\mathcal{R}$ iff the following inequality holds:

$$f_R(x,y)*f_R(y,z) > f_R(x,z). \qquad (2-5)$$

Thus the ambiguous property comes from the usage explications of a word in various contextual situations. It would be impracticable however to have separate symbolic expression (i.e., word) for every referent situation.

Finally let us define the meaning of any word by introducing a new reflexive and symmetric relation $R*$ whose value of the fuzzy expression f_{R*} for the word pair not satisfying the fuzzy transitive inequality is 0. Here the term "meaning" is used in the sense "alike in meaning" [42].

Definition 2.4. The meaning M(y) of any word y in X is a synonym class $\{x: xR*y\}$ of y, and the fuzzy expression of M(y) is a function $g_{M(y)}$ such that:

$$g_{M(y)}(x) = f_{R*}(x,y). \qquad (2-6)$$

Since the synonymous words should be treated as a semantic unit by the likeness in their usage situations, the "meaning" of any word means a set of semantic units to show how the word is used in relation to the other words. Hereafter every word is considered as a conceptual entities by relating the word with its meaning. Set X in which the meaning is defined for any word is called a meaning space M(X), and the meaning space in which the various semantic entities, e.g., synonymy, ambiguity, multiple meaning, are specifiable for any word is called a word usage dictionary.

In the following section, the algorithm, called algorithm A, for obtaining synonym relation R^* is formulated by giving a fuzzy expression f_R of a similarity relation R. The similarity relation for word set is practically computed based on the word usage experiences in the document sentences.

2.3 Algorithm A

The input to algorithm A is a fuzzy expression f_R of a similarity relation R satisfying fuzzy reflexive and symmetric laws. Algorithm A is intended to extract synonymous word pairs by the iterative executions of Steps 2, 3 and 4. In Step 2, it is examined whether a pair in set Y, which is used to hold the synonymous word pairs, satisfies the fuzzy transitive inequality or not. Any pair not satisfying the transitive inequality is excluded from Y to note that this pair is nonsynonymous in this iteration. In Step 3, word pairs which are restored to Y as the synonymous pairs are selected based on the fuzzy transitive inequality. The condition of the termination that the set of synonymous pairs newly selected in Step 3 is the same as the set in the previous iteration is tested in Step 4. Set Y is initialized, since no synonymous pair is known beforehand, to the collection $\{(x,y)\}$ of all pairs in X^2, and sets E and F, which is used to store the pairs determined to be synonymous in Step 3, are initialized to empty.

[Algorithm A] (Synonymous Word Pairs)

In: Symmetric similarity matrix R (any diagonal element is set to 1).

Procedure:

Step 1. Let Y be a set of all word pairs (x_i,x_j), $1\leq i\leq n$, $1\leq j\leq n$ $(n=|X|)$, in X. Initialize both E and F to $\emptyset$.

Step 2. Delete from Y all pairs (x_i,x_k) not satisfying the following inequality for some x_j such that (x_i,x_j) and $(x_j,x_k)\in Y$,

$$f_R(x_i,x_j)*f_R(x_j,x_k) \leq f_R(x_i,x_k). \qquad (2\text{-}7)$$

Step 3. If a pair (x_i,x_k) in $\bar{Y}$ (the complement of Y) satisfies inequality (2-7) for all x_j such that (x_i,x_j) and $(x_j,x_k)\in Y$, then set $Y=Y\cup\{(x_i,x_k)\}$ and $E=E\cup\{(x_i,x_k)\}$.

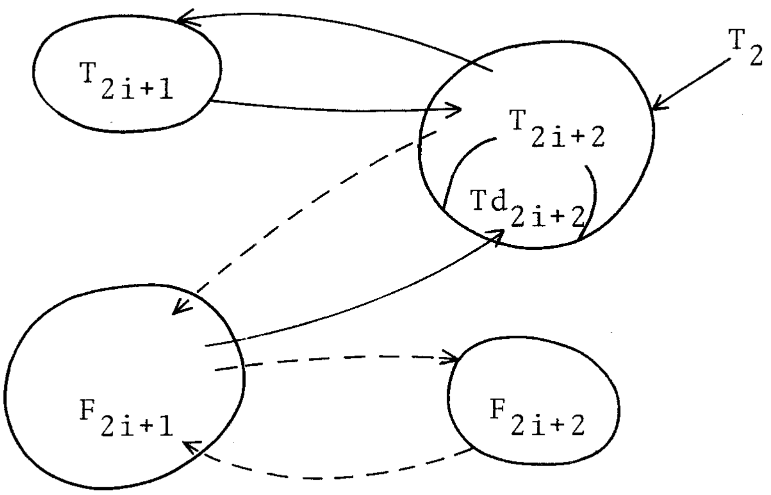
Step 4. If $E\neq F$, then go to Step 2 setting F to E and E to $\emptyset$; otherwise, continue to Step 5.

Step 5. Define the fuzzy expression of a new relation R^* to be f_{R^*} such that:

$$f_{R^*}(x_i,x_j) = \begin{cases} f_R(x_i,x_j) & \text{if} \quad (x_i,x_j)\in Y, \\ 0 & \text{if} \quad (x_i,x_j)\in \bar{Y}, \end{cases} \qquad (2\text{-}8)$$

and terminate the algorithm.

Set Y in Step 2 is monotonically decreasing in the number of elements during the successive executions of Steps 2, 3 and 4. It is possible to show that all of Y is synonymous word pairs in R and all of $\bar{Y}$ is nonsynonymous word pairs.



i : cycle number
——— : flow of synonymous word pairs
– – – : flow of nonsynonymous word pairs

Fig. 2.1. Timing expressions of sets Y, $\overline{Y}$ and E in algorithm A.

Theorem 2.1. Set Y in algorithm A converges, in a finite number of steps, to the set of all synonymous word pairs in $\mathcal{R}$.

Proof. Let the ith ($i \geq 1$) cycle be the ith execution of Steps 2, 3 and 4, and let T_{2i+1} (or F_{2i+1}) and T_{2i+2} (F_{2i+2}) be the timing expressions of set Y ($\bar{Y}$) in Steps 2 and 3 at the ith cycle, respectively (The timing expression of set Y in Step 1 is written as T_2). Sets T_{2i+1} and T_{2i+2} are said to be the sets of synonymous pairs, and F_{2i+1} and F_{2i+2} the sets of nonsynonymous pairs. Further let Td_{2i+2} be the timing expression of set E in Step 3 at the ith cycle (see Fig. 2.1).

If the sequences $<F_{2i+2}>$ and $<T_{2i+1}>$ of sets satisfy

$$F_4 \subseteq F_6 \subseteq \cdots \subseteq F_{2i+2} \subseteq F_{2i+4} \subseteq \cdots, \tag{2-9}$$

$$T_3 \subseteq T_5 \subseteq \cdots \subseteq T_{2i+1} \subseteq T_{2i+3} \subseteq \cdots, \tag{2-10}$$

then there exists an integer N such that for $n \geq N$, $F_{2n+2} = F_{2n+4}$ and $T_{2n+1} = T_{2n+3}$. Since, by the definition, $F_{2n+2} = F_{2n+1} - Td_{2n+2}$ and $Td_{2n+1} = T_2 - T_{2n+1}$. We have that $Td_{2n+2} = Td_{2n+4}$, the condition of the termination.

Next let us show that (2-9) and (2-10) hold.

Lemma 2.1. The sequences $<T_{2i+1}>$ and $<F_{2i+2}>$ are monotonically increasing with respect to (shortly w.r.t.) set inclusion.

Proof. For an arbitrary pair (x,z) in T_3, consider pairs (x,y) and (y,z) in T_2. Then we have that

$$f_R(x,y) * f_R(y,z) \leq f_R(x,z). \tag{2-11}$$

Since set T_2 includes set T_4, it follows that (2-11) holds for pairs (x,y) and (y,z) in T_4, and hence $T_3 \subseteq T_5$ and $F_3 \supseteq F_5$. Inequality (2-11), however, does not hold for any pairs $(x,z)\in F_4$ and $(x,y),(y,z)\in T_3$ (We describe this such that F_4 is false w.r.t. T_3). Therefore, since $T_3\subseteq T_5$, $F_4\subseteq F_6$ and $T_4\supseteq T_6$. Similarly, we have that $T_{2j+1}\subseteq T_{2j+3}$ $(j\leq i)$ and F_{2j+2} $\subseteq F_{2j+4}$ by assuming that $T_{2j-1}\subseteq T_{2j+1}$ and $F_{2j}\subseteq F_{2j+2}$. Thus, by induction, the proof is complete.

Lemma 2.2. There exists an integer N such that for $n\geq N$, every element in T_{2n+4} (or F_{2n+4}) is a synonymous (nonsynonymous) pair w.r.t. $\mathcal{R}$.

Proof. Suppose that $Td_{2n+2}\neq\emptyset$ for every n (≥ 1). Then there must be at least one pair (s,t) in Td_{2n+2}. If $|Td_{2n+2}|$, the number of the elements in Td_{2n+2}, is 1, then $(s,t)\notin F_{2n+3}$. Otherwise, let us assume that every pair in Td_{2n+2} is determined to be nonsynonymous in Step 2 at the (n+1)th cycle. Then for any $(x,y)\in Td_{2n+2}$, we have that $f_R(x,s_1)$ $*f_R(s_1,y)>f_R(x,y)$ for some $s_1\in X$ such that (x,s_1) (and/or (s_1,y)) is in Td_{2n+2}. By the above hypothesis, we have that $f_R(x,s_2)*f_R(s_2,s_1)$ $>f_R(x,s_1)>f_R(x,y)$ for some s_2 in X. Then (s_2,s_1) (and/or (x,s_2)) is in Td_{2n+2}, and $(x,s_1)\neq(x,y)$. Similarly for any $s_k\in X$ $(k\geq 3)$, $f_R(u,s_k)$ $*f_R(s_k,v)>f_R(u,v)>\cdots>f_R(s_2,s_1)>f_R(x,s_1)>f_R(x,y)$, where $u,v\in\{x,y,s_1,$ $s_2,\cdots,s_{k-1}\}$. It follows that (u,s_k) (and/or (s_k,v)) is in Td_{2n+2} and $(u,s_k)\notin\{(x,y),(x,s_1),\cdots,(u,v)\}$. Since the number of elements in Td_{2n+2} is finite, there must be a synonymous pair $(s,t)\in Td_{2n+2}$. Thus $(s,t)\notin F_{2n+3}$, which contradicts $T_{2n+1}=T_{2n+3}$, and hence $Td_{2n+2}=\emptyset$.

By Step 3, it is true that F_{2i+2} is false w.r.t. T_{2i+1}. Since Td_{2n+4} is empty for $n\geq N$, $T_{2n+3}=T_{2n+4}$. Therefore, any element in F_{2n+4} does not satisfy the transitive inequality for some element in T_{2n+4}.

Similarly, it follows that any element in T_{2n+4} satisfies the transitive inequality for any element in T_{2n+4}.

By Lemmas 2.1 and 2.2, we can complete the proof of Theorem 2.1.

Since, by Theorem 2.1, any word pair (x,y) satisfying $f_{R*}(x,y)\neq 0$ (word pairs satisfying $f_{R*}(x,y)=0$ may be considered as nonsynonymous pairs, for these represent a null relation) is mutually synonymous, relation R* defines the synonym graph G(X) such that nodes and weighted lines correspond to words and synonymous word pairs, respectively. The meaning M(x) ($g_{M(x)}(y)\neq 0$ for $y\in M(x)$) of any word x is expressed by the set of all nodes adjacent in G(X) to x. As every nonadjacent pair is nonsynonymous, two words belonging to a clique, or maximal complete subgraph, of G(X) are mutually synonymous and no other word is synonymous with all words of this clique. The following two properties are used to specify the ambiguity of word x. Here W is an ambiguous word set w.r.t. $\mathcal{R}$*, and MCS(x) is a set of cliques of G(X) containing x.

(III-1) Word x is a member of X-W iff the number m(x) of elements in MCS(x) is equal to 1.

(III-2) Word x is a member of W iff m(x) is greater than 1.

Thus the cliques of the synonym graph are the semantic units for obtaining the ambiguous properties of natural language words. The algorithms for finding all cliques of a graph are shown in [2, 9, 23] and in Appendix A.7.

2.4 Analysis of Ambiguity

The identification of the sense of a word in a given sentence is one of the most important problems for the automatic content analysis of natural language document sentences. Since, however, many words having several senses, called multiple meaning words, are included in the document sentences, the application areas of every word should be distinguished beforehand to disambigulate contextually the words in a given sentence. Here a computational approach to determining multiple meaning words is discussed using the complexity of the semantic relation. The set of word application areas is then settled by clsutering the meanings of non-multiple meaning words so that the semantic complexities should not be increased.

Ullmann [61] referred as multiple meaning to the complication of semantic patterns such that more than one sense is attached to one word and more than one word to one sense. The multiple meaning of a word here is treated as one of the chief symptoms of causing complication of the semantic relations by its ambiguous meaning.

The number of cliques of $G(X)$ containing a word w is shown to be sufficient to determine whether or not w has ambiguity w.r.t. R^* (see (III-1) and (III-2)). It may be interesting for specifying multiple meaning words, or words having multiple meanings, to formulate the complexities of semantic relations based on the cliques of the synonym graph. Two types, i.e., static and dynamic, of complexities are defined as follows:

Static complexity caused by a word whose meaning consists of the cliques with a small number of common words. The multiple meaning word obtained by the static complexity is considered to have several remote application areas.

Dynamic complexity caused by a word having a more ambiguous nature in a new meaning space N(X) than in M(X). Space N(X) is obtained by the execution of algorithm A on a relation S whose value of the fuzzy expression f_S is given by (2-16). Since in N(X) ambiguity of word w is measured by the number n(w) of cliques of the graph H(X) of N(X), a multiple meaning word w having a dynamic complexity is one which meets

$$m(w) \leq n(w) \ (\neq 1). \qquad (2\text{-}12)$$

The dynamic complexity is analyzed more explicitly as follows: Note that each member of M(w) is in general a member of N(w). (Consider the inequality $f_S(x,k)*f_S(k,w) \leq f_S(x,w)$ for $x \in M(w)$ and $k \in X$. Even if the value of $f_S(x,w)$ becomes smaller than that of $f_R(x,w)$, the value of $f_S(x,k)$ (and/or $f_S(k,w)$) generally becomes smaller than that of $f_R(x,k)$ ($f_R(k,w)$), and hence the above inequality holds (the experimental analysis verify this fact).) Suppose first that every clique C of G(X) containing w is homogeneous in H(X). The term "homogeneous" is used to note that clique C is included as a subgraph in only one clique of H(X). If the transitive inequality w.r.t. S does not hold for every nonsynonymous word pair in M(w), then the words in such a pair is not adjacent in H(X). It follows that n(w) is equal to m(w). While if some nonsynonymous word pair in M(w) satisfies the transitive inequality, then the words in that pair share at least one clique in H(X), and hence n(w) is smaller than m(w). Thus, considering that every homogeneous clique corresponds to one sense of a word, w which meets (2-12) is a word with several application areas or "polysemy" in the document sentences. Secondly, suppose that the cliques of G(X) containing w are not homogeneous, it is possible that, by a word or a synonymous word pair in N(w) and not in M(w), n(w) is greater than or equal to m(w). If the cliques having a common characteristic word are considered to be one sense of a word, then word w which meets (2-12) possesses several shades of application or "shifts in application."

From the above consideration, it is shown that a multiple meaning word obtained by the complexities has several application areas in the document sentences (These sentences are used to give the similarity relation R on X). The advantage of this method of specifying multiple meaning words is its simplicity --- the method simply compares the value of n(w) with that of m(w).

In the rest of this section, let us discuss the problem of grouping set X into the clusters for setting word application areas based on the notion of multiple meaning words. Since every word pair in a clique is mutually synonymous, each clique can be regarded as a cluster of synonyms. Such clusters often overlap with each other for the sets including many ambiguous words. Gotlieb and Kumer [24], in this connection, have developed a procedure (called a merging procedure) for combining cliques into diffuse classes. Their method of merging, however, does not assure that the number of nodes of a newly generated graph would not exceed that of the original graph. In graph H(X), on the other hand, there exists the same number of nodes as in G(X). Further it is possible for merging cliques to use the following theorem.

Theorem 2.2. Let $Z=\{x_1, x_2, \cdots, x_k\}$ $(k \leq n)$ be a set of single (non-multiple) meaning words not adjacent in H(X) with each other. Then

$$m(Z) \geq n(Z), \tag{2-13}$$

where m(Z) (or n(Z)) is the total number of cliques of G(X) (H(X)) containing at least one of Z.

Proof. By the property of the meaning, we have that

$$\begin{aligned} m(\{x_i,x_j\}) &= m(x_i) + m(x_j) - m(x_ix_j), \\ n(\{x_i,x_j\}) &= n(x_i) + n(x_j) - n(x_ix_j), \end{aligned} \tag{2-14}$$

for any $x_i, x_j \in Z$, where $m(x_ix_j)$ (or $n(x_ix_j)$) is the number of cliques of G(X) (H(X)) containing both x_i and x_j. If both x_i and x_j are members of Z, then $m(x_i) \geq n(x_i)$, $m(x_j) \geq n(x_j)$ and $n(x_ix_j)=0$. Since, by the previous assumption, any pair of nonadjacent nodes x_i and x_j in H(X) is not adjacent in G(X), we have that $m(x_ix_j)=0$. Thus

$$\begin{aligned} m(\{x_i,x_j\}) &= m(x_i) + m(x_j) \\ &\geq n(x_i) + n(x_j) = n(\{x_i,x_j\}). \end{aligned} \tag{2-15}$$

To complete the proof of Theorem 2.2, suppose that the theorem holds for any subset V ($\neq$Z) of Z. Since any word x in Z ($x \notin V$) is independent of any other word in V, $n(V \cup \{x\})=0$ and $m(V \cup \{x\})=0$. It follows that $m(V \cup \{x\}) \geq n(V \cup \{x\})$, and hence $m(Z) \geq n(Z)$. (This is an equation when $m(Z)=|Z|$.)

The similar theorem holds for a set of single meaning words not adjacent in G(X) with each other. If the meanings of multiple meaning words are distributed among those of single meaning words, then all words in X are included in M(Z). Considering that the single meaning words are the characteristic elements of the clusters, Theorem 2.2 gives the basis for obtaining application areas for every word of X.

Section 2.5 goes on to show the experimental result of selecting a set {w} of multiple meaning words in X based on the complexities specified by the following criteria:

(IV-1) $n(w) \neq 1$ and w is a cut-node[†] of G(M(w)), a subgraph of G(X) whose nodes consists of M(w).

(IV-2) $m(w)>2$ and $n(w) \geq m(w)$.

2.5 Computational Experiments

The sample corpus X for the computational construction of a dictionary consists of about 220 verbs selected from ABSTRACTs of 100

[†] A set of cut nodes is found by using connected component and fundamental cycle generation algorithms (see Appendices A.5 and A.8).

documents in IEEE Transactions. The input to algorithm A is a matrix representation of a fuzzy expression f_R, or a labeled similarity matrix R, of a similarity relation $\mathcal{R}$. A number of similarity measures have been proposed by various authors [2, 7, 24]. Here a set-theoretical measure is defined as:

$$f_R(x,y)=((\tau+3)|M(x)\cap M(y)|)/(|M(x)|+|M(y)|+(\tau+1)|M(x)\cap M(y)|), \quad (2\text{-}16)$$

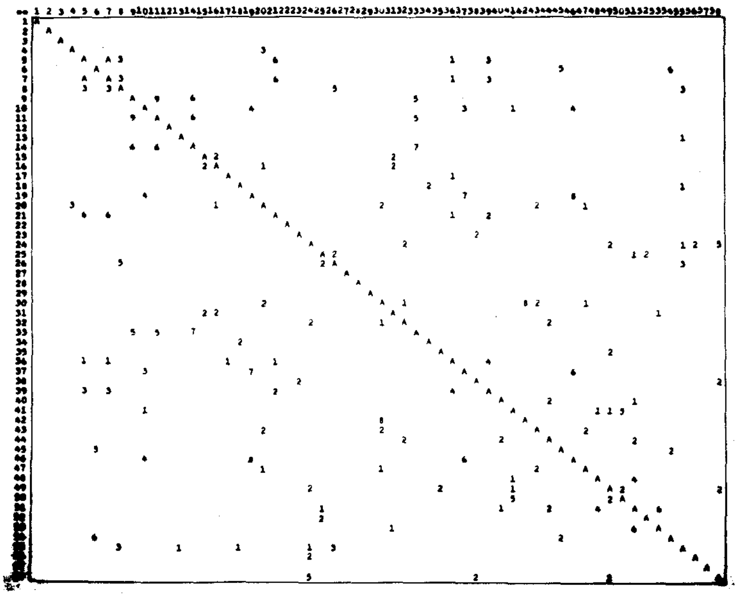
where $|M(x)|$ is the number of words synonymous with x in the thesaurus dictionary [64] (or in the space M(X) for $f_S(x,y)$), and τ, which takes a large value when $|M(x)\cap M(y)|$ is small in comparison with $|M(x)|$ or $|M(y)|$, is a factor for reinforcing $|M(x)\cap M(y)|$ (τ is set to 1 in the experiment). The thesaurus dictionary used for obtaining $f_R(x,y)$ is employed to give an objective standard for evaluating the computationally generated dictionary. (The similarity measure computed based on the word usage experiences is presented in the next chapter.)

The efficiency of the proposed method of constructing a dictionary can be measured by the retrieval factor for the semantic entities, i.e., synonyms, antonyms, multiple meaning words and homonyms, found in the above thesaurus dictionary. Since, however, the codomain of the similarity measure is [0, 1], synonyms and antonyms are not distinguishable by the computational approach [36]. Further homonymy is regarded as the same concept as multiple meaning without the support of a context. The measures, recall R_σ (or R_ρ) and precision P_σ (P_ρ) factors [50], of retrieval effectiveness are then utilized for evaluating the retrieved synonyms (multiple meaning words).

The previously proposed graph-theoretical methods have used the threshold level (called method B) [2, 7, 24] or the nearest neighbor rule (called method C) [33, 70] to produce a graph from the similarity matrix. In method B, a line of the graph corresponds to a synonymous word pair whose similarity value is not smaller than the threshold value T, and in method C, it corresponds to a word pair, one of which belongs to N nearest-neighbors of the other. Once the graph is produced, it is possible to find multiple meaning words based on criteria (IV-1) and (IV-2). A pseudo parameter S_L, to exclude unnecessary similarity by setting any value smaller than the value of S_L to 0, is employed in algorithm A for the comparative studies with the others.

Fig. 2.2(b) shows the result of rearranging the similarity matrix in Fig. 2.2(a) by algorithm TDSO (see Appendix A.2), which is formulated for the compression of the similarity matrix R. The rearranged matrix can be manipulated by subdividing it into the smaller matrices than the original one.

The modified version of TDSO, where Step 2 of procedure ODSO in TDSO is replaced by "If $r_{L(i)L(s)}$ is not smaller than the value of T, then exchange the contents of L(s) and L(t), and add 1 to t," can be used to classify the word set X into the subclasses in which in-class words are connected by a path of lines having similarity values not smaller than T. The clustering result of X is shown in Fig. 2.3, where M_c is the maximal size of clusters, and N_c is the total number of clusters except for 1-element clusters. It is seen that the variances of



1. absorb 2. abstract 3. accelerate 4. accept
5. accomplish 6. accord 7. achieve 8. act
9. adapt 10. add 11. adjust 12. aim 13. allow
14. alter 15. appeal ...

* $R(i,j)=k \ (1 \leq k \leq 10(=A)) \leftrightarrow k \leq 10 f_R(i,j) < k+1.$

Fig. 2.2(a). One part of similarity matrix R.

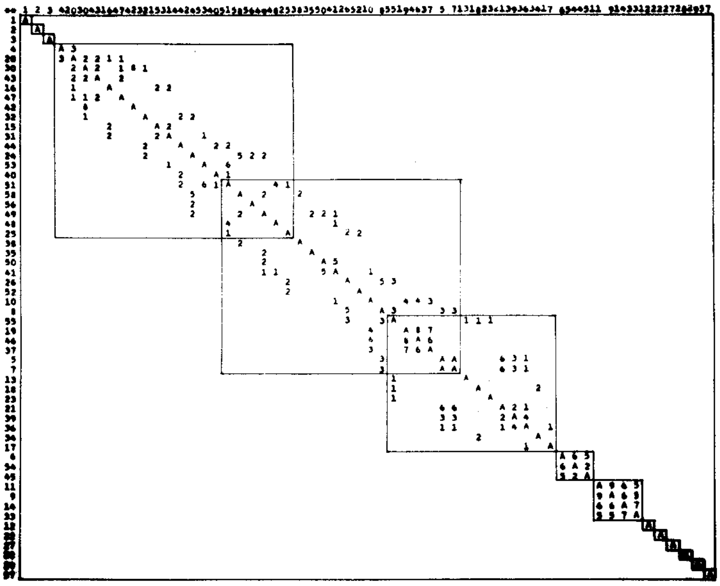


Fig. 2.2(b). Rearranged similarity matrix.

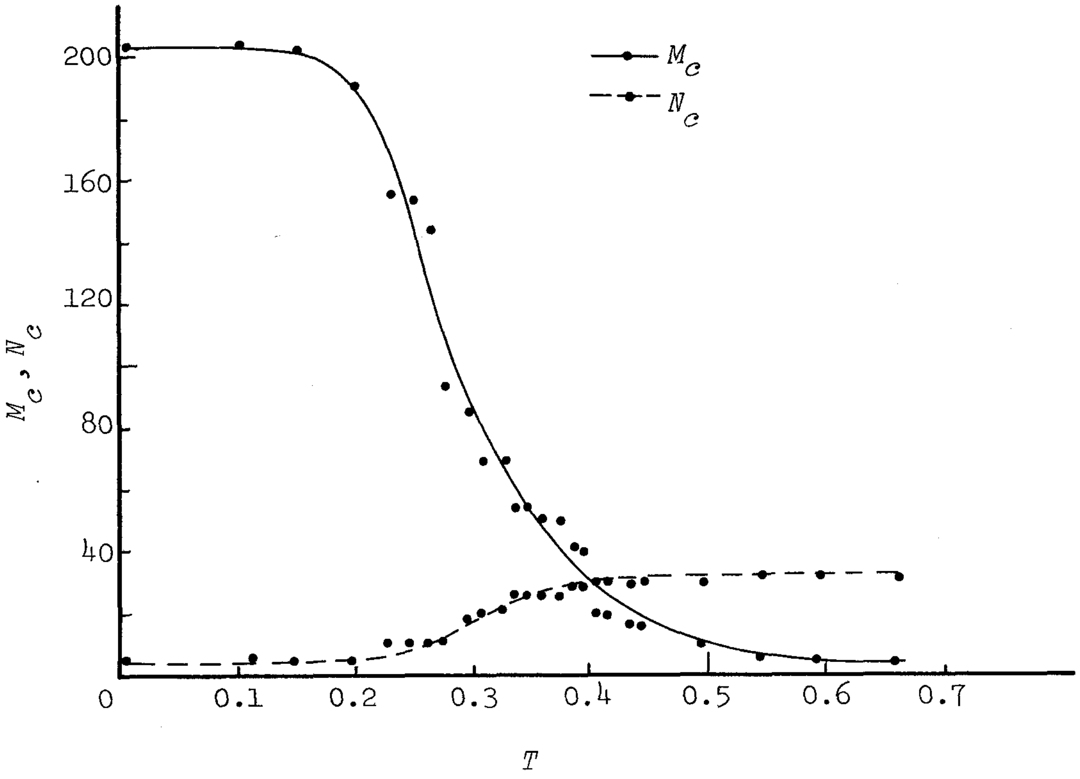


Fig. 2.3. Results of clustering by TDSO.

the values M_c and N_c for T in (0, 0.15) and (0.4, 1.0] are relatively small. This is an indication that the parameter value in those regions has little influence on retrieving synonyms and multiple meaning words.

Table 2.1 presents the timing estimates for methods A and B. (The cycle number for algorithm A is the number of executions of Steps 2, 3 and 4, and for method B the number of matrix products required until any connected pair satisfies the transitive inequality.) The cycle number of algorithm A is smaller than that of method B. This means that in algorithm A the similarity matrix can be manipulated by subdividing it in a smaller scale.

Table 2.2 illustrates some of correctly retrieved, incorrectly retrieved and non-retrieved correct multiple meaning words (mmw) followed by their relevant synonyms. Correctly retrieved multiple meaning word w has its two application areas each of which corresponds to the cliques connected by a single meaning word chain in M(w). The word *bound*, where m(*bound*)=n(*bound*)=2, is found by (IV-1). The mutually synonymous multiple meaning words *define* and *establish* are selected by (IV-2). When the adjective form (*-ing*) of a verb is considered as a meaning element, the homonym *close* is correctly extracted by (IV-1) as a multiple meaning word. The retrieval effectiveness for synonyms and multiple meaning words w.r.t. various values of S_L, T and N is given in Tables 2.3, 2.4 and 2.5. The behavior of algorithm A (product × is employed in place of *) is generally superior to that of the others. The low value of P_σ for method B indicates that simialrity does not

Table 2.1. Cycle numbers before convergence.

method	Algorithm A				Method B			
operation *	×		min		×		min	
sim. matrix $S_{L,T}$	R	S	R	S	R	S	R	S
0.00	1	2	3	4	7	6	11	6
0.1	1	2	3	4	7	6	11	8
0.16	1	1	3	3	7	6	10	12
0.2	1	1	3	3	7	6	14	13

* Every element indicates the maximal cycle number for 7 small matrices of size 58×58 words which are obtained by subdividing similarity matrix R of size 220×220 words.

Table 2.2. Multiple meaning words and synonyms (algorithm A (*=×, S_L=0.1)).

Cr. mmw	Synonyms†
bound:	1. limit restrict 2. enclose include
calculate:	1. compute estimate 2. suppose assume expect conceive
close:	1. conclude terminate complete 2. approach(ing) reach(ing)
define:	1. explain describe formulate illustrate 2. fix establish base constitute
establish:	1. fix constitute define base 2. prove demonstrate show represent
extend:	1. stretch increase magnify 2. give present supply furnish provide deal
recognize:	1. know identify 2. allow accept receive
In. mmw	**Synonyms**
estimate:	calculate compute rank regard decide
indicate:	show demonstrate represent denote express suggest present imply
present:	give offer extend indicate deliver express
restrict:	limit bound control contain
Nr. mmw	**Synonyms**
deliver:	1. give provide furnish 2. express present
express:	1. present offer deliver 2. indicate
make:	1. form compose construct generate constitute 2. cause

Table 2.3(a). Evaluation of structure concepts (algorithm A (*=×)).

S_L	R_σ	P_σ	R_ρ	P_ρ
0.00	0.92	0.77	0.71	0.41
0.1	0.91	0.77	0.71	0.49
0.16	0.90	0.81	0.44	0.54
0.2	0.89	0.84	0.32	0.69
0.3	0.88	0.91	0.06	...
0.4	0.85	0.95	0.03	...

Table 2.3(b). Evaluation of structure concepts (algorithm A (*=*min*)).

S_L	R_σ	P_σ	R_ρ	P_ρ
0.00	0.84	0.79	0.68	0.41
0.1	0.84	0.80	0.65	0.47
0.16	0.84	0.83	0.18	0.46
0.2	0.82	0.86	0.09	0.50
0.3	0.82	0.92	0.06	...
0.4	0.80	0.96	0.03	...

Table 2.4. Evaluation of structure concepts (method B).

T	R_σ	P_σ	R_ρ	P_ρ
0.00	0.92	0.73	0.77	0.34
0.1	0.92	0.74	0.53	0.42
0.16	0.90	0.79	0.35	0.44
0.2	0.89	0.83	0.18	0.43
0.3	0.88	0.90	0.09	...
0.4	0.86	0.95	0.03	...

Table 2.5. Evaluation of structure concepts (method C).

N	R_σ	P_σ	R_ρ	P_ρ
9	0.91	0.77	0.68	0.38
8	0.90	0.79	0.53	0.38
7	0.89	0.80	0.41	0.41
6	0.88	0.81	0.47	0.43
4	0.78	0.86	0.38	0.48
2	0.74	0.93	0.21	...

always represent the degree of synonymy between words. That is, the notion of ambiguity must be taken into account for the selection of synonym clusters, i.e., the semantic processing units. The variance of P_ρ (and R_ρ) for S_L in (0, 0.1] is small in Table 2.3(a) as was predicted in Fig. 2.3. The low value of P_ρ for all cases is caused by the words which are synonymous with correctly retrieved multiple meaning words, and are regarded as single meaning words in the thesaurus dictionary. If P_ρ is measured by assuming these words to be the correctly retrieved multiple meaning words, it becomes (0.72, 0.80, 0.83, 1.0) for S_L=(0.00, 0.1, 0.16, 0.2) in Table 2.3(a). From the above consideration, the most satisfactory dictionary for natural language text processing may be obtainable when graph G(X) is generated based on algorithm A.

An efficient procedure for updating the graph according to the changes of similarity relations between words will be formulated by removing the altered similarity values into a small matrix (TDSO can be used for this purpose).

2.6 Conclusions

A graph-theoretical method of constructing a dictionary, given the usage similarity between every two words, was discussed for the purpose of contextual processing of natural language texts. The specification of synonym classes and ambiguous words, which was done by algorithm A, is needed for the construction of such a dictionary. Further the notion of multiple meaning words was introduced using the cliques of a synonym graph to treat computationally the irregularities of semantic relations. The multiple meaning word is also disambigulated by fixing one of its application areas in a given sentence.

The dictionary construction operation was examined on a set of verbs extracted from ABSTRACTs of 100 documents in IEEE Transactions and the results were compared with those of previously proposed graph-theoretical techniques. It was shown that the proposed method was superior to the others. This is due to the appropriate specification or formulation of ambiguity as well as syonoymy in natural language words.

In the next chapter, a method of computing the similarity between two words is discussed based on the exchangeability over the predicate network of the document sentences. The predicate network is used to decide the application area of the word in a given sentence as well. Another treatment of ambiguity will be shown in Chapter 4, where the document data including ambiguous elements is organized into a structured file for the fast retrieval of the relevant documents to the information queries.

CHAPTER 3

Contextual Text Processing for Automatic Information Retrieval

An approach to the automatic content analysis of textual data is theoretically and experimentally discussed using a predicate network of document sentences. The sentences in the predicate network are expressed by the predicate patterns and the word classes to which the word application areas are attached. The formation of the network is an iterative procedure such that predicate patterns for relating words are settled by using word application areas, and word application areas are obtained by grouping the words having similar usages in the predicate network at the previous iteration. Several word grouping criteria are examined comparatively by constructing a predicate network for the sentences of scientific documents.

3.1 Introduction

The formulation of representing information contents of a document based on the semantic analysis of natural language has been recognized to be important for the design of an automatic information retrieval system. Some of the advantages of natural language information retrieval are the availability of coping with a large user population of nonspecialists, the flexibility of expressing the subject themes of various documents, and the extensity for handling the changes of a document collection for future use. It is also important for obtaining the high recall system performance without reducing the precision to distinguish the substantial differences between the subject areas of the documents.

The establishment of an *automatic* information retrieval system, regardless of whether it employs a keyword indexing schedule or a relevancy computation process, should start with providing a contextual analysis method of natural language document sentences. The procedure of constructing a dictionary for regulating the ambiguity or the complexity of natural language was discussed in the previous chapter, where the similarity between two words is assumed to be given. Here the problem of computational treatment of document sentences is logically and theoretically analyzed, and a predicate network for giving the similarity between two sentences is proposed.

The semantic analysis of natural language sentences has mainly been studied in the field of artificial intelligence (AI) as a model of the human psychological process of inference and deduction for language understanding and translation. Schank [54, 55] and Quillian [41], in this connection, have set up a conceptual dependency or semantic network representation theory for a meaning understanding of natural language sentences. (The similar studies were seen in [6, 10, 17, 22, 53, 57, 63].) They have been conducting the studies by posing manually settled semantic primitives from a conceptual level and allowable inference process between these primitives. Such approaches to computational semantics of natural language seemed not to be satisfactory, when applied to the IR field, to deal with a large amount of document data including complicated entities.

Quine [42] emphasized, as shown in Chapter 2, that the study of semantic part of languages comes not to appeal to meaning but to concern with synonymy in significant word sequences. That is to say, the "meaning" of a word is fixed by the two contextual definitions "alike in meaning (or synonymy)" and "having meaning (or significant)." The present chapter develops, using a dictionary which specifies the words similar in usages, a method of resolving sentences into a predicate network for a computational approach to the semantic processing of textual data.

It is shown in Section 3.3 that a predicate P of any sentence should computationally be treated as an extensional entity except for considering the symbol "p" is a meta expression by itself. A dictionary for setting word application areas and for establishing predicate patterns (the predicate pattern provides semantic relations between the words) is generated in Section 3.4 by rectifying recursively the incorrect setting of application areas in the given sentence examples. Spärck-Jones [58] has set up a standard for classifying index terms by noting that two words may be considered semantically equivalent or synonymous if in a given context the words are interchangeable without changing *meaning*. However, the resulting clusters of index terms are unsatisfactory to process contextually the document data by its poor definition of "without changing meaning." The grouping criterion proposed here is such that a set of the words interchangeable with a single meaning word without changing surrounding situations in a predicate network provides one of the word clusters for specifying word usages. The validity of the proposed method is shown in Section 3.5 by the experimental construction of a predicate network for the sentences of scientific documents.

3.2 Basic Assumptions

The analysis of a document collection in an information retrieval environment should be proceeded so as to establish automatically a conceptual or network representation of the document sentences. Such a network, called a predicate network, is used for establishing relevancy between two sentences or a sentence and an incomming query.

A method of analyzing the document sentences, which are considered intuitively to be "significant" in a document collection, is here discussed by taking the following three assumptions for granted. These are intended for the automatic processing of natural language document data.

1) Any simple sentence S can be decomposed into the basic components, several terms and a predicate for relating these terms.

2) The content of any S can be interpreted by relating the components of S to those of other sentences, i.e., establishing a predicate network for the document sentences.

3) The "interchangeability" for the basic components can be tested based on their contextual positions in the network.

The basic components in the first assumption are commonly specified by the information for the part of speech, idiomatic phrases, attribute indications, etc. The second assumption says that the subject content of S is seized within a predicate network, and the third one denoting interchangeability gives the standard for obtaining similarity between the basic components. The interchangeability here is formulated based on the fact that there are fundamental features of the way of conceptualizing language contents and of specifying synonym word classes. That is, two components observed in the similar relational situation in the predicate network differ only in the symbolic forms and not in the conceptual indications.

In the following sections, a method of obtaining predicate network for natural language document sentences is discussed based on those three assumptions.

3.3 Predicate Network

The structural decomposition of natural language document sentences is needed for the analytic understanding of the document contents and the easy judging of semantic relevancy between the two documents. In the analysis of sentences, a phrase in the term showing a fixed concept should be treated as a single indivisible word. Thus by assumption 1 the predicative expression of any sentence is defined as:

Definition 3.1. Let P and $\chi = (x_1, x_2, \cdots, x_r)$, $r \geq 1$, be a predicate and a word vector (shortly word), respectively. Then a simple sentence S is expressed by

$$S = P\chi. \qquad (3\text{-}1)$$

(If the number of argument in χ is 1, then P is often called an attribute; otherwise called a relation-in-intension interconnecting all x_i, $1 \leq i \leq r$.)

An abstract or concrete singular term is eligible for every x_i, and a general term for P. The following two attributes (in Russell's circumflexed notation) for a simple sentence S (=Pχ) can be derived by the definition.

$$P\hat{\chi}, \qquad (3\text{-}2)$$

and

$$\hat{P}\chi. \qquad (3\text{-}3)$$

Expressions (3-2) and (3-3), respectively, mean the attributes "having a relation P" and "relating the words in χ." The content of S therefore is interpreted by specifying these attributes such that predicates and words having similar usage experiences are related with each other in a network form. The attribute $P\hat{\chi}$ has been intensionally realized at the AI field in conjunction with a modeling of human psychological process for natural language understanding and translation. The main defect of such a program is that the manually decided (certainly, *artificially*-programmed *intelligence*) semantic primitives and semantic relations between them are too complicated to use for the processing of a large amount of document data.

The attribute $P\hat{\chi}$ can be explained extensionally, in addition to intensionally, as a class such that $\{\chi: P\chi\}$. Extensionality is what separates classes from attributes (i.e., classes are considered to be identical when their members are identical). But attributes are unlike mere classes in the capability of distincting from one another even when relating extensively the same things. The distinction between them is referred to as Russell's theory of types [43]. Intensional expression of P is computationally obtained only when it appears in other sentences as the general term for P (The general term for P differs from P in *order* or *type*).

Proposition 3.1. Let P be a predicate in a significant simple sentence S. Then P could not be intensionally explained except for the following two special cases.

(1) Symbol "P" itself shows an attribute of P.

(2) General term for P appears as the singular term in another relation or attribute.

We can conclude by Proposition 3.1 that P should be treated as an extensional entity for the computational construction of a predicate network. The similarities h and f of two predicates P and Q, and two words x and y are given by the concretion of the abstracts $\mathcal{V} = \{(P,Q)$: P and Q give the similar relation to all words$\}$ and $\mathcal{W} = \{(x,y)$: any predicate relates x and y to the other words similarly$\}$ such that

$$h_V(P,Q) \qquad \text{for} \qquad P\mathcal{V}Q, \tag{3-4}$$

$$f_W(x,y) \qquad \text{for} \qquad x\mathcal{W}y. \tag{3-5}$$

Hereafter $P\hat{x}$ and $\hat{P}x$, respectively, are called a predicate pattern for P and word usage examples (or experiences) for x. Any predicate pattern is settled by relating the words of the predicate to the word classes which those words belong to. Thus the contextual expression of S is obtained by relating (i) the indivisible word to its application areas, and (ii) the predicate to its predicate pattern. Incomming sentences are thus predicatively analyzed by using a word usage dictionary and a set of predicate patterns. The dictionary is, as shown in the previous chapter, generated by grouping the interchangeable (see assumption 3 word pairs in a predicate network. Predicate pattern is used to predict the adequate individuals for the predicate arguments as well. Thus both the predicate pattern and the dictionary are needed to disambigulate the word application areas in a given sentence, to find the correct references of various pronouns, to decide the dependency form of modifications, and to limit the number of arguments for a predicate. When a word application area is expressed in the form of a storage location which is connected with χ by a pointer (called an application pointer) from χ to the locations, then $P\hat{\chi}$ is obtained simply by pursuing pointers of χ in the positive directions and $\hat{P}\chi$ in the opposite directions. A predicate network thus is constructed by giving a set of application pointers to the predicate arguments and the words.

The predicative analysis result is exemplified in Table 3.1 for the sentences drawn from several scientific documents. These ideas about simple sentences can be used to analyze complex sentences and documents by considering the basic components are simple sentences, modification style, type of conjunction, several kinds of pronouns, etc.

Table 3.1. Sentence examples showing word usage experiences.

No.	Sentences
1	A class of *pictures* is composed of line-like elements, specifically, digitized *bubble-chamber photographs.*
2	Shaded regions in a class of *pictures* of human faces, *photographs* of cloud formation.
3	Piecewise-linear classification is replaced by *edge* detecting preprocessings.
4	*Edges* between different texture regions are detected in a composite output.
5	Texture *edges* in digitized *pictures.*
6	Digitized *picture* processing-operations are discussed with Gestalt psychologists' laws of pictorial pattern-recognition.
7	Small regions of the film are scanned automatically in a special-purpose *bubble-chamber* film measuring-device.
8	*DAPR* is a digital automatic pattern-recognition system.
9	*DAPR* is able to measure *bubble-chamber* films from *photographic* development-process.
10	A digital abstraction of the information is contained in the *bubble-chamber* film.
11	A sequential-decision model selects *feature* subsets in pattern-recognition.
12	A character-recognition experiment demonstrates the feasibility of the *feature* selection strategies.
13	A *graph-theoretical* cluster detection is applied to the selection of a good *feature* space for pattern-recognition.

* Predicates and words for setting application pointers are written in block letters and italics, respectively.

3.4 Predicate Pattern and Dictionary Generation

The predicate network for an automatic analysis of document sentences is constructed by establishing predicate patterns and word application areas for the predicates and the words of those sentences. An automatic method (and also a manual one) of obtaining intensional expressions for the predicates is hard to establish, because in most cases the singular term for a predicate would not be specified computationally.

Here a lexicographic problem of obtaining application pointers and predicate patterns is discussed. A predicate pattern, one of the semantic primitives for the predicate network, is used to decide how the words are conceptually related with each other. Any application pointer corresponding to a word class is given by clustering synonym classes. Synonyms are extracted by executing algorithm A on the similarity matrix of the word set. The similarity measure is here computed based on the interchangeability such that two words are similar iff they are interchangeable in some contexts of a predicate network without changing the surrounding connections. The interchangeability is then defined as:

Definition 3.2 (Law of Interchangeability): Let $\mathcal{X}$ and $\mathcal{Y}$ be word vectors for predicates $\{P\}$ and $\{Q\}$ such that $P\mathcal{X}$ and $Q\mathcal{Y}$ ($P\in\{P\}$, $Q\in\{Q\}$), respectively. Then the interchangeability of the two words x in $\mathcal{X}$ and y in $\mathcal{Y}$ is the similarity f(x,y) between the two situations which are connected to x or y via $\{P\}$ or $\{Q\}$ in the predicate network.

Definition 3.2 is circular, i.e., the interchangeability for defining the class of synonym words and the word application areas are computed by using a predicate network, which is settled by replacing the arguments of the predicate patterns with the application pointers. An iterative procedure, therefore, is employed for generating a predicate network, where intuitively correct applications are firstly given as an input data, and the other correct ones are successively decided based on the pointers and patterns at the previous iteration. (Hereafter every iteration for setting application pointers is called by the term "cycle.")

Let us next discuss about which word set gives a right cluster. Though the interchangeability is considered to give an appropriate approximation for synonymy, no right word cluster at cycle i (≥ 1) is unfortunately known beforehand. If the true clusters are obtained at some cycle k (≥ 0), then we can believe that the correct clusters at cycle i ($\geq k$) are deducible iteratively. Let us now examine the relationship between two similarity values $f_i(x,y)$ and $f_{i+1}(x,y)$ for pair (x,y) at cycles i and i+1. If any word cluster at cycle i has no word pair with different areas, then the application pointer set S_i at cycle i is included in the pointer set S_{i+1} at cycle i+1. The similarity value $f_i(x,y)$ obtained by using S_i is then not greater than that of $f_{i+1}(x,y)$. Conversely, if $f_{i+1}(x,y) \geq f_i(x,y)$ for all (x,y), then any area at i+1 is considered to include no ambiguous word. Thus the clustering strategy is decided so that the similarity value

$f_{i+1}(x,y)$ will become not smaller than that of $f_i(x,y)$ and the resulting clusters include no pair of words having different application areas.

[Algorithm D] (Dictionary Generation)

In: Initial (i=1) predicate network, i.e., initial word application pointers and predicate patterns.

Out: All the application pointers.

Note: The following steps are formulated for cycle i (>1).

Procedure:

Step 1. Compute the similarity value f(x,y) for words x and y in the predicate network.

Step 2. By algorithm A, extract all the synonym classes.

Step 3. Cluster the synonym classes for setting application pointers so that every application area will include at least one characteristic word (see Theorem 2.2). Set $f_i(x,y)$ to $\max(f(x,y), f_{i-1}(x,y))$ to note that (x,y) satisfying $f_i(x,y) \geq f_{i-1}(x,y)$ is a correctly interchangeable word pair. If $f(x,y) = f_{i-1}(x,y)$ for any pair (x,y), then terminate the algorithm.

Step 4. Update the predicate patterns based on the newly obtained pointers, and go to Step 1 to enter the next cycle.

Algorithm D gradually selects the characteristic synonym classes by examining the correctness of the settled application pointers through the word usage examples in the predicate network. The updating of predicate patterns is intended to establish the right interrelationships between the words. Though more than one application pointers are also able to be attached to any argument of the predicate pattern, the disambigulation of the word could be attained contextually by the preferred inference process [63]. The termination property of algorithm D is proved by Theorem 2.1 and increasing property of f_i $(i \geq 1)$. After the termination of this algorithm, the fixed set of application pointers are obtained.

In the next section, computational experiments will proceed by taking word stems as one of the input application pointers. Several criteria for clustering synonym classes are employed for the performance evaluation of algorithm D.

3.5 Computational Experiments

An experimental system in Fig. 3.1 operated on a set of sentences of scientific documents is designed for the construction of a predicate network. The core of this experiment consists of the predicative sentence analysis and the grouping of synonym word classes.

The sample corpus consists of about 200 sentences extracted from the ABSTRACTs of 40 documents in IEEE Transactions. Preprocessing of the sentences is such that any pronoun is replaced by its designating noun, any verb is changed into its present v_o or participle v_p form, and complex or compound sentences are decomposed into a set of simple sentences.

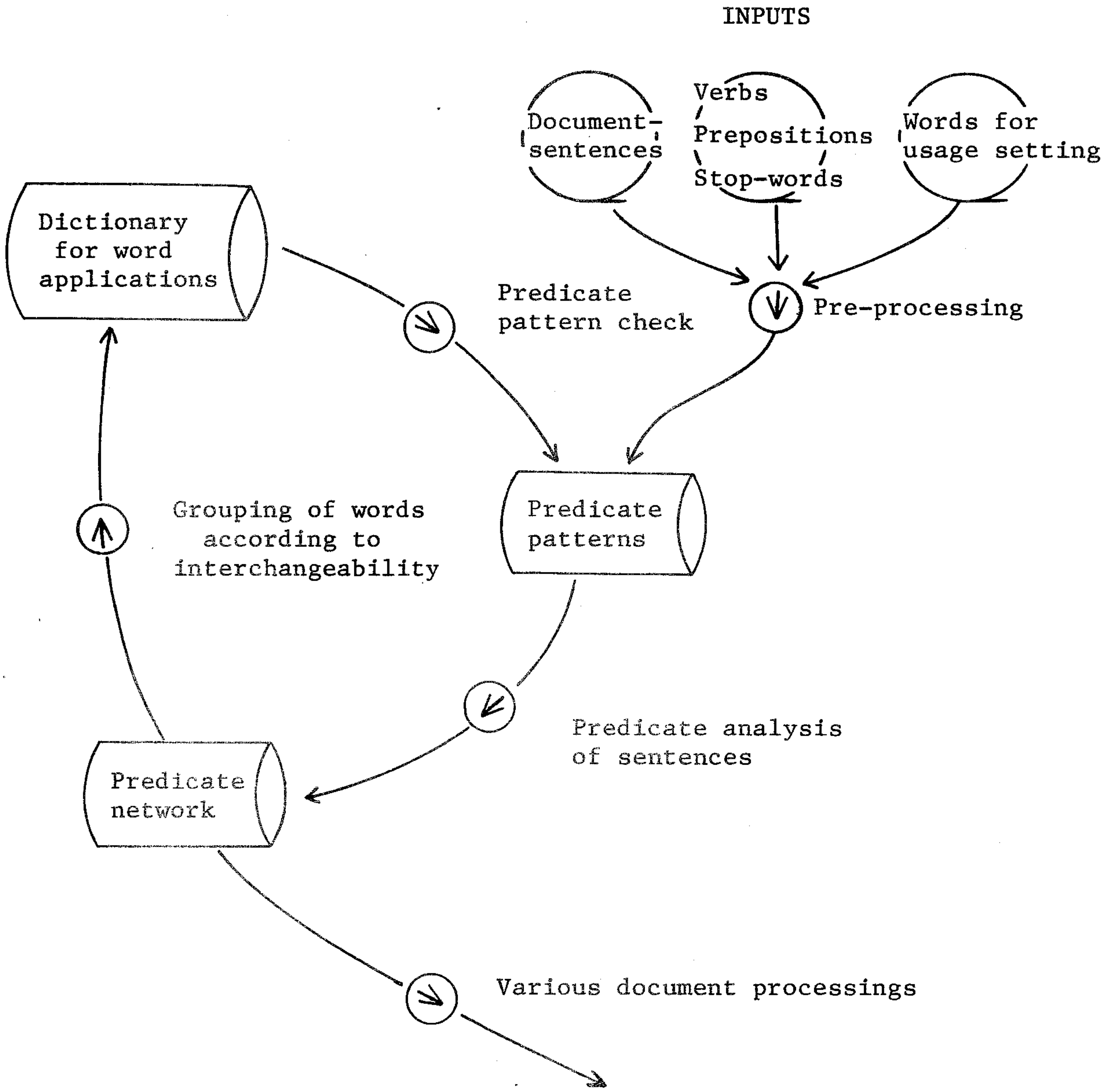


Fig. 3.1. Predicate network generation steps.

Further auxially verbs are deleted, and nouns (including several adjectives) are replaced by their word stems.

Any simple sentence is predicatively analyzed based on verbs (v_o and v_p) and prepositions [27, 63]. The general form of a predicate is considered to be an n-place predicate P_n having n terms ($n \geq 1$). The specification of 3-, 4-, etc., predicates, however, needs *a priori* knowledge of semantic relations between the words in the terms. The predicative sentence analysis proposed here treats any predicate as the combination of 2-place predicates of the forms (i) v_o (or v_p) + [preposition]† (ii) is + v_p (including adjective) + [preposition], or (iii) preposition only. Any predicate pattern (and necessarily a predicate network) is then obtained simply by giving application pointers to the two arguments of the predicate. Some of the 2-place predicates are shown in Table 3.2. Other input data, the stop-words, are used to prevent the inessential setting of associations between words. The initial set of application pointers is a collection of the word stems to which the final pointer setting is intended.

Application pointers at cycle i (≥ 1) is obtained by grouping the words according to the usage closeness in the predicate network at cycle i-1. The closeness between word w_1 in term t_1 and word w_2 in term t_2 is given by Definition 3.2 as follows: Let us denote by $\{P(t_1,u)\}$ or $\{Q(t_2,v)\}$, the predicative expressions of sentences

† [preposition] means that the preposition can be eliminated.

Table 3.2. Examples of 2-place predicates.

according-to
aimed (at)
applied (to)
associated (with)
assumed (to)
based (on)
compared (with)
composed (of)
concerned (with)
contained (in)
corresponding (to)
defined (by, to)
demonstrated (for)
derived (from)
detected (in)
described (by, for)
employed (for, in)
established (for)
expected (for)
found (for)
generated (by, for)
given (for, to)
ignored (in)
illustrated (by)
introduced (by, to)
make-use-of
obtained (by)
presented (for, to)
processed (by, with)
produced (for)
searched (for)
tested (on)
take-into-account
take-place
used (for, to)
with-respect-to

including w_1 or w_2, where P and Q are 2-place predicates, and u and v are terms. The situations, where w_1 and w_2 are used in the predicate network, are given by $\{(P,u)\}$ $(=U_1)$ and $\{(Q,v)\}$ $(=U_2)$, respectively. The similarity $f(w_1,w_2)$ for words w_1 and w_2 is then computed by modifying equation (2-16) as

$$f(w_1,w_2)=(\tau+3)\cdot\min(\nu(U_1|U_2),\nu(U_2|U_1))/ \\ (\nu(U_1)+\nu(U_2)+(\tau+1)\cdot\max(\nu(U_1|U_2),\nu(U_2|U_1))), \tag{3-6}$$

where $\nu(U_1|U_2)$ indicates the sum of the matches of U_1 for U_2. The degree of the match of (P,u) for U_2 takes one of the values {1, 1/2, 0} according to the following three cases: For some $(Q,v)\in U_2$, (i) P=Q, and $A(u)\cap A(v)\neq\emptyset$ (A(u) means a set of application pointers of the words in u), (ii) $P\neq Q$ and $A(u)\cap A(v)\neq\emptyset$, and (iii) $A(u)\cap A(v)=\emptyset$.

A word set X required for setting application pointers consists of about 50 words extracted from the input sentences. The similarity matrix for these words is arranged similarly in Section 2.5 by TDSO and divided into 4 small matrices. The extraction operation of synonyms by algorithm A and the grouping operation of these extracted synonyms by algorithm D are executed in turn. Also algorithm A constructs a synonym graph G(X) of X.

Five criteria SL, ML, SW, MW and CQ for clustering word set X are formulated and used in Step 3 of algorithm D.

SL: All the words in the cliques, each of which has a member in common with some other clique, are given an application pointer.

ML: All the words in the cliques, each of which has members in common with some other clique except one member, are given a pointer.

SW: 1) All the words in the cliques including at least one characteristic word are given a pointer (single meaning words belonging to a small number of cliques are selected as candidates for the characteristic words). 2) Every word with more than one application areas, i.e., multiple meaning word, is given no pointer except the word stem pointer.

MW: 1) All the words in the cliques including at least one characteristic word are given a pointer. 2) Every multiple meaning word is given the same pointer as the word closest to it.

CQ: All the words in a clique is given a pointer.

The result of algorithm D for these criteria at cycle $i\in\{1, 2, \cdots, 6\}$ is shown in Table 3.3 (the significance level S_L of algorithm A is fixed to 0.15). Every element of this table means $(|A_i|/|X|, |B_i|/|X|)$, where A_i={a set of application areas including more than one word at cycle i} and $B_i=\{x: x\in X, A_i(x)=A_{i-1}(x)\}$.

The validity of the employed criterion is measured by the variance of $|B_i|/|X|$, so algorithm D[†] is executed by excluding the statement "otherwise, set $f_i(x,y)$ to $\max(f(x,y),f_{i-1}(x,y))$ to $\cdots$." in Step 2. It is seen that incorrect pointers are initially settled for a large value of τ.

[†] Algorithm D terminates when the present application pointers are the same as the previous ones.

Table 3.3. Comparisons of clustering abilities for setting application pointers (τ=-1).

cycle i / criterion	1		2		3		4		5	
	a	b	a	b	a	b	a	b	a	b
SL	0.21	0.32	0.13	0.74	0.13	1.00	*		*	
ML	0.31	0.40	0.27	0.64	0.23	0.82	0.23	0.94	0.23	0.96
SW	0.35	0.68	0.33	0.88	0.33	0.96	*		*	
MW	0.35	0.46	0.31	0.72	0.27	0.78	0.29	0.86	0.29	0.96
CQ	0.56	0.48	0.56	0.44	0.56	0.78	0.56	1.00	*	

* Variable a (and b) shows the ratio of the number of application areas consisting of more than one word at cycle i (the number of application areas commonly occured in both cycles i-1 and i) to the total number of words in X.

The criteria SL and ML, often used in graph-theoretical clustering algorithms, are intended to collect the words linked by ambiguous words into one cluster. The linkage cluster analysis, however, has a well known defect that two words having different application areas are given more inclusive application pointers than required. Thus the word clusters not suitably subdivided are obtained by the linkage criteria SL and ML.

Other criteria SW and MW are formulated based on Theorem 2.2, where the meanings of the characteristic words correspond to the correct application areas. It is seen in Tables 3.3 and 3.4 that SW gives the more preferable program to MW in the way of obtaining the correct application areas. This is not unexpected, since only the classes including no word pairs with different usage experiences are selected at every cycle, and the words with different application areas are decided to be multiple meaning words. When many pointers are attached to those multiple meaning words (which cause multiplicity via the usage examples in the document sentences), the settling of pointers becomes unstable by their ambiguous usages. According to the last criterion CQ, the resulting word groups are not well clustered.

The larger size of clusters is obtained as the value of the similarity increases. Table 3.4 shows the clustering result obtained by criteria SW and MW for various values of τ and i. Algorithm D employing MW requires many cycles for fixing application pointers for the small value of τ.

Table 3.4. Clustering statistics for various values of τ (criteria SW and MW).

cycle i / τ		1		2		3		4		5	
		a	b	a	b	a	b	a	b	a	b
-2	SW	0.66	0.82	0.66	0.96	*		*		*	
	MW	0.65	0.74	0.60	0.92	0.58	0.90	0.60	0.90	0.60	0.98
-1	SW	0.35	0.68	0.33	0.88	0.33	0.96	*		*	
	MW	0.35	0.46	0.31	0.72	0.27	0.78	0.29	0.86	0.29	0.96
0	SW	0.31	0.68	0.31	0.86	0.31	0.98	*		*	
	MW	0.29	0.28	0.27	0.88	0.25	0.96	*		*	
1	SW	0.25	0.64	0.25	0.92	0.25	0.98	*		*	
	MW	0.29	0.30	0.27	0.84	0.27	1.00	*		*	

Tables 3.5(a) and 3.5(b) show the transient application areas settled by criterion SW. Table 3.5(a) tells that the application area for the word *DAPR* is the same as that of *bubble*, and different from that of *edge*. By referring to Table 3.1 exemplifying the usage experiences for the words in Table 3.5(a), the attribute for *DAPR* and/or *bubble* in area 1 is explained contextually as "DAPR (which is a pattern-recognition system) for selecting features of a bubble-chamber photograph, one of the digitized pictures." Similarly, the attributes for *edge* in area 3 and *graph* in area 2, respectively, are explained as "Digitized pictures and photographs include texture edges," and "A graph-theoretical method selects features (for pattern-recognition)."

Since the users' requests for the document information are submitted in various conceptual levels, the word application areas for the contextual analysis of document sentences should be hierarchically related such that the areas at the lowest level is used to obtain and disambigulate the specific word usages, and the areas at the higher level to show the more general ones. For example, areas 2 and 3 containing *picture* could be grouped into one general application area to retrieve the documents denoting a method of processing line-like photographs or detecting texture edges as the ones having the subject theme about picture processing. Thus the final pointers settled by SW can be grouped, by a linkage cluster criterion, into a new pointer showing a general areas. It is hard, however, to divide the final areas obtained by SL and/or ML into the subsets for the specific areas. The process for

Table 3.5(a). Application areas versus cycles.

cycle i	0	1	2	3
application areas	1. bubble 2. DAPR 3. photograph 4. feature 5. picture 6. edge 7. graph	1. bubble DAPR photograph* 2. feature 3. photograph* picture edge 4. graph	1. bubble DAPR photograph* feature picture 2. photograph* edge 3. graph	1. bubble DAPR photograph* feature* picture* 2. feature* graph 3. photograph* picture* edge

† The asterisks are used to show multiple meaning words.

Table 3.5(b). Application areas versus cycles.

cycle i	0	1	2
application areas	1. decision 2. correlation 3. weight 4. sequential 5. nonparametric 6. linear 7. discrimination 8. class	1. decision class* 2. weight nonparametric linear* discrimination* class* 3. correlation class* 4. sequential linear* discrimination*	1. decision discrimination* class* 2. weight nonparametric linear discrimination* class* 3. correlation class* 4. sequential discrimination*

setting application areas, therefore, is to specify the synonym classes of the characteristic words, and then to cluster those areas into diffuse classes.

3.6 Conclusions

The method of predicative analysis of natural language was presented, using predicate patterns and word application areas, for the easy relevancy computation between the document sentences. Conventional techniques of text analysis in IR areas have exploited keyword or class number indexing schedules for the cataloging of the document contents. Then the main problem of automatic document processing is not in the sentence analysis but in the indexing of documents.

The predicative expression of the document sentences was proposed for the contextual processing of the document themes. That is, document sentences are automatically analyzed by examining the usage experiences of predicates and words in the network. Experimental experiences for the construction of a predicate network have revealed that the application areas for every word should be specified by the characteristic single meaning words and the words similar to the characteristic words in usage experiences are grouped into one cluster.

The established predicate network, which formally represents the conceptual relations between sentences, is used for the relevancy computation between two documents. The verbs appeared in the predicate should be treated as both special terms and general terms for the analysis of complex and compound sentences. That is, the intensional explanation and extensional understanding of the predicates will contribute to the future researches of contextual text processing in the design of automatic information retrieval systems.

CHAPTER 4

Structured File Organization for Fast Information Retrieval

An automatic real-time information retrieval system is expected to provide a fast file search algorithm for retrieving the related documents to the various submitted queries. In this chapter, a new method of organizing a file, called a structured file, is discussed based on the linear ordering of a document set. The document set is here arranged according to the similarity between two documents, and is organized in a form of hierarchically networked structured files. Fast file search algorithms which can access to the confined small subsections of the document file are also presented.

4.1 Introduction

The utilization of a computer system for various information processings has become possible by the prevalence of data-base management and computer networking techniques. An information retrieval system working in a real-time operating environment is then expected to process many search queries against a large document file. The information queries about the searching of the document file, which vary with the user population, are classified into the following three basic types:

1) Determine whether or not the document identical to the incomming query is in the file.

2) Obtain all the documents in the file having the closest or the most relevant properties to the query.

3) Obtain the documents in the file showing high relevancy to the query, or arrange these documents in such a way that their similarities to the query are ordered into a decreasing sequence.

The type 1 query commonly occured in constructing symbol tables is answered by conventional techniques of binary search, direct chaining or key hashing methods [4, 5, 37, 38, 44]. On the other hand, it is a typical problem in a document retrieval environment to search a file for the documents which are appropriately related to, but not always identical to, the query. Also, an exhaustive search examining all the stored items from one end to the other can answer such problems. Since, however, the file to be searched is usually very large, it is hard to examine the relevancy of the queries against the entire collection of the stored item.

Salton *et al.* [32, 49, 50, 62], in this connection, have formulated partial file search procedures by constructing groups of related documents. All the accesses to the file were made through the centroids of the document groups. Burkhard *et al.*[11, 56] have devised efficient file search methods for the closest (or the best-match) key to the query by subdividing the given set into subclasses based on the distances on a set of index keys. Saltons' methods, however, do not always find the closest documents to the queries, and Burkhards' methods are only applicable to search the documents whose keys form a disatnce space. Further those methods are inefficient to cope with the type 3 request because of inadequacy of the definition of the term "closeness" or "relevancy." Considering that the given similarity is an index for clustering the documents, relevancy should be measured by a new similarity obtained after grouping the document set.

It is noted that the similarity between two documents would be computed by using a predicate network which regulates the ambiguity of the document sentences. Then the discussion here proceeds firstly by assuming that a fuzzy transitive inequality holds for the given document set. It is possible by a Two-Dimensional Sorting Operation to arrange the documents in a one-dimensional storage space for a hierarchical clustering, and to organize the document set into a structured file so as to answer effectively the types 2 and 3 requests. Next, the discussion about structuring a file is made by not assuming the transitive property of the document set. A new measure, intra-class similarity, is defined to show the relevancy of two documents in a cluster as the minimum line weight of a path connecting these documents in a maximal spanning tree of a similarity weighted graph. All the documents are then grouped into a set of document clusters, each of which constitutes a structured file, based on the newly obtained measure. The entire file is organized by placing the files for document clusters and their abstracted forms on a storage hierarchy [45] in the form of hierarchically networked structured files. The retrieval of the required documents from the storage hierarchy is attained by searching iteratively the structured files at every storage level. Experimentally the proposed methods of file organizing and searching are shown to be useful for answering the queries of the above three types.

4.2 Basic Observation

In this section, several characters of a maximal spannig tree (MST) and a fuzzy equivalence relation are examined from a point of clustering analysis.

The concept of an MST of an undirected finite graph was first used to detect and describe the structure of a point cluster in spark-chamber photographs, and then gradually becomes to be used for various problems in cluster analyses. This is because the MST reflects well the structure of the input data, and is obtained by the computationally simple procedures [70]. Zadeh [69] and Tamura [60] gave independently developed based on a fuzzy equivalence relation a pattern classification method which, as Dunn [15] proved, essentially constructs an MST. One efficient procedure for obtaining an MST is shown in Appendix A.6.

Let us observe the fuzzy classification schema in the light of hierarchical cluster analysis. Consider f satisfying equations (2-2) and (2-3) be a similarity measure defined on two members of a document set D ($|D|=n$). The similarity f leads a reflexive and symmetric relation $\mathcal{R}$ by Definition 2.1. Next donote by G(D) a graph whose node d and weighted line dc correspond to a document d and a similarity value f(d,c) between documents d and c in D. Further let g(d,c) be a concretion with a pair (d,c) of an n-step fuzzy relation $\mathcal{S}$ [60, 69] obtained by the maxmin compositions of relation $\mathcal{R}$.

Since $\mathcal{S}$ satisfies all of fuzzy reflexive, symmetric and transitive laws, $\mathcal{S}$ is an equivalence relation. For each threshold value $\tau\epsilon[0, 1]$, define a new relation S_τ to be a set of document pairs having the similarity value not smaller than τ. Then the following theorem holds [70].

Theorem 4.1. Let C_τ be an equivalence class of S_τ. Then any pair of documents in C_τ is connected in an MST by a unique path of lines whose weights are not smaller than τ.

This theorem states that for $d, c\epsilon C_\tau$ and $e\not\epsilon C_\tau$ the similarity of a pair (d,c) is greater than that of pairs (d,e) and (c,e). Since a class in $\{C_\tau\}$ is expressed by a finite union of nonempty classes $\{C_{\tau'}\}$, $\tau<\tau'$, an m-level hierarchy of clusters is obtained according to a monotone decreasing finite threshold sequence $1=\tau_0>\tau_1>\tau_2>\cdots>\tau_m\geq 0$, $1<m<n$.

The notion of MST is also used for finding maximal capacity routes in a weighted graph. The capacity of a path connecting two nodes in weighted graph G_W is the minimum line weight of the path. Any path in MST of G_W is proved to be one of the paths with the maximal capacity. Thus the intra-class similarity g(d,c) defined as the maximal capacity of the path connecting d and c is identical to the concretion of fuzzy relation S with (d,c).

Hereafter a document set D in which the similarity g(d,c) is given for any pair (d,c) is called a transitive space. Whereas a set D in which the distance for any (d,c) is given by a metric function is called a distance space.

4.3 Document Ordering for Structured File Organization

In an information retrieval environment, where the order of hundreds of thousand of documents may be processed, a development of a fast file search algorithm is an important problem to obtain effectively the related documents to every incomming query. The document collection, therefore, is organized so that the searches could be restricted to the desired subsections of the stored file.

A method of arranging the documents in a one-dimensional storage space is presented from which the relevant documents to any query is effectively retrieved. The discussion here is made by assuming that the given document set D constitutes a transitive space with respect to the similarity measure g. The following algorithm, called MTDSO, can be used to arrange all of D such that

$$
\begin{aligned}
& g(d_i,d_j) \geq g(d_i,d_k), \qquad 1\leq i\leq j\leq k\leq n, \\
\text{or} \quad & g(d_i,d_j) \leq g(d_i,d_k), \qquad 1\leq j\leq k\leq i\leq n,
\end{aligned}
\tag{4-1}
$$

where d_i, d_j and d_k are documents in D, and i, j and k are indices for document ordering.

[Algorithm MTDSO] (Modified TDSO)

In: Assume that all of D are indexed by finite integers 1, 2, ⋯, and n, which are stored in a one-dimensional array L (size n). The similarity $g(d_i,d_j)$ is expressed by an element a_{ij} of matrix A (n×n). Also matrix A is considered to be labeled by L.

Out: Ordering of L which satisfies inequality (4-1).

Note: A variable i is used for indicating a row of matrix A.

Procedure:

Step 1. Do Step 2 for i=1, 2, ⋯, n-1.

Step 2. Rearrange the contents of [L(i+1), L(i+2), ⋯, L(n)] so that a continued part $[a_{L(i)L(i+1)}, a_{L(i)L(i+2)}, \cdots, a_{L(i)L(n)}]$ in row L(i) of A will be ordered into a decreasing sequence. (The row L(i) of A is expressed as $[a_{L(i)L(1)}, a_{L(i)L(2)}, \cdots, a_{L(i)L(n)}]$.)

Step 3. Renumber all the documents by a permutation

$$L(i) \rightarrow i, \tag{4-2}$$

and terminated the algorithm.

$$\begin{bmatrix}
a_{11} & a_{12} & \cdots & & a_{1i} & a_{1,i+1} \\
 & a_{22} & \cdots & & a_{2i} & a_{2,i+1} \\
 & & & \cdots & & \\
 & & a_{jj} & \cdots & a_{ji} & a_{j,i+1} \\
 & & & \cdots & & \\
 & \cdots & & & & \\
 & & & & a_{ii} & a_{i,i+1} \\
 & & & & & a_{i+1,i+1}
\end{bmatrix}$$

Fig. 4.1. Similarity allocation in Matrix form A.

Theorem 4.2. Document set D can be arranged in a one-dimensional space so as to satisfy inequality (4-1).

Proof. The following proposition is used to prove the theorem.

Proposition. Let $g(d,c)$, $g(c,e)$ and $g(e,d)$ be three similarity values for three documents d, c and e in a transitive space. Then two of them are equal, and the rest is not smaller than the other [60].

The proof of the theorem is made by induction on a size n of D (see Fig. 4.1). Here a_{ij} means the element $a_{L(i)L(j)}$ of A. Clearly the theorem holds for n=1 and 2.

For n=3, L can be rearranged by Step 2 of algorithm MTDSO such that $a_{12}\geq a_{13}$ holds. Then, by proposition, we have $a_{23}\geq a_{13}$ (This is an equation when $a_{12}>a_{13}$).

Next assume that this theorem holds for a value not greater than i. Similar to the above, we can rearrange the set $\{a_{12}, a_{13}, \cdots a_{1i}, a_{1,i+1}\}$ into a monotone decreasing sequence $a_{12}\geq a_{13}\geq\cdots\geq a_{1i}\geq a_{1,i+1}$.

Now let us show by induction on j ($1\leq j<i$) that $a_{j,i+1}<a_{j+1,i+1}$ $<a_{j+1,i}$ holds. When j is equal to 1, since $a_{12}\geq a_{1,i+1}$, $a_{1,i+1}\leq a_{2,i+1}$. There are two cases to be considered for a_{2i} and $a_{2,i+1}$.

Case (i). If $a_{1i}=a_{1,i+1}$, then by Step 2 for row L(2), we obtain that $a_{2i}\geq a_{2,i+1}$.

Case (ii). If $a_{1i}>a_{1,i+1}$, then since $a_{12}\geq a_{1i}$, we have that $a_{2,i+1}$ $=a_{1,i+1}\leq a_{2i}$.

Generally, if the above inequality holds for k smaller than or equal to j, then since $a_{ki}>a_{k,j+1}$, we have that $a_{k,j+1}\geq a_{ki}$, and $a_{j+1,i+1}\geq a_{j,j+1}$. For $a_{j+1,i}$ and $a_{j+1,i+1}$, the following cases are considered.

CASE 1. If $a_{ki}=a_{k,i+1}$ holds for all k $(1\leq k\leq j)$, then by Step 2 for row L(j+1), we obtain that $a_{j+1,i}\leq a_{j+1,i+1}$.

CASE 2. If $a_{ki}>a_{k,i+1}$ holds for some k $(1\leq k\leq j)$, then by $a_{k,k+1}\geq a_{ki}$, we have that $a_{k+1,i+1}=a_{k,i+1}<a_{k+1,i}$. Therefore, $a_{j+1,i}>a_{j+1,i+1}$. Thus by induction the proof is complete.

The speed of a file search algorithm is accelerated by avoiding wasteful access to the subsections of a stored file not including related items to the queries. The following theorem is used to restrict the search process within a subfield which contains the closest documents to query q.

Theorem 4.3. Let us denote by $D=[d_1,d_2,\cdots,d_i,\cdots,d_j,\cdots,d_k,\cdots,d_n]$ $(1\leq i\leq j\leq k\leq n)$ the documents arranged by MTDSO. Then, if query q, which will be a member of transitive space D, satisfies an inequality

$$g(q,d_i) > g(q,d_j), \qquad (4\text{-}3)$$

then we have the following inequality

$$g(q,d_i) > g(q,d_k). \qquad (4\text{-}4)$$

The proof of this theorem is obtained by the above proposition.

Theorem 4.3 shows the closest or best-match documents (shortly the best-matches) to query q are located on the left half side in D of d_j. The similar theorem holds for the documents located on the left half side in D of d_i.

The next theorem, in addition to Theorem 4.3, can be used for reducing the number of relevancy comparisons between the documents and the query for obtaining the best-matches.

Theorem 4.4. Let B_{st} be a set of the best-matches for q. Then all of B_{st} are in a continued part of D.

Proof. Suppose that d_i is the most left side document of B_{st} in D, and d_j and d_{i+1} $(j>i+1)$ are a member and a nonmember of B_{st}, respectively. Then, by the definition of the best-matches, it holds that $g(q,d_i)=g(q,d_j)>g(q,d_{i+1})$. This is a contradiction.

Thus, after one d_i of B_{st} is retrieved, all the other members of B_{st} are found effectively by examining the document file in the order $[d_i, d_{i+1}, d_{i-1}, d_{i+2}, d_{i-2}, \cdots]$.

4.4 File Organization and Search Strategies

The efficiency of a file search algorithm varies with the division strategies of document set D and with the selection criteria of representative documents of subdivisions. When set D is divided into non-overlapped subsets of an appropriately identical size, the efficiency becomes maximum. The representatives should be selected so that the required field of arranged set D could be specified simply by computing the relevancy between the query and those representatives. Set D is here divided in the following way:

[Division $\mathcal{T}$] Let K be a continued part in D. The division of K into a set of subdivisions K_1, K_2, $\cdots$, K_j, $\cdots$ and K_t $(1 \leq j \leq t)$ is obtained by a monotone decreasing sequence of "thresholds" $1=\tau_0>\tau_1>\cdots>\tau_j>\cdots>\tau_t>0$ such that any document d outside K_j does not satisfy $\tau_{j-1}(\geq) g(d,k_s) \geq \tau_j$ (the equation in the parentheses holds only when j=1). The representative k_s for subdivision K is generally set to the most left hand element in K.

A structured file for set D is constructed by dividing first D into a set of subdivisions, and in turn by applying division $\mathcal{T}$ to each of the resulting subdivisions until the desired hierarchical structuring is attained.

Fig. 4.2 shows a computer implementation of an m-level structuring of D. Since every subdivision is in a continued part in D, the structured file is constructed by accompanying each pointer with the representative, and the positions of the first and the last documents in D of its corresponding subdivision. The representative is generally selected as the most left hand document of the subdivision.

Next let us describe the structured file search algorithms for obtaining the best-matches B_{st} and better-matches B_{tr} for query q

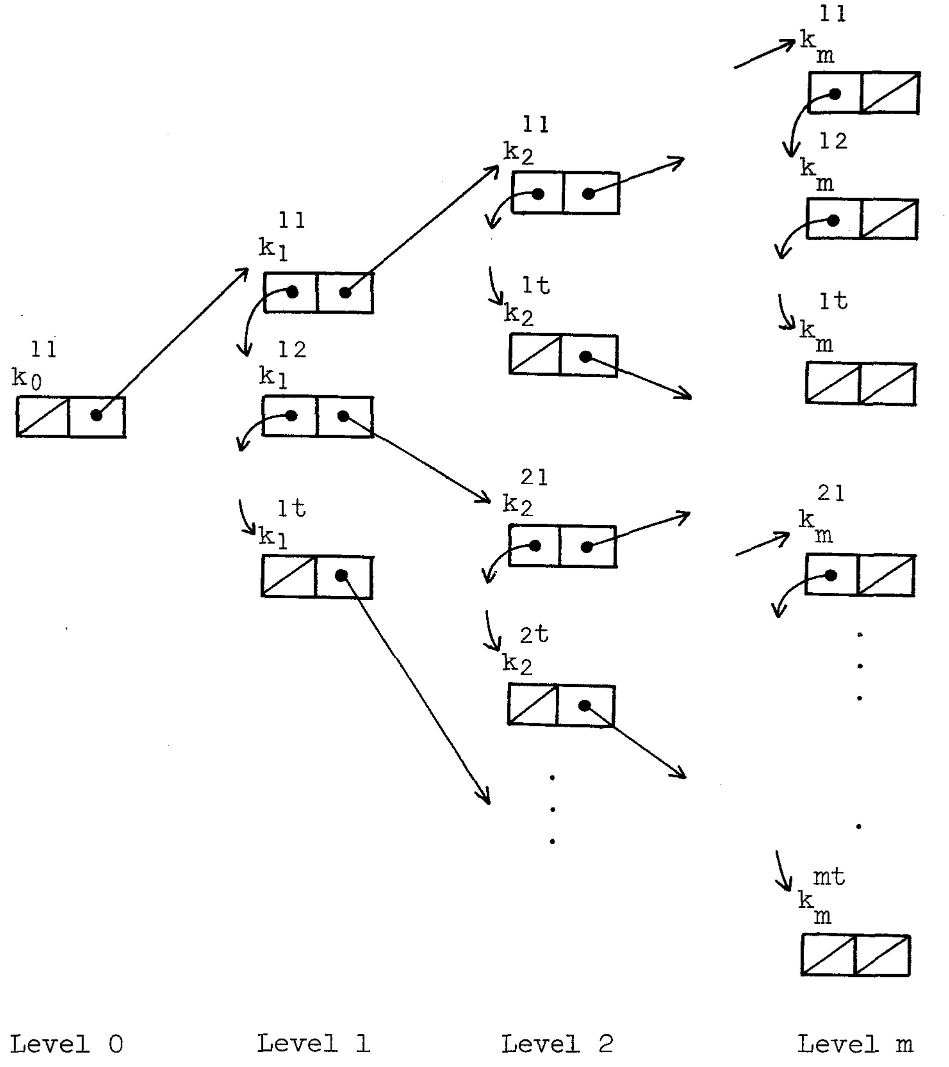


Fig. 4.2. Pointer setting for an m-level structuring of D.

(The better-matches here mean a set of documents arranged in the order of similarity to q). An algorithm for finding B_{st} is first formulated and an algorithm for B_{tr} follows it.

[Algorithm BST] (Best-Matches)

In: Structured file SF and query q.

Note: The subdivision in D under searching is denoted by K. The representative for K is the most left hand element k_s. Further the most right hand element in K is written by k_r.

Procedure:

Step 1. Search $K=\{K_1, K_2, \cdots, K_t\}$ to find the subdivision K_i such that $\tau_{i-1}\overset{>}{(-)}g(k_s,q)\geq\tau_i$, $1\leq i\leq t$ (The equation in the parentheses is for i=t). If $\tau_t>g(k_s,q)$ or $|K|=1$, then set k_r to b, and go to Step 3.

Step 2. If i=1 and $g(k_s,q)\geq g(k_s,k_{s+1})$, then set k_s to b, and go to Step 3; otherwise, set K to $K_i-\{k_s\}$, and go to Step 1.

Step 3. Let j be the position in D of b, and $D_s=[d_1, \cdots, d_j]$ and $D_r=[d_{j+1}, \cdots, d_n]$ be the two continued parts of D. Then, by theorem 4.4, the set B_{st} of the best-matches for q is located in the contiguous parts D_s and/or D_r to b. Terminate the algorithm.

The underlying idea of this algorithm is the same as the one in a method of symbolic key addressing. Algorithm BST is intended to decide the position of the query in a one-dimensional document space by comparing the threshold values and the similarity of the query to the representatives.

The next algorithm is formulated to find the set B_{st} of better-matches for q. After B_{st} is retrieved, the two sets of documents in the left and right hand in D of B_{st} are merged into one document sequence for B_{tr}.

[Algorithm BTR] (Better-Matches)

In: Structured file SF, query q, and the best-matches B_{st} for q.

Note: The set B_{st} is expressed as $[d_i, d_{i+1}, \cdots, d_j]$ $(1<i\leq j<n)$.

Variables D_s, D_r and C are one-dimensional arrays.

Procedure:

Step 1. Set $D_s=[d_{i-1}, d_{i-2}, \cdots, d_1]$ and $D_r=[d_{j+1}, d_{j+2}, \cdots, d_n]$.

Step 2. Merge D_s and D_r into C so that the similarities between the members of C and query q will be arranged in a decreasing order.

Step 3. Merge B_{st} and C into B_{tr} in the same manner as Step 2. Terminate the algorithm.

The discussion in this section is about the file organization and search methods for a document set consisting a transitive sapce with respect to the similarity measure g. The next section is devoted to develop these algorithms for structuring document data which do not always constitute a transitive space.

4.5 Search Algorithms for General Document Space

A document collection of real world information, where the similarity measure is given by f, in general includes ambiguous documents having more than one subject themes. So the problem of searching a document file for the best- or better-matches should be discussed after the grouping of the documents into clusters. Namely, the similarity between two documents is measured by an intra-class similarity, and the search is made separately in every cluster of D. Let us show by the following theorem that the modified version of algorithm MTDSO in which documents are arranged according to f can be used for obtaining document clusters.

Theorem 4.5. Let C ($|C|=m\leq n$) be a subset of D rearranged so as to satisfy $f(d_i,d_j)\geq f(d_i,d_k)$, $1\leq i\leq j\leq k\leq m$ (or $f(d_i,d_j)\leq f(d_i,d_k)$, $1\leq j\leq k\leq i\leq m$) for any d_i, d_j and d_k in D. Then those arranged document pairs satisfy inequality (4-1) with respect to g, an m-step fuzzy relation on C.

Proof. Denote by a_{ij} ($1\leq i\leq j\leq m$) the similarity value $f(d_i,d_j)$, and by a^2_{ij} ($=\max_{1\leq k\leq m}(a_{ik}*a_{kj})$) the value of the two-step fuzzy relaton for d_i and d_j (* is the composition). Then, since the documents in D are rearranged with respect to f, it holds that

$$a_{ij}\geq a_{i,j+1}*a_{j+1,j},\ a_{i,j+2}*a_{j+2,j},\ \cdots,$$

and then $a^2_{ij} = \max_{1\leq k\leq j}(a_{ik}*a_{kj})\geq\max_{1\leq k\leq j}(a_{ik}*a_{k,j+1}) = \max(\max_{1\leq k\leq j}(a_{ik}*a_{k,j+1}), a_{i,j+1}) = \max_{1\leq k\leq j+1}(a_{ik}*a_{k,j+1})$. Similarly, we have that $a^2_{i,j+1} = \max_{1\leq k\leq m}(a_{ik}*a_{k,j+1}) = \max_{1\leq k\leq j+1}(a_{ik}*a_{k,j+1})$. It follows that

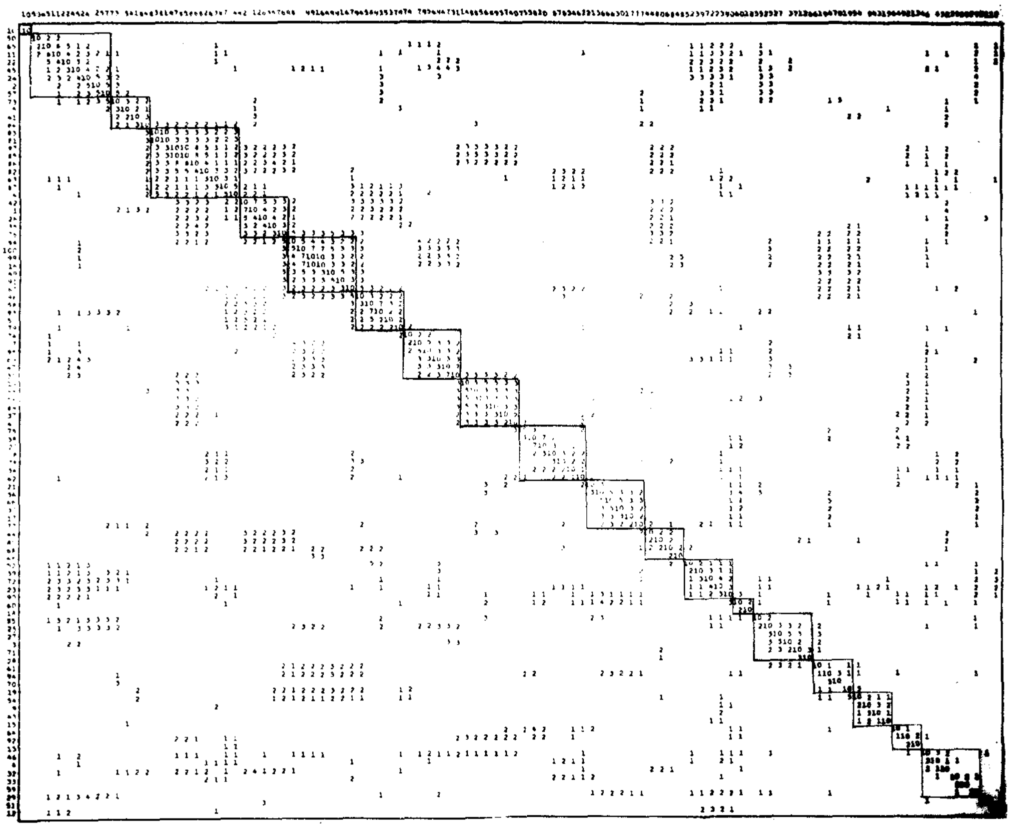
a_{ij}^2 is not smaller than $a_{i,j+1}^2$. Similar to the above, we have that $a_{ij}^2 \leq a_{i+1,j}^2$.

Since $g(d_i, d_j)$ is given by $a_{ij}^{m-1} = a_{ij}^{m}$ $(=a_{ij}^{m+1} = \cdots)$, by the iterative process of the compositions, $g(d_i, d_{j+s}) \leq g(d_i, d_j) \leq g(d_{i+t}, d_j)$, $0 \leq s \leq m-j$, $0 \leq t \leq j-i$. Thus the proof is complete.

This theorem states that all of the documents in C can be placed on a transitive space without changing the indices for document ordering. The set C is considered by the discussions in Chapter 2 to be one of the document clusters. The intra-class similarity g for any document pair in C is defined as an m-step fuzzy relation obtained by maxmin compositions of f.

Fig. 4.3 shows the rearranged similarity matrix for a document collection drawn from the recent Comm. ACM contributions (100 documents). Every similarity value is computed by equation (2-16) (τ=1) in which M(x) means *CR categories* of document x. The documents which constitute a cluster are located in a continued part of L in this figure. The small matrix corresponding to one of the clusters is enclosed with a rectangle[†]. The similarity information outside the rectangles reveals the interrelationship between the clusters, and is used for grouping the documents further for the hierarchical clustering.

[†] Some of the similarity values in the small matrices do not satisfy inequality (4-1) with respect to f, for a general document space consists of documents with more than one subject theme.



* Every row (or line) index shows a document number.
R(i,j)=k (1≤k≤10) ↔ k≤10f(i,j)<k+1.

Fig. 4.3. Similarity matrix for document data drawn from Comm. ACM contributions.

The following theorem holds between two similarity measures f and g.

Theorem 4.6. For any pair (d,c) in C, we have that

$$f(d,c) \leq g(d,c). \qquad (4\text{-}5)$$

If d and c are the most similar document pair with each other, then this is an equation.

Proof. Since g(d,c) is the minimum weight of lines in a maxmin path connecting d and c, inequality (4-5) holds for any pair (d,c). If d is the closest document to c, then these are connected by a line (or path) whose weight is equal to f(d,c).

Theorems 4.4, 4.5 and 4.6 state that the best-matches B^g_{st} ($\subseteq$C) with respect to g is in the continued part of arranged C, and the set B^f_{st} with respect to f is included in B^g_{st}. Thus the sets B^g_{st} and B^f_{st} are retrieved by using algorithm BST. The similarity measure in a general document space D is given by f, so it is required for obtaining B^g_{tr} to compute the intra-class similarity g(d,q) between d and query q. Since the similarity weight $f(d_{L(k-1)},d_{L(k)})$ of any adjacent document pair $(d_{L(k-1)},d_{L(i)})$ in arranged C is the same as $g(d_{L(k-1)},d_{L(k)})$, the value $g(q,d_{L(j)})$ (called transient similarity) for $d_{L(j)}$ is obtained by computing $g(q,d_{L(k)})=\max(f(q,d_{L(k)}),\min(f(d_{L(k-1)},d_{L(k)}),g(q,d_{L(k-1)})))$ for k, $1<k\leq j\leq m$, iteratively, whenever row L(k) is selected for Step 2 in MTDSO. Thus the set B^g_{tr} of the better-matches for q, in which documents are arranged into a decreasing order of similarity g to q, is retrieved similarly by algorithm BTR. (The problem of establishing an efficient method of

finding B^f_{tr} is unsolvable, because no expected value is assumed for f.)

Next let us formulate an algorithm for obtaining the best-matches in a file of the document cluster C. If any query is a member of C, then the best-matches are obtained by using modified versions of algorithm BST and division T such that the required subdivisions are specified based on the values of f. On the other hand, if the above assumption does not hold for a query, then the best-matches are retrieved in the following way: Document clusters arranged with respect to g are divided into a set of subdivisions in which all of the document pairs have the similarity values not smaller than a threshold value τ ($0 \leq \tau \leq 1$), and are organized into an entire document file. The searching of the file is to specify in turn a smaller subdivision K satisfying $f(q,k_s) \leq g(d,k_s)$ for any $d \in K$ by computing the similarity between query q and a representative k_s of the subdivision. Thus an algorithm for finding the best-matches (which is not so efficient as algorithm BST) is formulated as:

[Algorithm GBST] (Best-matches within a document cluster)

In: Structured file for a document cluster and query q.

Note: The continued part in C under searching and its representative (if it exists) are denoted by K (initialized to C) and k_s (initialized to the left side element in K), respectively. Further let s (initialized to 0) be a variable to store $f(q,k_s)$. A document which will show the best-match with q is stored in variable b.

Procedure:

Step 1. If $f(q,k_s)>s$, then set s to $f(q,k_s)$, and search K to find the subdivision M such that a member d of this subdivision satisfies $g(d,k_s)\geq s$. Further set K to $M-\{k_s\}$, and set b to k_s to note that b is one of the best-matches in the already examined documents. Otherwise, set $K=K-\{k_s\}$.

Step 2. If $|K|=0$, then go to Step 3; otherwise, set k_s for a new K to the left side element in K, and go to Step 1.

Step 3. Consider b to be one of the best-matches. Then all of B^g_{st} are in a part contiguous to b in C. Search B^g_{st} exhaustively to find all of B^f_{st} by examining $f(q,d)$ for every d in B^g_{st}. Terminate the algorithm.

When the condition "$f(q,k_s)>s$" in Step 1 of this algorithm does not hold for (q,k_s), the area in K to be searched is the same as the one in the former iteration except k_s.

Algorithm BST (and GBST) is formulated to find the best-matches within a cluster (Hereafter, since $B^g_{st}\supseteq B^f_{st}$, the best-matches mean the documents showing the highest similarity to q with respect to g). Next, a method of determining which clusters are significant to be searched for the best-matches in the entire file is discussed by employing an appropriate storage hierarchy (h level, $h\geq 1$). The entire file is organized by placing document clusters on a storage at the lowest level, and at the higher storage level the abstracted form of the bibliographic items are allocated. The hierarchical data abstraction is obtained by grouping the rearranged items into one stored item at the one higher storage level, which corresponds to a cluster C of documents. Algorithm MTDSO is also usable for this purpose. Since the intra-class similarity g is defined as the minimum line weight of MST, the similarity between two document clusters, i.e., two items, C_i and C_j at this level is defined as:

$$f(C_i,C_j) = \max_{d \in C_i, c \in C_j} f(d,c). \qquad (4\text{-}6)$$

After the desired data abstraction is attained, the documents are indexed by the category numbers reflecting the hierarchical structure. The indexing process proceeds in the following way.

[Indexing *I*] Let $\{C^p_1, C^p_2, \cdots, C^p_{i_p}\}$ be the document clusters at level p ($1 \le p < h$). (i) Index every document d by $\xi^1 \cdots \xi^p \cdots \xi^h \xi^{h+1}$ ($1 \le \xi^p \le i_p$. ξ^{h+1} is a unique document number), called a category number. If documents d and c belong to the same cluster at level p, then d and c give the same values for $\xi^1, \xi^2, \cdots$ and ξ^p. (ii) If documents d and c in different clusters at level h give a line of MST, then d (or c) is further indexed by the category number of c (d).

The documents or their abstracted forms at every hierarchical level are also stored in the form of a structured file. Note that the data banks of the higher storage level take much more compact forms than that of the lower hierarchical storage level. The configuration of a storage hierarchy is depicted in Fig. 4.4, where every document cluster corresponding to a stored item at one higher level constitutes a document file DF.

The search process for the best-matches under the storage hierarchy is firstly to specify in turn the lower level document file which will

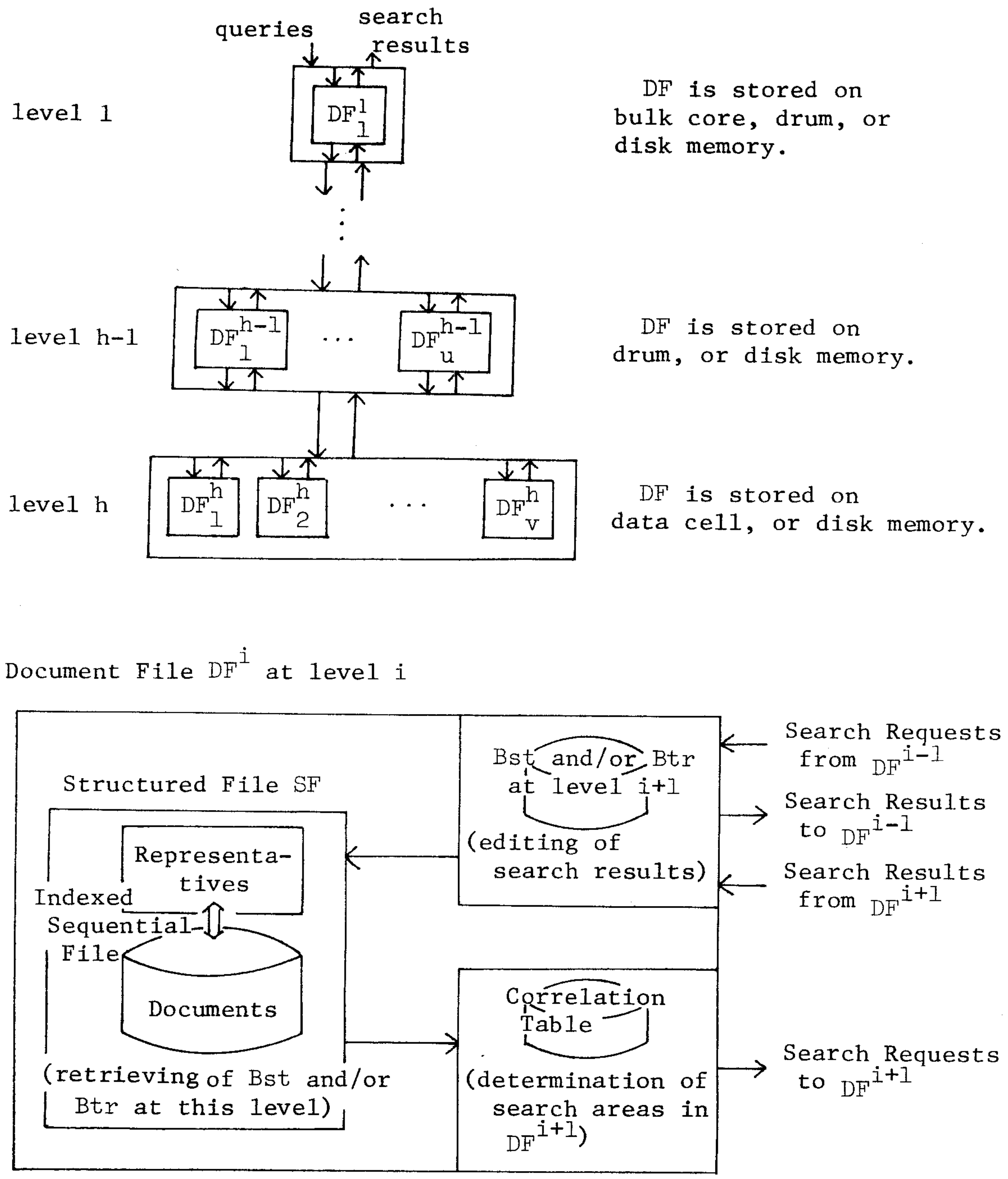

queries
search results
level 1
DF^1_1
DF is stored on bulk core, drum, or disk memory.
level h-1
DF^{h-1}_1
...
DF^{h-1}_u
DF is stored on drum, or disk memory.
level h
DF^h_1
DF^h_2
...
DF^h_v
DF is stored on data cell, or disk memory.
Document File DF^i at level i
Structured File SF
Representatives
Indexed Sequential File
Documents
(retrieving of Bst and/or Btr at this level)
Bst and/or Btr at level i+1
(editing of search results)
Correlation Table
(determination of search areas in DF^{i+1})
Search Requests from DF^{i-1}
Search Results to DF^{i-1}
Search Results from DF^{i+1}
Search Requests to DF^{i+1}

Fig. 4.4. Storage hierarchy configuration of document files.

include related data to the incomming query. The search query could be supposed to be suitably indexed by the automatically generated category numbers, for the user's profile can be expressed and stored in the abstracted form of the document data which were determined to be relevant in the user's past history of retrieval experiences. Next one of the best-matches within the selected document file DF^h at the lowest level h is retrieved by algorithms BST or GBST. Since this document is not always a member of the best-matches in the entire file, it is updated by searching the other files decided to include one of the best-matches in a file of the upper level of DF^h. After one of the best-matches is found, all of the best- and better-matches are easily be retrieved by using the index or indices of this best-match.

The idea of storage hierarchy is intended for the fast searching of a document file of real world information for the document subset showing high relevancy to the submitted queries. The subdivision in which most of the members satisfy (4-1) with respect to f is organized into a structured file, and the documents in the different subdivisions are grouped into a file at the higher storage level. Thus the file structure for the general document data is composed of a hierarchically networked structured file. A best-match search algorithm for the storage hierarchy is formulated as:

[Algorithm HBST] (Best-matches in a storage hierarchy)

In: Hierarchical storage configuration with a structured document file DF^i at every storage level i ($1 \leq i \leq h$), and search query q.

Out: One of the best-matches for q.

Note: Working variables i and j (initialized to 1) are used to show the storage levels under searching. Variable b stores a best-match document in the already examined files.

Procedure:

Step 1. Do Step 2 from level i=j to h-1.

Step 2. Determine the document file DF^{i+1} at level i+1 which is considered to have a related item to q (If q is indexed, this search process can be accelerated).

Step 3. Set j=h-1. Search by algorithm BST (or GBST) the file DF^h for one of the best-matches to q in DF^h, and set b to the best-match document in the already examined files.

Step 4. Let DF^j be a file at level j storing the abstracted form of the previous document file DF^{j+1}. Search DF^j by algorithm BST (or GBST) to obtain an unexamined file DF^{j+1} at level j+1 which shows the similarity to the above abstracted form not smaller than f(b,q), and go to Step 1. If no such file exists, then set j=j-1. If j<1, then terminate the algorithm; otherwise repeat Step 4.

A better-match search algorithm for storage hierarchy is formulated as follows:

[Algorithm HBTR] (Better-matches B^g_{tr} in a storage hierarchy)

In: Hierarchical storage configuration, search query q for r better-matches and best-match document b for q in a file DF^h.

Out: One-dimensional array B^g_{tr} (initialized to $\emptyset$) for storing r better-matches.

Note: Variable B^p ($B^p=\emptyset$ for $1\leq p<h$, and $B^h=[b]$) for every storage level p is a one-dimensional array of size r. Working variable j (initialized to h) indicates the storage level under searching.

Procedure:

Step 1. Extract from DF^h the better-matches to the documents in B^h using algorithm BTR, and merge them into B^g_{tr} based on measure g until r better-matches for q are stored in B^g_{tr}, or DF^h becomes empty. Further store these extracted better-matches into B^h. Set j=h-1.

Step 2. Let DF^j be a file at level j storing the abstracted form of the previous document file DF^{j+1}. Merge B^{j+1} and B^j into B^j based on measure g, and clear B^{j+1}. Set s to the intra-class similarity value between q and the document placed at the r-th position in B^g_{tr}.

Step 3. Search, by algorithm BTR, file DF^j for an unexamined file DF^{j+1} whose abstracted form at the level of Step 2 has a similarity value not smaller than s to the above abstracted form. If no such file exists, then set j=j-1. If j=0, then terminate the algorithm; otherwise, go to Step 2.

Step 4. Select a file DF^{j+1} (first, the file DF^{j+1} at Step 3 is selected) whose corresponding cluster contains a document with the same index part $\xi^1_0\cdots\xi^{j+1}_0$ as that of some document d in B^j (Also the document showing greater similarity to q is apt to be selected as d). If file DF^{i+1} does not exist, then go to Step 3. Otherwise, set B^{j+1} $(\subseteq B^j)$ to a set of documents with index part $\xi_0^1 \cdots \xi_0^{j+1}$, and set j=j+1. If j=h, then go to Step 1; otherwise, repeat Step 4.

The computer implementation of this hierarchical storage configuration consists of a memory hardware (as proposed by Salasin [45]) with increasing data access time and data storage capabilities. This figure would be extended to a computer network for document information retrieval by considering the hierarchical storage hardware as the computer hierarchy with increasing facility of data processing and generality of application areas.

4.6 Computational Experiments

Computational experiments for retrieving the best- and the better-matches in a storage hierarchy are performed on a set D of 100 documents drawn from the Comm. ACM contributions. The similarity matrix for set D is shown in Fig. 4.3. A storage configuration for the document files takes the form of an h-level $(1 \leq h \leq 4)$ hierarchy. A cluster at every storage level, which is obtained by algorithm MTDSO, is stored as an indexed item in the storage at one higher level. The similarity values for items are also stored after subdividing them into clusters at this level. The stored information at the lowest level is a set of indexed documents and the input and intra-class similarity values for every pair of adjacent documents. Since all the items in a cluster is stored as an item at one higher level, the files at the higher storage level require much smaller storage area. Also all items in a file at every hierarchical level are arranged linearly so that the required items will be retrieved effectively. The numbers of items at storage levels 1, 2, 3 and 4 are 9, 16, 35 and 100, and the average (or maximal) numbers of documents in clusters[†] at levels 1, 2 and 3 are 20.0, 8.3 and 3.2 (53, 22 and 7), respectively.

Algorithms HBST and HBTR are used to obtain the best- and better-matches for every incomming query. Four kinds Q_1, Q_2, Q_3 and Q_4 of queries, each of which consists of 10 members exactly matched with the stored documents, are prepared for a storage hierarchy (Hereafter the h-level storage hierarchy is written as SH^h). Any query in Q_i $(2 \leq i \leq h)$ is indexed by the correct class numbers for hierarchy levels 1, $\cdots$ and i-1, and any query in Q_1 is given no class number. Class numbers are used in Step 2 of algorithm HBST to specify the document file which will include a best-match document b to query q. When no class number is given for query q, any unexamined document file is first selected. Next, this algorithm proceeds into Steps 3 and 4, and, if possible, returns to Step 1 for searching an unexamined file with the class number of b. The efficiency of file searching is largely reduced for the input data including many ambiguous documents (which do not satisfy (4-1) with respect to f), for many document files are determined to be searched in Step 4.

[†] One-element clusters are excluded.

After one b of the best-matches is found by algorithm HBST, the better-matches B^g_{tr} ($|B^g_{tr}|=r$, $2\leq r\leq 6$) are found by algorithm HBTR. In this experiment a document in B^g_{tr} having the same similarity value to q as a document outside B^g_{tr} is not retrieved, since it is hard to decide which document is a required one.

Table 4.1 shows the average numbers of relevancy computations between the queries in $\{Q_2, \cdots, Q_h\}$ and the documents at the lowest level or between the stored items at the other levels until one of the best-matches is retrieved from SH^h. The files at the lowest level is searched linearly in this case. Algorithm HBST searches about the half documents in any storage hierarchy to obtain one of the best-matches for a query in Q_1[†]. On the other hand, the full search is required for the document file being not organized hierarchically. The result of retrieving $r \in \{2, 4, 6\}$ better-matches in SH^4 is shown in Table 4.2.

The file search time for a storage hierarchy mainly depends upon the numbers of relevancy computations at the lower storage level. Table 4.3 shows the retrieval result of algorithm HBST for SH^2 and SH^3, where the searching of the files at the lowest level is done by algorithm BST instead of the linear search process. Also these files, i.e., structured files, are organized by subdividing the document clusters into subdivisions. The sequence of thresholds and the numbers of representatives for division T are $0<0.2<0.4<1.0$ and 18, respectively. Variable rp

[†] When the files at the higher storage levels include the representative documents, algorithm HBST can search the files more efficiently.

Table 4.1. Number of items examined at level ℓ for obtaining one of the best-matches in SH^h.

$\ell \backslash h$	2	3	4
1	1.0	1.0	1.0
2	15.5	1.4	1.2
3	*	7.7	1.9
4	*	*	5.4

Table 4.2. Number of items examined at level ℓ for obtaining r better-matches in SH^h.

$\ell \backslash r$	2	4	6
1	1.1	1.1	1.3
2	1.1	1.2	1.6
3	1.5	2.0	2.8
4	3.2	5.5	8.4

Table 4.3. Number of items examined at level ℓ for obtaining one of the best-matches in SH^h.

$\ell \diagdown h$	2	3
1	1.0	1.0
2	2.2 (= rp) 5.2 (=nrp)	1.4
3	*	1.7 (= rp) 4.1 (=nrp)

* The searching of the files at the lowest level is done by algorithm BST.

(or nrp) in Table 4.3 counts the number of representative (non-representative) documents drawn in the executions of algorithm BST. The relevancy computations between the query and the stored items in the structured file are executed sequentially from the first item to the last one in the specified subdivision of the cluster. Thus the rearranged documents are placed on a sequential file with a key region. The result for SH^3 at the lowest level is superior to the other. This is because the more confined subdivisions can be specified in SH^3 than in SH^2. When the stored documents are accompanied by the similarities to the other documents in a cluster, since every document is considered to be a representative, the more prominent result is attained. Also in this case much more area to store the similarity information is needed for SH^2 than for SH^3. Algorithm HBST can be used to retrieve the best-matches to the queries, none of which exactly matched any stored document, at the slight deterioration of the retrieval effectiveness. The reductions of the precision values for retrieving one of the best-matches to the queries in Q_1, Q_2 and Q_3 are 0.21, 0.07 and 0.04, respectively.

The experimental experience reveals that algorithms HBST and HBTR search effectively the files of document data in a real world. The stored information in a storage hierarchy consists of the arranged items and the similarity values for a set of clusters at every level. Therefore, the storage requirement for hierarchically organized files is much less than that for a file storing the entire data without structuring.

4.7 Conclusions

As the studies of analyzing natural language texts proceed, the establishment of a file searching method becomes important for obtaining quickly the documents closely related to any natural language query. The conventional inverted file organization techniques devised in a historical keyword indexing background seem to be inappropriate to manipulate such a flexible problem. Here by ordering the documents linearly, the structured file organization and its searching techniques under the storage hierarchy are presented for obtaining the best- and the better-match documents to any incomming query. The computational experiments show that the algorithms HBST and HBTR search effectively the files of large document data for these documents.

The files in a storage hierarchy are maintained, when a new document c is added, in the following manner. First, by algorithm HBST, obtain a best-match document b in a file DF at the lowest level, and next, store c in the storage location adjacent to b in DF. The deletion of a document is executed similarly. Since the documents at the lowest level are arranged linearly according to their similarities, feed back runs to updata the retrieval result under real-time information retrieval systems can easily be performed by non-expert, and also expert, users.

CHAPTER 5

Concluding Remarks

The basic function of information storage and retrieval tasks has been explained based on the conceptual *relevancy* matching process between the documents and the submitted search queries.

First, the methods of constructing a *word usage dictionary* and of analyzing document sentences were given for the automation of document retrieval process. The dictionary used for specifying the application areas of the words was generated by grouping the words interchangeable in a *predicate network* of the given document sentences. The predicate network is obtained by relating the words and predicates of the sentences to the application indications and predicate patterns, respectively. The grouping method, which is an iterative procedure, is such that only the meaning of a single meaning word constitutes one of the correct application areas at every iteration. Thus the relevancy of the submitted query to the document sentence or to the document is obtained by examining the similarity between them in the predicate network. Next a *structured file organization* technique for the fast retrieving of the relevant documents to any query is formulated by the linear ordering of the documents according to the relevancy or similarity between two documents. The extension of the structured file organization for deciding which document data bases (or document groups) are significant to be searched was to construct a storage or computer hierarchy with increasing facility of data processing.

In the first chapter, it was indicated that the document retrieval process under the laboratory environment is classified into two types, and the most of existing computer information retrieval systems have utilized manual indexing schedules to obtain the document having the specified and formatted subject themes. The study in IR field, therefore, was mainly believed to develop a mechanism of indexing or cataloging the documents and organizing a inverted file. The IR system, however, should be designed, regardless of whether it employs indexing schedules or not, to manipulate the document data more "intelligently" such that the information storage and retrieval operation is made based on the relevancy computation between the conceptual expressions of the document contents.

The grouping of information queries or the specification of users is also an important factor for the design of desirable IR systems. That is, the users with refined profiles should be incorporated into the suitable information files for the efficient and accurate document retrieval. It is probable by the great prevalence of mini or micro computers as well as general purpose ones that the users' profiles might be expressed and stored by the abstracted forms of the document data retrieved to be relevant in the users' past history of the file searching. Individual systems developed separately in various laboratory environments, which are intended to answer the specified information queries of resident research workers, will be summarized into a hierarchically networked system of a world scale for the extensive retrieval of required information.

APPENDIX

Matrix Rearrangement Procedure
and
Graph-Theoretical Algorithms

A.1 Preliminaries

The aims of this Appendix are twofold. One is to present a new data structure for representing interrelationships between two items of given data X (all items are here assumed to be indexed by integers 1, 2, ··· and n). The other is to formulate efficient algorithms for finding graph-theoretical concepts, e.g., data compression, spanning tree, connected component, maximal spanning tree, and clique, used in this thesis. The "efficiency" of an algorithm is measured by the computing speed and the extra storage requirement of the algorithm, and the succinctness of its expression.

Two types of data structures, a neighbor-list and an adjacency matrix, are generally well known to be useful to represent in a computer the graph with a finite node set {1, 2, ···, n}. Also which representation is more convenient depends on the question types being asked in the algorithms. Here a new kind of data structure [30] is presented for expressing a graph G(X), where every node and line (weighted line) correspond to a data item and adjacency (or similarity) information on X. The data structure consists of an n×n adjacency (or similarity) matrix $A=[a_{ij}]$ and a linear list L of size n. The list L contains, as its value, row index of A corresponding to a node of G(X). Adjacency or similarity information between two nodes L(i) and L(j), $1\le i\le n$, $1\le j\le n$, is expressed by the element (L(i),L(j)) of A. (If two nodes L(i) and L(j) are connected with each other, then the value of $a_{L(i)L(j)}$ is set to 1; otherwise to 0.)

One- and Two-Dimensional Sorting Operations (written by ODSO and TDSO for briefness) on a newly proposed expression of a graph are presented before the formulations of graph-theoretical algorithms. These are operations which rearrange the contents of L according to the adjacency or similarity information of the specified row or rows of A. Since any access to A is made by referring to L, A is considered to be labeled by L.

A.2 Matrix Rearrangement Procedure

Let G be a finite undirected graph with nodes {1, 2, ···, n} expressed in the form of a labeled adjacency matrix. Hereafter an element of a labeled matrix in written as $a_{L(i)L(j)}$. First One-Dimensional Sorting Operation ODSO on a specified row of A is formulated.

Let a continued part of row $L(i) = [a_{L(i)L(1)}, a_{L(i)L(2)}, \cdots, a_{L(i)L(n)}]$ of A be denoted by $K(i)=[a_{L(i)L(j)}, a_{L(i)L(j+1)}, \cdots, a_{L(i)L(k)}]$, where j and k ($1 \leq j \leq k \leq n$) are variables to indicate the first and the last positions of K(i) in L(i), respectively. Then operation ODSO on K(i) is defined as follows:

[Algorithm ODSO]

In: Array L and the continued part K(i) of row L(i) of A specified by i, j and k.

Out: The position t of the last non-zero (or significant) element $a_{L(i)L(t)}$ of K(i).

Note: Working variable s is used for binary sorting process in Step 2.

Procedure:

Step 1. Set s and t to the value of j.

Step 2. If $a_{L(i)L(s)}>0$, then exchange the contents of L(s) and L(t), and add 1 to t.

Step 3. Set s=s+1. If $s\leq k$, then go to Step 2; otherwise, set t=t-1, and terminate the algorithm.

Note that ODSO collects the nodes adjacent to L(i) into the continued part [L(j), L(j+1), ···, L(t)] of L. When all accesses to matrix A are made through L, the rearrangement of L is reduced to that of A itself: That is, we have that

$$a_{L(i)L(j)}=\cdots=a_{L(i)L(t)}=1, \text{ and } a_{L(i)L(t+1)}=\cdots=a_{L(i)L(k)}=0$$

after the termination of the algorithm.

The Two-Dimensional Sorting Operation [29] is defined as the sequences of ODSO on each row L(i), $1\leq i\leq n$, $1\leq L(i)\leq n$, of labeled matrix A.

[Algorithm TDSO]

Procedure:

Step 1. Set L=(1, 2, ···, n). Set i to 1, and j to 2.

Step 2. Select row L(i) of A, and execute ODSO on ($a_{L(i)L(j)}$, $a_{L(i)L(j+1)}$, ···, $a_{L(i)L(n)}$). Set j=t+1.

Step 3. Add 1 to i. If i is equal to j, then add 1 to j.

Step 4. If i is not greater than n, then go to Step 2; otherwise, terminate the algorithm.

A.3 Matrix Compressing Process

The graph obtained from data in a real world is known to become sparse (A sparse graph is expressed by a matrix having small amount of non-zero or significant elements). Table A.1 shows some of the statistics for the graphs obtained by applying thresholds to the similarity matrix in Section 2.5. Parameters M_{cc} and ρ_a $(=2m/M_{cc}(M_{cc}-1)$: m is the number of lines of a graph), respectively, indicate the size and the line density of a graph G, and ρ_b the average line density of subgraphs of G. Table A.1 tells us that these graphs are totally sparse but locally dense. The similar situation is shown in Fig. 4.3. (Spärck-Jones [58] indicated that the density of significant elements in such a matrix is practically as low as 10 percent.)

One efficient way of representing a sparse graph is attained by subdividing the matrix of the sparse graph into a set of small matrices, each of which corresponds to one of subgraphs of that graph. Also the matrix should be subdivided so that the interconnection between resulting subgraphs would become smaller. An algorithm for compressing the matrix of a sparse graph is given below:

[Algorithm MC] (Matrix Compression)

In: Labeled similarity matrix A.

Out: Arranged A and L.

Note: One-dimensional array LEV is used to specify the region of A in which significant elements corresponding to lines of ST are stored.

Table A.1. Statistics for threshold graphs.

M_{cc}	ρ_a	ρ_b
100	0.05	0.28
88	0.04	0.26
66	0.05	0.29
21	0.16	0.39

Procedure:

Step 1. Set i to 1, and j to 2. LEV(1)=1.

Step 2. Select row L(i) of A, and execute ODSO on $(a_{L(i)L(j)}, a_{L(i)L(j+1)}, \ldots, a_{L(i)L(n)})$. Set LEV(i+1)=t and j=t+1.

Step 3. Add 1 to i. If i is equal to j, then add 1 to j.

Step 4. If i is not greater than n, then go to Step 2; otherwise, set LEV(i)=n, and terminate the algorithm.

Algorithm MC collects the significant elements into the upper triangular part $\{[L(i), L(LEV(i+1))] | 1 \leq i \leq n\}$ of matrix A. This rearranged matrix can be divided into a set of small matrices by considering $\{LEV(i+1)-i | 1 \leq i<n\}$ to be an index for subdivisions. Algorithm MC is also modifiable so that the more significant similarity information will be collected into the nearer area to the diagonal elements (see Chapter 4).

A.4 Spanning Tree

Next, let us formulate based on ODSO two algorithms for finding a spanning tree of a graph G. The concept of a spanning tree is used to define connected component, maximal spanning tree, cycle, block, and so on.

A spanning tree ST of G is constructed by expanding continuously nodes of any subtree T of G until all nodes of G are incorporated into T. Two methods, called breadth-first BF and depth-first DF methods, are well known in accordance with the way in which nodes are ordered for expansion [39]. The nexts are two spanning tree generation algorithms for a connected graph G.

[Algorithm BF] (Spanning Tree)

In: The graph G is expressed by an adjacency matrix A with label L.

Out: Any line of T under expanding is expressed by the element (L(i),L(s)) of A such that $a_{L(i)L(s)}>0$ in Step 2 of ODSO.

Procedure:

Step 1. Set i and j to 1.

Step 2. If j=n, then terminate the algorithm; otherwise, set j=j+1, and execute ODSO(i,j,n) considering L(i)L(s), which satisfies $a_{L(i)L(s)} \neq 0$ in Step 2 of ODSO, to be a line of ST. Set j=t.

Step 3. Set i=i+1, and go to Step 2.

[Algorithm DF] (Spanning Tree)

In and Out: Same as algorithm BF.

Procedure:

Steps 1 and 2. Same as algorithm BF.

Step 3. If no node L(s) satisfying $a_{L(i)L(s)} \neq 0$ in Step 2 of ODSO exists, then set i=i-1; otherwise, set i=j. Go to Step 2.

Algorithm BF can easily be understood by viewing [L(i),L(i+1),$\cdots$, L(j)], $1 \leq i \leq j \leq n$, as a queue in which every element is a node waiting in line for expansion [34]. After the expansion of L(i) by ODSO, node L(i) is deleted from and its successors are inserted into that queue by increasing the values of i and j. (Successors of L(i) are nodes adjacent in ST to L(i). Node L(i) is called a parent of these successors.) A row for ODSO in BF method, which corresponds to a node for expanding the tree, is selected in the order [L(1), L(2), ···, L(n)]. In DF method, however, a back-tracing procedure into L is needed to find a row for ODSO, and then many elements of A will be referred twice or more.

A.5 Connected Component

Two nodes disconnected in G are also disconnected in a spanning tree of G. An algorithm for finding connected components is then formulated by adding only one conditional statement to Step 3 of algorithm BF.

[Algorithm CC] (Connected Component)

Out: The node of every connected component are lined in a continued part of label array L.

Procedure:

Step 1. Set i and j to 1. Let L(1) be the first element of a connected component.

Step 2. If j=n, then terminate the algorithm by considering L(j) to be the last element of the connected component; otherwise, set j=j+1, and execute ODSO(i,j,n).

Step 3. If j=i, then let L(i) be the last element in L of the connected component, and L(i+1) be the first element in L of a newly generated connected component. Set i=i+1 and j=t, and go to Step 2.

A.6 Maximal Spanning Tree

The spanning tree, where the sum of the line weights is maximal of a line weighted graph (shortly weighted graph) G_W, is called a maximal spanning tree MST. The concept of MST is used in Chapters 2, 3 and 4 for the graph-theoretical clustering schemes. Let T be a subtree obtained by removing the lines of MST whose weights are not greater than a given threshold value τ. Then two nodes in T is connected by at least one path (called a maxmin path) whose minimum line weight is not greater than τ. A node set of T, therefore, becomes one of the single linkage clusters of G_W. An MST generating algorithm is obtained by modifying Prim's algorithm [70].

[Algorithm MST] (Maximal Spanning Tree)

In: Labeled similarity matrix A of G_W.

Out: Refer to algorithm ST.

Procedure:

```
    Step 1. Set i=1.
    Step 2. Exchange the contents of L(i+1) and L(k), where node
L(k) ($i+1 \le k \le n$) shows the greatest similarity to L(i).
    Step 3. Set i=i+1. Let L(i)L(s) be a line of MST, where L(s)
($1 \le s \le i-1$) shows the greatest similarity to L(i).
```

Step 4. If i=n, then terminate the algorithm; otherwise, go to Step 2.

A.7 Clique

The most efficient clique finding algorithm in the previous works has been proposed by Bron and Kerbosch [9]. Their method, however, is not so useful for a sparse graph. Here a new clique finding algorithm is formulated which works effectively for a sparse graph without any extra two-dimensional storage. The next theorem is important to show the correctness of any clique finding algorithm.

Theorem A.1: Let K ($|K|=i-1$, $i>1$) be a set of adjacent nodes in G with each other. Further let Q be a set of nodes adjacent to all of K, and R be any subset of Q. If any clique having, as its nodes, all of K and some of R is already found, then a new clique having all of K includes at least one node, which is a member of Q-R (=P) and not adjacent with some of R. Hereafter R (or P) is denoted by R(i) (P(i)) as the already examined node set (unexamined node set) at level i. Similarly K is expressed as K(i).

Each of K(i), R(i) and Q(i) can be expressed, by rearranging the contents of L, as was shown in Fig. A.1(a). Here LIST(i) and ALDY(i) denote the last and the first positions in L of the nodes of Q(i) and R(i) at level i, respectively. Set S(i) is a collection of

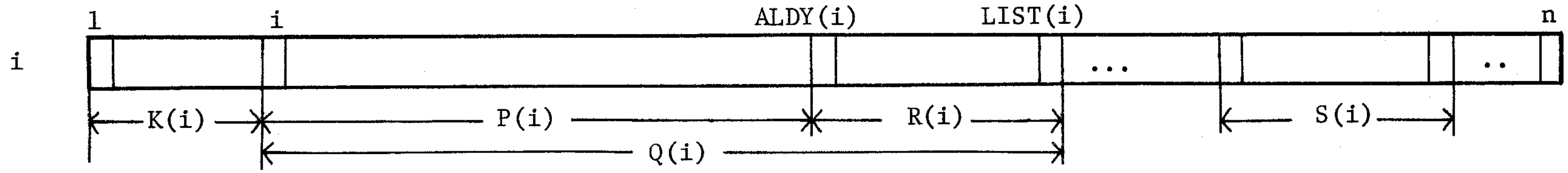


Fig. A.1(a). Arranged pattern of L at level i.

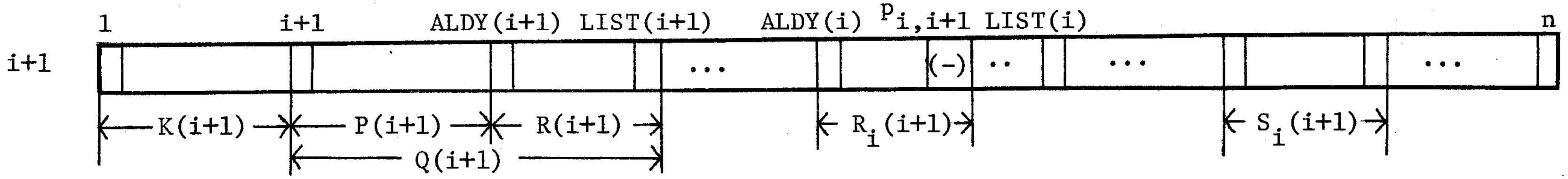


Fig. A.1(b). Arranged pattern of L at level i+1.

nodes which are not in Q(i), and adjacent to all of K(i). If $P(i)=\emptyset$, then a complete subgraph K with nodes $K(i)\cup\{L(i)\}$ can be considered as a candidate for a clique. On the other hand, if $P(i)\neq\emptyset$, then K should be expanded by adding to K a node adjacent to all of K (see (IK1) and (IK2)). To do so, execute ODSO on $P(i)-\{L(i)\}$, on R(i) and on S(i) independently, and obtain Q(i+1), R(i+1) and S(i+1) $(=R_i(i+1)\cup S_i(i+1))$. Set $R_i(i+1)$ is a collection of nodes adjacent to L(i). Set $S_i(i+1)$ is a collection of nodes which are adjacent to all of K(i+1), and are members of already examined nodes at level j ($\leq$i). More explicitly, $S_i(i+1) = \bigcup_{j\leq i} R_i(i+1)$. Set R(i+1) is initialized to $\emptyset$ by setting ALDY(i+1) to LIST(i+1)+1. After all complete subgraphs having $K(i+1)\cup\{L(i+1)\}$ are found, node L(i+1) is incorporated into the already examined node set R(i+1) at level i+1. The index $p_{i,i+1}$, expressed by the sign bit of an element in $\{a_{L(i+1)L(p_{j,i+1})} \mid 1\leq j\leq i\}$, is used to indicate the position of the last element in L of $R_i(i+1)$, and then the set $R_j(i+1)$ $(1\leq j\leq i)$ is located in a continued part [L(ALDY(j)), $\cdots$, $L(p_{j,i+1})$] of L. Thus node set $K(i+1)\cup\{L(i+1)\}$ constitutes a clique iff P(i+2), R(i+1) and $S(i+1)=\emptyset$.

The selection of node L(i) for expanding K(i) is executed as:

(IK1) Let c_i in R(i) be a node having the smallest connected number[†] in S_G whose nodes consist of $K(i)\cup Q(i)$. Then select as L(i) a node in P(i) not adjacent to c_i in G (see Step 6).

[†] A connected number of c_i is a number of nodes connected to c_i in S_G.

The following rule is applied in addition to IK1 for the graph having a locally dense matrix.

(IK2) Execute ODSO on S(i) and select a level h (<i) such that set P(h) contains at least one node useful for expanding a complete subgraph K(h) (see Step 5).

[Algorithm CL] (Clique)

In: Labeled adjacency matrix A.

Note: Variables m and f used for indicating the levels satisfying $R(m)=\emptyset$ and $R_1(f)=\cdots=R_f(f)=\emptyset$, respectively. Note that K constitutes a clique when m>f.

Procedure:

Step 1. Set i=0, m=n, LIST(1)=n, ALDY(1)=n+1 and K=L(1).

Step 2. Add 1 to i. If i>n, then terminate the algorithm. Otherwise, if i>m, i.e., $P(i)=\emptyset$, then go to Step 3; otherwise, execute ODSO (i,i+1,n) to extend K by L(i+1) setting the signs of $a_{L(i)L(s)}$ (and $a_{L(s)L(i)}$) for all s, $i+1\leq s\leq m$, to (+). Set m to the value of t in Step 3 of ODSO, and set LIST(i+1)=m and ALDY(i+1)=m+1.

Step 3. Execute ODSO for nonempty set $R_h(i)$ with the highest level h ($1\leq h\leq i$) setting the sign of $a_{L(i)L(s)}$ (and $a_{L(s)L(i)}$) to (-) in Step 2 of ODSO. At this time, for the value t in Step 3 of ODSO, put the (-) sign to $a_{L(i+1)L(t+1)}$ (and $a_{L(t+1)L(i+1)}$), and set f=i.

Step 4. If $R_h(i)=\emptyset$ for all h, and ODSO in Step 2 is not executable, then go to Step 5; otherwise, go to Step 2.

Step 5. If m>f, then consider K to be a clique, and set i=m. On the other hand if $m \leq f$, then select the greatest r (by rule IK2), $1 \leq r \leq m+1$, satisfying LIST(r)>f, and set r to i.

Step 6. Compare the disconnected numbers for c_i and L(i) at level i (c_i=L(i), when R(i)=∅), and select a new c_i based on rule IK1. Further exchange L(i) for a node L(s) in P(i) disconnected from c_i. If one of the above disconnected numbers is 0, then set i=i-1, and repeat Step 6 until both disconnected numbers are non-zero.

Step 7. Delete 1 from ALDY(i), and exchange L(s) for L(ALDY(i)). Set m=ALDY(i)-1, and go to Step 2.

A.8 Fundamental Cycle

Finally, let us formulate a fundamental cycle finding algorithm [30] to show how the matrix rearrangement procedure works effectively. The formulation of such an algorithm is important to analyze various graph-theoretical structures. Note that any line, called a *chord*, of a graph G not in a spanning tree ST of G, will yield a fundamental cycle. Any chord of ST can be expressed by 1's element in area $R=\{[a_{L(i)L(i+1)},\cdots,a_{L(i)L(t_{i-1})}] \mid 1<i<n\}$ of A, where t_{i-1} is the value of the output parameter t of ODSO for L(i-1) (refer to algorithm BF). The following theorem is used to compute a fundamental set of cycles.

Theorem A.2. For any chord L(x)L(y) $(x<y)$ in R, it is true that the location z in L of the parent node of L(y) is not greater than that of L(x).

Proof. Assume that the value of z is greater than that of x. Suppose first that L(x) and L(z) are not adjacent in ST with each other, then ODSO for L(x) is executed before ODSO for L(z), and we have $z\geq y$. On the other hand, suppose that L(x) and L(z) are adjacent, then the line L(z)L(y) of ST becomes a chord of G. Both cases lead to the contradiction, and then we have $z\geq x$. Similarly, it is true that the location in L of the parent of L(x) is not greater than z.

[Algorithm CY] (Fundamental Cycle)

In: Same as algorithm BF.

Note: Steps 1, 2 and 3 generate a spanning tree ST of G, and Steps 4, 5 and 6 find a set of fundamental cycles through ST. A one-dimensional array PARENT stores a pointer PARENT(i) $(1\leq i\leq n)$ to indicate the position in L of the parent of its successor L(i).

Procedure:

Step 1. Initialize i and j to 1.

Step 2. If j=n, then go to Step 3; otherwise, set j=j+1, and execute ODSO(i,j,n) setting PARENT(s)=i for s such that $a_{L(i)L(s)}\neq 0$. Set j=t+1.

Step 3. If i=n, then go to Step 4; otherwise, if i=j, then set j=j+1 and i=i+1. Go to Step 2.

Step 4. Select a new chord L(x)L(y) ($1<x<y<n$) from C={[L(i+1),$\cdots$, L(t_{i-1})]}. If no new chord exists, then terminate the algorithm; otherwise, go to Step 5 to generate a new fundamental cycle C by putting L(x) and L(y) as the members of C.

Step 5. Set z to PARENT(y), and add L(z) to C. Set y=x and x=z, and repeat Step 5 until x=y.

Step 6. Go to Step 4.

An efficient cycle finding algorithm CY, which is coded into a PL/I procedure, is seen in [30], where algorithm CY and algorithm BGENERATOR proposed by Gibbs [20, 40] are compared. Some of the reasons for the superiority of CY over BGENERATOR are: (i) CY needs, as the extra-storage requirement, two linear lists L and PARENT, and some variables. No copy of the input graph is necessary, for adjacency matrix A is not destroyed in the process of CY; (ii) Any element of A is referred only once in CY, but in BGENERATOR many elements will be referred twice or more; (iii) CY is coded succinctly, and is available for a disconnected graph as well as a connected graph.

References

1. Anderson, T. W. On estimation of parameters in latent structure analysis. Psychometrika, 19 (Mar. 1954), 1-10.
2. Augustson, J. G., and Minker, J. An analysis of some graph theoretical cluster techniques. J. ACM, 17 (Oct. 1970), 571-588.
3. Becker, J., and Hayes, R. M. Information Storage and Retrieval: tools, elements, theories. Wiley, New York (1967).
4. Bell, J. R. The quadratic quotient method: A hash code eliminating secondary clustering. Comm. ACM, 13 (Feb. 1970), 107-109.
5. Bell, J. R., and Kaman, C. H. The linear quotient hash code. Comm. ACM, 13 (Nov. 1970), 13.
6. Bobrow, D. G., and Collins, A. Representation and Understanding: Studies in Cognitive Science. Academic Press, Inc., New York (1975).
7. Bonner, R. E. On some clustering techniques. IBM J. Res. Develop., 8 (Jan. 1964), 22-32.
8. Borko, H. The construction of an empirically based mathematically derived classification system. Rep. No. SP-585, Syst. Develop., Corp., Santa Monica, Calif. (Oct. 1961), 279-289.
9. Bron, C., and Kerbosch, J. Finding all cliques of an undirected graph. Comm. ACM, 16 (Sept. 1973), 575-577.
10. Bruce, B. Case systems for natural language. Artif. Intell., 6 (1975), 327-360.

11. Burkhard, W. A., and Keller, R. M. Some approaches to best-match file searching. Comm. ACM, 16 (April 1973), 230-236.

12. Cárdenas, A. F. Analysis and performance of inverted data base structures. Comm. ACM, 18 (May 1975), 253-263.

13. Codd, E. F. A relational model of data for large shared data banks. Comm. ACM, 13 (June 1970), 377-387.

14. Codd, E. F. Recent investigations in relational data base systems. Information Processing 74 (1974), 1017-1021.

15. Dunn, J. C. A graph theoretical analysis of pattern classification via Tamura's fuzzy relation. IEEE Trans., SMC-4 (May 1974), 310-313.

16. Edmundson, H. P. New methods in automatic extracting. J. ACM, 16 (April 1969), 264-285.

17. Fillmore, C. The Case for Case. In Universals in Linguistic Theory. Holt. Rinehart, and Winston, New York (1968).

18. Gibbs, N. E. A cycle generation algorithm for finite undirected linear graph. J. ACM, 16 (Oct. 1969), 564-568.

19. Gibbs, N. E. Basic cycle generation. Comm. ACM, 18 (May 1975), 275-276.

20. Gibbs, N. E. Generation of all the cycles of a graph from a set of basic cycles. Comm. ACM, 18 (June 1975), 310.

21. Goguen, J. A. L-fuzzy sets. J. Math. Anal. Appl., 18 (1967), 145-174.

22. Goldman, N. M. Sentence paraphrasing from a conceptual base. Comm. ACM, 18 (Feb. 1975), 96-106.

23. Gotlieb, C. C., and Cornell, D. G. Algorithms for finding a fundamental set of cycles for an undirected linear graph. Comm. ACM, 10 (Dec. 1967), 780-783.

24. Gotlieb, C. C., and Kumar, S. Semantic clustering of index terms. J. ACM, 15 (Oct. 1968), 493-513.

25. Gower, J. C., and Ross, G. J. S. Minimum spanning trees and single linkage cluster analysis. Appl. Statistics, 18 (1969), 54-64.

26. Hsiao, D., and Harary, F. A formal system for information retrieval from files. Comm. ACM, 13 (Feb. 1970), 67-73.

27. Ito, T., and Kizawa, M. A method of information retrieval using the dictionary with editing functions. Tech. Rep. of IECEJ, AL72-43 (July 1972) (in Japanese).

28. Ito, T., and Kizawa, M. Information retrieval using a dictionary. National Convention Records of Information Processing Society, 12, 1971 (in Japanese).

29. Ito, T., and Kizawa, M. On the semantic structure of natural language. Trans. of IECE, 59-D (March 1976), 141-148 (in Japanese).

30. Ito, T., and Kizawa, M. The matrix rearrangement procedure for graph-theoretical algorithms and its application to the generation of fundamental cycles. ACM Trans. on Math. Software, 3 (Sept. 1977), 227-231.

31. Ito, T., Toyoda, J., and Kizawa, M. Structured file organization for information retrieval. Trans. of IECE, 60-D (July 1977), 478-480 (in Japanese).

32. Jardine, N., and van Rijsbergen, C. J. The use of hierarchic clustering in information retrieval. Inform. Stor. Retr., 7 (1971), 217-240.

33. Jarvis, R. A., and Patrick, E. A. Clustering using a similarity measure based on shared near neighbors. IEEE Trans. Comput., C-22 (Nov. 1973), 1025-1034.

34. Knuth, D. E. The Art of Computer Programming, Vol. 1: Fundamental Algorithms. Addison-Wesley, Reading, Mass. (1968), 234-238.

35. Lancaster, F. W. Information Retrieval Systems. Wiley, New York, (1968).

36. Lewis, P. A. W., Baxendale, P. B., and Bennett, J. L. Statistical discrimination of the synonymy/antonymy relationship between words. J. ACM, 14 (Jan. 1967), 20-44.

37. Lum, V. Y. General performance analysis of key-to-address transformation methods using an abstract file concept. Comm. ACM, 16 (Oct. 1973), 603-612.

38. Morris, R. Scatter storage techniques. Comm. ACM, 11 (Jan. 1968), 38-44.

39. Nilsson, N. Problem-Solving in Artificial Intelligence. McGraw-Hill, New York (1971), 119-123.

40. Paton, K. An algorithm for finding a fundamental set of cycles of a graph. Comm. ACM, 12 (Sept. 1969), 514-518.

41. Quillian, M. R. The teachable language comprehender: A simulation program and theory of language. Comm. ACM, 12 (Aug. 1969), 459-475.

42. Quine, W. V. O. From a Logical Point of View. Harvard Univ. Press, Cambridge, Mass. (1971).

43. Quine, W. V. O. Set Theory and Its Logic. Harvard Univ. Press, Cambridge, Mass. (1970).

44. Rothnie Jr, J. B., and Lozano, T. Attribute based file organization in a paged memory environment. Comm. ACM, 17 (Feb. 1974), 63-69.

45. Salasin, J. Hierarchical storage in information retrieval. Comm. ACM, 16 (May 1973), 291-295.

46. Salton, G. Associative document retrieval techniques using bibliographic information. J. ACM, 10 (Oct. 1963), 440-457.

47. Salton, G. Dynamic document processing. Comm. ACM, 15 (July 1972), 658-668.

48. Salton, G. Recent studies in automatic text analysis and document retrieval. J. ACM, 20 (April 1973), 258-278.

49. Salton, G. Search strategy and the optimization of retrieval effectiveness. Mechanized Information Storage and Dissemination, North-Holland (1968), 73-106.

50. Salton, G. The SMART Retrieval System. Prentice-Hall, Inc., New Jersey (1971).

51. Salton, G., and Lesk, M. E. The SMART automatic document retrieval system: An illustration. Comm. ACM, 8 (June 1965), 391-398.

52. Salton, G., and Lesk, M. E. Computer evaluation of indexing and text processing. J. ACM, 15 (Jan. 1968), 8-36.

53. Sandewall, E. J. Representing natural language information in predicate calculus. Machine Intell. 6, Elsivier, New York (1971), 255-277.

54. Schank, R. C. Conceptual Information Processing. In Fundamental Studies in Computer Science 3, North-Holland Publishing Company, Amsterdam (1975).

55. Schank, R. C., Goldman, N., Rieger, C. J., and Riesbeck, C. MARGIE: memory, analysis, response generation and inference on English. Proc. Third Internat. Joint Conf. on Artif. Intell. (1973), 255-261.

56. Shapiro, M. The choice of reference points in best-match file searching. Comm. ACM, 20 (May 1977), 339-343.

57. Simmons, R., and Slocum, J. Generating English discourse from semantic networks. Comm. ACM, 15 (Oct. 1972), 891-905.

58. Spärck-Jones, K. Automatic keyword classification for information retrieval. Butterworth, London (1971).

59. Spärck-Jones, K., and Needham, R. M. Automatic term classification and retrieval. Inform. Stor. Retr., 4 (1968)

60. Tamura, S., Higuchi, S., and Tanaka, K. Pattern classification based on fuzzy relations, IEEE Trans. Syst., Man, Cybern., SMC-1 (Jan. 1971), 61-66.

61. Ullmann, S. The Principle of Semantics. Basil Blackwell, Oxford (1957).

62. Van Rijsbergen, C. J. The best-match problem in document retrieval. Comm. ACM, 17 (Nov. 1974), 648-649.

63. Wilks, Y. An intelligent analyzer and understander of English. Comm. ACM, 18 (May 1975), 264-274.

64. Wittel, H., and Greisman, J. Thesaurus Dictionary. Grosset and Dunlap Publishers, New York (1971).

65. Woods, W. A. Transition network grammers for natural language analysis. Comm. ACM, 13 (Oct. 1970), 591-606.

66. Yu, C. T., and Salton, G. Effective information retrieval using term accuracy. Comm. ACM, 20 (March 1977), 135-142.

67. Yu, C. T. A formal construction of term classes. J. ACM, 22 (Jan. 1975), 17-37.

68. Zadeh, L. A. Similarity relations and fuzzy orderings. Inform. Sci., 3 (Apr. 1971), 177-200.

69. Zadeh, L. A. Fuzzy sets. Inform. Contr., 8 (June 1965), 338-353.

70. Zahn, C. T. Graph-theoretical methods for detecting and describing gestalt clusters. IEEE Trans. Comput., C-20 (Jan. 1971), 68-86.