Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

AUTOMATIC DOCUMENT CLASSIFICATION AND EXTRACTION SYSTEM (ADoCES)

by
Xuhong Li

Document processing is a critical element of office automation. Document image processing begins from the Optical Character Recognition (OCR) phase with complex processing for document classification and extraction. Document classification is a process that classifies an incoming document into a particular predefined document type. Document extraction is a process that extracts information pertinent to the users from the content of a document and assigns the information as the values of the "logical structure" of the document type. Therefore, after document classification and extraction, a paper document will be represented in its digital form instead of its original image file format, which is called a frame instance. A frame instance is an operable and efficient form that can be processed and manipulated during document filing and retrieval. This dissertation describes a system to support a complete procedure, which begins with the scanning of the paper document into the system and ends with the output of an effective digital form of the original document. This is a general-purpose system with "learning" ability and, therefore, it can be adapted easily to many application domains.

In this dissertation, the "logical closeness" segmentation method is proposed. A novel representation of document layout structure - Labeled Directed Weighted Graph (LDWG) and a methodology of transforming document segmentation into LDWG representation are described. To find a match between two LDWGs, string representation matching is applied first instead of doing graph comparison directly, which reduces the

time necessary to make the comparison. Applying artificial intelligence, the system is able to learn from experiences and build samples of LDWGs to represent each document type. In addition, the concept of frame templates is used for the document logical structure representation. The concept of Document Type Hierarchy (DTH) is also enhanced to express the hierarchical relation over the logical structures existing among the documents.

AUTOMATIC DOCUMENT CLASSIFICATION AND EXTRACTION SYSTEM (ADoCES)

by

Xuhong Li

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
In Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

Department of Computer and Information Science

May 1999

APPROVAL PAGE

AUTOMATIC DOCUMENT CLASSIFIACTION AND EXTRACTION SYSTEM (ADoCES)

Xuhong Li

Dr. Peter A. Ng, Dissertation Advisor — Date
Professor and Chair of Computer Science, University of Nebraska at Omaha, Omaha, NE

Dr. Gary Thomas, Dissertation Co-Advisor — Date
Professor of Electrical and Computer Engineering, NJIT, Newark, NJ

Dr. D.C. Douglas Hung, Committee Member — Date
Associate Professor of Computer and Information Science, NJIT, Newark, NJ

Dr. Franz Kurfess, Committee Member — Date
Assistant Professor of Computer and Information Science, NJIT, Newark, NJ

Dr. Ronald S. Curtis, Committee Member — Date
Assistant Professor of Computer Science, William Paterson University, Wayne, NJ

BIOGRAPHICAL SKETCH

Author: Xuhong Li

Degree: Doctor of Philosophy

Date: May 1999

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer and Information Science,
  New Jersey Institute of Technology, Newark, New Jersey, 1999

- Master of Computer Science and Technology,
  Beijing University, Beijing, P. R. China, 1994

- Bachelor of Computer Engineering and Science,
  Tianjin University, Tianjin, P. R. China, 1991

Major: Computer Science

Publications:

Xuhong Li, Jianshun Hu, Xien Fan, C.Y. Wang and Peter A. Ng, "Automated Document Filing and Retrieval System: An Overview", *Proceedings of the Third Biennial World Conference on Integrated Design and Process Technology (IDPT Vol 4)*, Berlin, Germany, July 6-8, 1998, SDPS, pp. 231-241.

Xuhong Li, Jianshun Hu, Zhenfu Cheng, D.C. Hung and Peter A. Ng, "Automatic Document Analysis and Understanding System", *The First International Conference on Enterprise Information Systems*, Setubal Portugal, March 27-30, 1999.

Xuhong Li, Jianshun Hu, Zhenfu Cheng, Simon Doong, D.C.Hung and Peter A. Ng,"An Integrated Document Processing System: Document Classification and Information Extraction", To appear in *Proceedings of the 4th World Conference on Integrated Design and Process Technology*, June 1999.

Jianshun Hu, Xuhong Li, Simon Doong, D.C. Hung and Peter A. Ng, "A Thesaurus Model for Document Processing System: The TEXPROS Approach", To appear in *Proceedings of the 4th World Conference on Integrated Design and Process Technology*, June 1999.

Jianshun Hu, Xuhong Li, D.C. Hung, Simon Doong and Peter A. Ng, "Managing Knowledge for an Intelligent Document Processing System", *The First International Conference on Enterprise Information Systems*, Setubal Portugal, March 27-30, 1999.

This dissertation is dedicated to all who love me and whom I love

ACKNOWLEDGEMENT

I would like to express my deep and sincere gratefulness to my advisor, Professor Peter A. Ng, for his patient guidance, constant encouragement and generous help throughout this work and my studying life in U.S.A. Special thanks are given to Dr. Gary L. Thomas, Dr. D.C. Hung, Dr. Ronald S. Curtis and Dr. Franz Kurfess for actively participating in my committee and giving me precious comments to this dissertation.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES
(Continued)

LIST OF FIGURES
(Continued)

LIST OF FIGURES
(Continued)

GLOSSARY

ADoCES..... Automatic Document Classification and Extraction System

LDWG..... Labeled Directed Weighted Graph

DTH..... Document Type Hierarchy

TexPros..... Text Processing System

ODA..... Office Document Architecture

OCR..... Optical Character Recognition

ISO..... International Standardization Organization

HTML..... Hyper Text Markup Language

WWW..... World Wide Web

CHAPTER 1

INTRODUCTION

1.1 Introduction to TexPros System

We live in a decade of information-explosion. With the advance of computer technologies including computer networking and the advent of WWW (world wide web), computers are commonly used and are playing a significant role helping us to get work done effectively and efficiently. In contrast with the tremendous advance of computers, the applications of computer technologies to *document processing* are still very limited. We want to make our offices "electronic". Not only will this save a lot of resources for manufacturing paper; more importantly, this can save our energy and time in retrieving the related information to be processed if we can deal with the documents electrically. It is still a long way from allowing a computer to behave like a human being who "reads" and "understands" the documents, to construct automatically the document indexing to analyze contents of the documents, to categorize a document based on its content, to retrieve a document based on a vague query, etc. A machine is a machine. A computer has to learn how to process documents. By applying artificial intelligence, computers are able to "think" based on knowledge and "experiences" of doing routine work.

In the past decade, document processing has drawn much attention as a research topic. Based on the different media of the documents, document processing can be divided into document image processing and document file processing. As part of the document image processing, a paper media document (such as a conference announcement, a book, a registration form, a patient record, a telex from a client) has to be first scanned as an image into the computer. Document file processing begins when a

document is already in an electronic format (such as a file in plain text, MS-Word, postscript, LaTeX, HTML). For the later case, the scanner and image related technologies are not needed, but our way of processing is still applicable to the file format document.

Based on the varieties of document contents, document processing can be further classified as character-based document understanding (such as reading newspapers, processing forms, office paper document processing, etc) graphics-based document processing, including understanding engineering line drawings, perspective line drawings and music scores. In this dissertation, we restrict our research to office paper document processing. It may also be called structured document processing because we can classify any document into a particular document type based on some attributes within each kind of documents. These structured documents cover most of the paper documents used widely in offices, such as business letters, forms, memos, and scientific and technical articles.

Document image processing becomes an important aspect in office automation. Paper documents still remain the main media of information exchange today. It consumes a large amount of resources when documents become voluminous, and it could far exceed the human being's capability of processing them or at least to do so quickly and conveniently. Beginning with OCR (Optical Character Recognition), document processing proceeds with the automatic document geometric page layout segmentation and logical layout structure analysis. By transforming the original document image into a more compact format, it reduces the large amount of storage required for storing online the document. The resulting format makes easier the further document processing, such as document retrieval and synthesizing.

TexPros (Text Processing System) [5-20] is an integrated document filing and retrieval system. It supports document classification [7, 8, 14, 15], categorization [11, 12], storage [11, 12] and reproducing documents [93, 94], as well as extracting [5, 6, 14, 15], browsing [9, 10], retrieving [9, 10, 18, 19, 20], and synthesizing [18, 93, 94] information from a variety of documents of a pre-defined application domain. Figure 1.1 depicts an overall architecture of TexPros. OCR is used to transform original document images to computer-readable ASCII representations. For each document, the content of its textual parts are recognized and descriptions of its non-textual parts such as logos, figures and pictures are extracted.

In TexPros, the document model employs a dual modeling approach to describe, classify, categorize, file and retrieve documents. This model consists of two hierarchies: a *document type hierarchy*, which depicts the structural organization of the documents, and a *folder organization*, which represents the user's real-world document filing system. In a

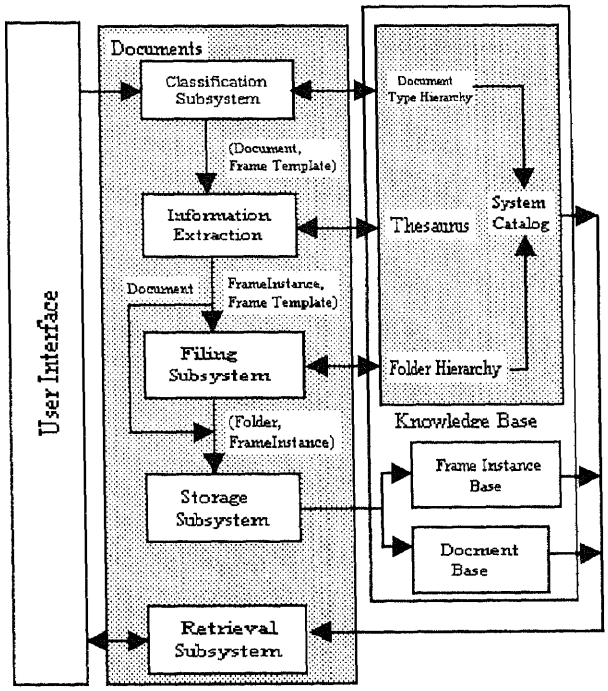


Figure 1.1 Overall architecture of TexPros

user's office environment, by identifying common properties for each document class, documents are partitioned into different classes. Each document class is represented by a *frame template*, which describes the common properties in terms of attributes of the documents of the class and is referred to as the document type (or simply type) of that class. As a powerful abstraction for sharing similarities among document classes while preserving their differences, the frame templates are related by specialization and generalization and are organized as a document type hierarchy whose members are related by an is-a relationship. This is-a relationship and the mechanism of inheritance helps to reduce the complexity of models and redundancy in specifications [95]. After classification, a particular office document can be summarized from the viewpoint of its frame templates to yield a synopsis of the document, which we called a *frame instance*. The frame instances of various document types are deposited in folders over time. A frame instance will generally be in multiple folders along 1 or more paths from the root. Hence, we consider folders to be heterogeneous repositories that are related by an inclusion relationship to form a folder organization. This folder organization is defined by dividing documents for particular areas of discourse into groups until well-defined groups are reached.

1.1.1 Classification and Extraction Subsystem

In TexPros, we develop a knowledge-based document classification system for classifying documents based on the analyses of layout and logical structures. The goal of *document classification* is to find the best fitting document type (represented by a frame template) for a given document. The process of *document extraction* is to instantiate a

frame instance by filling the underlying frame template with information extracted from the document pertinent to the user.

The *layout structure* of a document describes where the segments (also called blocks) of the document are positioned in the physical media, such as paper and electronic media. In order to represent this layout structure, upon which documents could be classified, a *Labeled Directed Weighted Graph (LDWG)* is proposed to capture accurately the layout characteristics of the document. A common approach to recognize the layout structure of a document is segmentation, which divides the document into rectangular areas, called segments. In a Labeled Directed Weighted Graph, each vertex represents a block in the original document and the edge, which connects two vertices, represents the relative positional relation between two blocks. Each edge is labeled as V-edge, H-edge or D-edge to indicate that the relation between two nodes is vertical, horizontal or diagonal respectively. To recognize any documents, such as memos, technical reports and research papers, which often have complex layout and content structures, the one-level segments can not represent accurately their layout structures, since most of the documents usually use more than one spacing scale to separate their layout objects. The *Labeled Directed Weighted Graph* supports the multi-level representation for these complex documents. In general, the variation of spacing used between the layout objects of a document asserts that the layout objects, which lie closely together, tend to have semantically related contents. In addition to the spacing, the segmentation of a document is based on "logical closeness" within and between these blocks. In order to be applicable to the layout varieties, the "logical equivalence" concept is also proposed.

To classify incoming documents automatically, this system has embedded *learning* ability. At first, the system has an empty knowledge base. Upon the input of a collection of sample documents, the system learns and stores the derived knowledge of the layout and logical structures in the knowledge base. The system proceeds to divide each document into segments (which we call the segmentation), to derive a string representation and a Labeled Directed Weighted Graph for each of these documents By choosing good choice of examples, the classification time will be reduced tremendously. After the learning stage, the knowledge base contains the document type names, several sample documents with their string representations and LDWGs under each document type. Based on this knowledge, the system is capable of classifying a new, unknown document by itself with some degree of flexibility, and then extracting information from the document. Frame instances are instantiated, each of which is a simplified form containing all the information pertinent to the significance of the users, and will be used in the later processes, such as automatic filing and retrieval.

Document samples are derived by assigning the properties (in terms of attributes and their values) of its *logical structure* into the layout structure of each document. The content of a document can be divided into structured and unstructured parts. The *structured part* specifies, more or less, the intention of a document. A document is classified as a particular type because all the documents of the same type share the same logical structure even though they may have different layout structures. For example, although two technical papers, one published in the Journal on Systems Integration and the other in Communications of the ACM, may have different layout structures (one is one-column style and the other is two-column style), they uniquely belong to the

technical paper type because both of them share the same logical structure. Both of them have a title, heading, names of the authors and their affiliations, an abstract, keywords, footnotes, page numbers, and a reference list. Usually, the structured part of the documents of the same document type possesses those attributes, which can represent the most important and meaningful information. For example, in the memo type, the structured part of a memo document includes the terms "MEMORANDUM" or "MEMO", "TO", "FROM", "SUBJECT" or "RE", "DATE" etc. The unstructured part of a document is the main body of the document, which is written in free text. To represent the hierarchical relation over the various document types, the document type hierarchy (DTH) is introduced in TexPros [5-8]. It is a parent-children, one-to-many relation between two document types. That is, a parent may have several children, and each child has at most one parent. A child document type inherits all the attributes from its parent. In this dissertation, we complement DTH with further defining of the relation between frame template and DTH and case studies. In the learning stage, besides the system deriving the layout structure knowledge by itself, users participate to build up the relation between the vertices in the layout structure and their corresponding attributes in the logical structure of the same document type. In this way, the extraction can be performed based on the underlying relation between the layout structure and the logical structure.

In operation stage, upon the arrival of an incoming document of an unknown type, the system does the segmentation, and then transforms the segmentation and the positional relation among the blocks into a string representation and a LDWG. To classify an incoming document, the system tries to find a good match with one of the sample documents. If the degree of their "similarity" is above the pre-defined threshold

("Goodness") and the distances between this incoming document and the other sample documents are "far enough" ("Uniqueness"), then the type of this document can be confirmed. Classification is completed successfully. In the case that after thorough matching with all the samples in the document sample base, all the degrees of "similarity" are below the threshold, then the system requires the users to interact in re-training it, in order to let the system incorporate this new case. If the distance between this LDWG and some of other sample documents are not "far enough", it means that the "uniqueness" is unsure. The system then consults the "similarity table", which stores the "similarity" between any pair of document types.

After an incoming document is assigned with a document type, it proceeds to the extraction stage. First of all, the system finds a LDWG under this document type which is the best mask for this incoming document. Based on the relation between the layout structure and the logical structure, the extraction of information from the document can be done automatically by the system.

1.1.2 Storage and Automatic Filing Subsystem

In TexPros' filing system, the notions of document type hierarchy and folder organization are incorporated into a multilevel repository architecture for storing documents [11,12]. We employ a three-level architecture of a document repository to store documents. At the first level, the storage contains original documents. A physical storage containing frame instances is at the second level. There are many ways for organizing the frame instances to enhance the performance of frame instances search and retrieval. The notion of a bookcase organization (also described in terms of relation tables [9]) for storing frame

instances is utilized [102]. Analogous to inverted indexing, each frame instance has a pointer, which points to its corresponding original document. The third level is the logical storage, which is organized as a folder organization. Each folder is a virtual repository for a set of frame instances. It is a virtual repository because it stores pointers pointing to the frame instances which are at the second level.

The filing system provides a flexible search and retrieval facility that allows browsing through collections of frame instances and retrieving frame instances according to different criteria, using the information related to document types and the frame instances in close proximity within a folder in the folder organization.

Automatic filing of frame instance into proper folders of a folder organization based on each folders' criteria becomes a central issue here. Predicates are formalized for specifying folder criteria which govern the grouping of frame instances, regardless of their document types. The predicates are used for specifying characteristics of frame instances and the properties of attributes (i.e., those attributes appeared in frame templates for describing the document types).

An agent-based filing architecture is used to implement the extended notion of folder organization, which could automate the document filing (i.e., deposit an incoming frame instance into an appropriate folder) and cope with the subtitles of folder organization. Associated with each folder, there is a filing agent along with a set of criteria. Each agent has its private data structures (called attributes) and operations for manipulating the data structures. The agents communicate with each other through message passing.

Each agent stores frame instances in two places: repository and output-buffer. The repository stores only the pointers which point to frame instances that satisfy the agent's criteria. Only frame instances in the output-buffer are waiting for distribution. For filing a frame instance w into a folder organization, when it arrives at the output-buffer of the agent of the rooted folder, the agent then distributes a copy of the frame instance w to its descendants, which, in turn, distribute a copy of w to their descendants. Some of the operations involved in passing frame instances among the agents are: Distribute (sending frame instances in the output-buffer of an Agent A to the output-buffer of A's descendants), Filing (depositing the frame instances, which satisfy the A's criteria, from the output-buffer A to its repository), Transfer (loading frame instances from the repository of A into A's output-buffer), and Discard (discarding a frame instance from the output-buffer of an agent A, if it does not satisfy A's criteria).

For detail discussion in this topic, please reference [11,12].

1.1.3 Document and Information Retrieval

The retrieval subsystem for TexPros provides functional capabilities for processing incomplete, imprecise and vague queries and provides users with semantically meaningful responses [18,19,20]. The design of the retrieval subsystem is highly integrated with various mechanisms for achieving these goals. Firstly, a system catalog including a thesaurus is used to store the knowledge about the database. Secondly, there is a query transformation mechanism composed of context construction and algebraic query formulation modules. Given an incomplete or imprecise query, the context construction module searches through the system catalog for the required terms and

constructs a query that has a complete and precise representation. The resulting query is then formulated into an algebraic expression. Thirdly, in the retrieval process, vague queries can be entered into the system until sufficient information is obtained, through the use of the browser, to the extent that the user is able to construct a query for his request. Finally, when processing of queries fails by responding with a null answer to the user, a generalizer mechanism is used to give the user cooperative explanation for the null answer.

Central to the query processing system is the system catalog. The system catalog contains not only the metadata (i.e., the description of the document type hierarchy and the folder organization) describing the database (i.e., the frame instances of documents), but also a thesaurus. The thesaurus contains synonymous for terms that are relevant to the user, terms that are semantically equivalent, and correspondences between terms and the index terms of folders, templates or attribute name types actually residing in the database. An object network (O-Net) is employed to describe the snapshot of a subset of information contained in the system catalog [18]. The O-Net provides a path for looking up relevant information from the related system frame instances. The components of the network include schema elements, data elements and the dual modeling relationships. The schema elements give the description of the folder organization and the document type hierarchy. This description includes the frame instances, the folders, the frame templates, and the attributes at different levels.

TexPros allows users, firstly to retrieve documents and information through frame templates and to match the given values against slot values on the frame instances of a single class or several classes. Secondly, it allows users to manipulate and query folders,

and to perform folder-at-a-time operations. Thirdly, TexPros allows users to browse through the folders containing frame instances of a specified document type, and the contents of frame instances of a particular type contained in a specified folder. Sometimes, the user may start with a vague idea, and as the search progresses, the notion of what is wanted becomes clear to the user, and there may be a shift in emphasis. Browsing the folders and the contents of frame instances helps users reformulate dynamically queries. Fourthly, if the user only has a rough idea or only can describe partially the requested documents, he may perform the concept-based retrieval. Documents whose keywords match the query partially are also returned. At times, a user may be impressed with a picture in the document, and may type in words describing the picture. It is also likely that the input query differs from the descriptions matched partially or conceptually with the query. In the concept-based retrieval, there is no clear distinction between documents that qualify the specified condition and those that do not; some documents are more relevant, while others are less so. For such "fuzzy" types of queries, TexPros always returns a list of documents, ranked according to the degree of their relevance to the query.

1.2 Introduction to ADoCES Subsystem

In office automation, processing, filing and retrieving documents are the essential functions that help to increase the productivity of an office work. Within this application domain, document analysis and understanding are the crucial activities for integrated office automation systems. However, despite major advances in computer technology, the degree of automation in acquiring and inquiring data from documents is still very limited and difficult to use. Most of the existing document analysis systems [2, 4, 27, 33, 37, 38,

40, 47, 48, 50, 55, 54, 56, 61, 66, 67, 75, 76] are restricted to a relatively small application domains. Even if some of the systems can be adapted to a new domain, this adaptation is as time consuming as developing a new system from scratch. Therefore, it is challenging to develop a system which can be easily adapted into any application domain and provides a high degree of automation by applying artificial intelligence in document analysis and understanding.

In practice, it is still very difficult to design an automatic system for transforming documents of different application domains into their corresponding frame instances. Usually interactive editing is unavoidable. One of the problems is how to decrease the human interaction to the least level and make human operations as simple and reasonable as possible. However, the system can be trained by learning from user interactions during editing to get better and smarter.

For the system to have some degree of intelligence and to process documents like human beings do, the system must be able to "learn" and store the background knowledge in a way which allows the document analysis and understanding to be done accurately, effectively and efficiently.

Document processing can be divided into two stages: *document analysis* and *document understanding*. A document has two structures, namely the layout (geometric) structure and the logical structure. These two structures of a document provide the alternative but complementary views of the same document. Extraction of the layout structure from a document is referred to as document analysis. Mapping the layout structure into logical structure is referred to as document understanding.

The relationship between the layout structure, logical structure, document analysis and document understanding can be found in Figure 1.2.

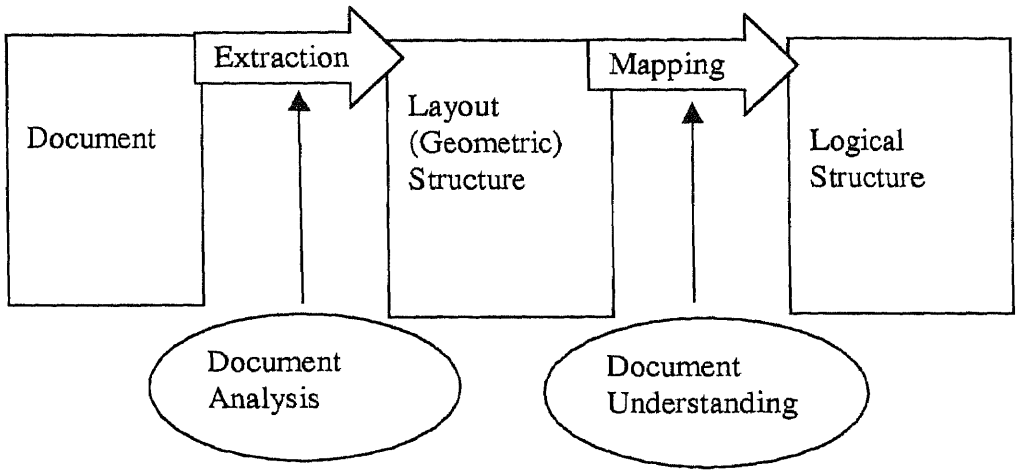


Figure 1.2 Overview of document process

Given a document, the document analysis determines its structure to form a layout structure, which is basically composed of a number of blocks; the document understanding gives annotations to the blocks in the layout structure to form the logical structure of the document.

In this dissertation, document analysis and document understanding are referred to as *document classification* and *document extraction*, respectively.

In document processing, most of the research work is in the application of newspaper reading [48], form content extraction [1, 24], email reading [33, 37, 38, 56] and text categorization [29, 30]. However, the Text Processing System (TexPros)[5-20] is a general-purposed intelligent document processing system. It could process most of the documents in an office. We consider these documents as semi-structured documents. These documents can be classified into different document types according to the geometric and logical structures of the documents.

1.2.1 Layout (Geometric) Structure

According to ISO 8613-1:1989(E) [45], a layout structure (or referred to as geometric structure) is obtained by dividing incrementally the content of a document into increasingly smaller parts, on the basis of its structural presentation based upon some predefined criteria. A geometric structure of a document consists of three basic elements: block, frame and page. A *block* is a basic geometric object, which corresponds to a rectangle containing a portion of the document content. A *frame* is a composite geometric object containing one or more blocks or other frames. A *page* is a basic or composite geometric object containing one or more frames (if it is a composite object). A *page set* is referred to a number of pages or one or more page sets. The object at the highest level in the hierarchy of a layout structure is then referred to the *Document Layout Root*.

1.2.2 Logical Structure

Logical Structure is the result of dividing repeatedly the content of a document into increasingly smaller parts, on the basis of the human-perceived meanings of the content. Each part has a logical object, which is an element of the specific logical structure of a document. The logical structure of a document is composed of logical root of the document, the basic logical object, or the composite logical object (i.e. groups of basic or composite).

1.2.3 Relationships between Geometric and Logical Structures

The geometric and logical structures of a document provide alternative but complementary views of the same document. The layout structure of a document

represents the geometric view of the document; the logical structure of a document has the logical meaning interpreted by the users based on the content of the document. The documents of the same type may have different layout structures but share the same logical structure. These two structures are independent of each other in principle. However, a correspondence between geometric objects and logical objects may exist. A logical structure corresponds to a variety of geometric structures. Therefore, there is no one-to-one correspondence. Thus, there is a mapping of a geometric structure into a logical structure, and the reverse transformation does not always exist.

Document can be identified as its document type using layout analysis and logical analysis. For layout analysis, an image page is partitioned into blocks, which are the maximal homogeneous regions. Each block is classified and then assigned with a type, such as text, graphic, image and table.

For logical analysis, given the type of a page, each block is assigned with a logical label, such as title, logo, footnote (base on its functionality in the document) or name, address and date (base on the content of this block). It determines the relationship between blocks and the reading order of the blocks.

The layout analysis and logical analysis output the new format of the original document image in such a way that is suitable for further processing, such as document filing and retrieval. Therefore, the output of the layout analysis should be in a format which is easy to process electronically and manipulated during the filing and retrieval operations. Any discussion of layout analysis without taking the other components into further, deep consideration is meaningless.

In contrast to the geometric layout analysis, the logical layout analysis has received much less attention. There are only a few papers discussing logical label assignment. [48, 54, 70, 73] These works apply the rule-base and the knowledge base on some specific applications. They didn't consider application on a general domain.

Taking a glance at a document of a known type, such as the memo type or a familiar kind of journal paper, we can identify the type of the document (classification) and identify certain components (extraction), such as the sender of a memo, and the title and the authors of a technical paper without reading the contents of the document. We can do it because we implicitly know from our experience the typical layouts of these types of documents. For example, take a glance at the layout structures given in Figure 1.3. By observation, we can tell easily that the three images in the upper part are the memos type, and the three pictures in the lower half are of article type. That implies that it is possible to conduct the document classification based on the layout structure of the document even without knowing the content of the document. In addition, we have enough flexibility to do the classification. Even if there are some stylistic variations, such as the font used in a document of a type is different from one we have seen before, we still can tell the correct document type. Furthermore, even if the format used in some parts of the document is different from the previous documents of the same type, we still can do the classification and the extraction correctly. This dissertation proposes a new layout analysis of documents based upon the deep analysis on the content of a document and animation of the users behaviors. Besides the relatively positional relations among the rectangles after the segmentation, we introduce the concept of logical equivalence of the blocks among the documents of the same type.

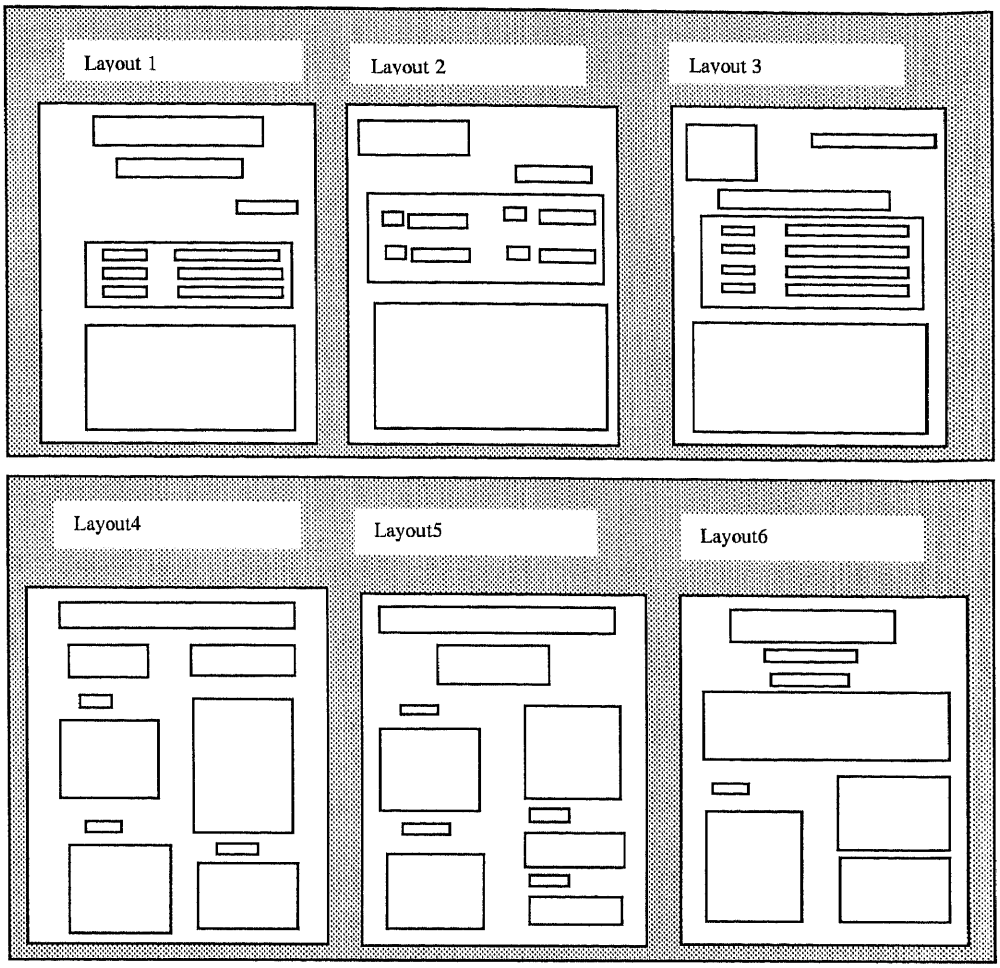


Figure 1.3 Different layout structures of the different document types

In order to get the layout structure of a document from its image file, two approaches have been used in document analysis -- *top-down* [41] and *bottom-up* [98]. After applied these approaches, we can divide a page of an image file into rectangular regions, each of which is of the type of text, image, graphics or logo. But no one does further structural analysis for each of the text region. A common approach to recognize the layout structure of a document is *segmentation*, which divides the document into rectangular areas, called segments. Dengel & Barth [37] and Mattos et al. (1990) divide

the document into several segments; each of the segments is associated with a semantic object such as title, subject, date, text strings, etc. To deal only with documents, such as electronic mails [33], business letters [37] or form documents [24] which have inherently fixed layout structures, the simple segmentation technique is sufficient for identifying their layout structures.

In [5, 6, 7, 8], the Nested Segmentation Algorithm and Adjacent Relation Segmentation were proposed to analyze the structures of documents. These two approaches use the L-S tree structure to represent the document's layout structure and logical structure of documents. Then for the document classification, the tree-matching is used to perform approximate matching between the L-S structure of a given document and a sample L-S structure from the knowledge base. During the tree-matching, the "editing-distance" is used to evaluate the degree of match between the document to be classified and the sample trees in the knowledge base. Basically, if a perfect match cannot be found, then the rules of layout equivalence are used to match two trees.

In this dissertation, the concept of "logical equivalence segmentation" is proposed. This means, the process of segmenting a given document into various blocks (called segmentation) is based on the relative spacing information and the "logical closeness" analysis among the adjacent regions. The adjacent regions will be grouped together if the contents of the adjacent regions are logically close enough. The detailed discussion of this part will be given in chapter 2.

The rationales for using the concept of logical equivalence instead of the layout equivalence for segmentation and matching are as follows: Basically, it is because the human being does the analysis in this way. For instance, given two memos, even if both

memos are written in different formats, a human being can still identify that these two parts are "equivalent". This equivalence is from a logical view. In addition, the rules of the layout equivalence, which are applicable to one document type may *not* be properly applied to another document type.

Given a document of multipages, from a global view, there corresponds a set of graphs. For each page of a document, there is a graph to represent the relative relation of blocks within the page. Each block has its specific information, such as document type, coordinates, weight, etc. Our focus is not on the layout analysis. It is to understand the document in terms of the mapping between its layout structure and the logical structure. The logical view of the document structure can be defined as a hierarchical relation. In addition to the classification of an incoming document as a type, our focus is to extract automatically the contents of the document and assign them to its layout structure to form the logical structure. However, it is possible to use the layout structure to match the existing layout structures to classify a document as a type. The automatic extraction of the contents of a given document is based on the relation between its physical layout structure of the blocks and its logical structure.

By processing a document, firstly, the system recognizes the type of the document, and extracts the related contents if a perfect match between the logical structures of the document could be found. Secondly, the system must have a certain degree of "flexibility" and "inferring" to enhance its ability of "learning" from its experiences.

1.2.4 Terminology

Before giving a full description of the automatic document classification and extraction system (ADoCES), we shall introduce some terminologies that will be used throughout this dissertation. The relations between the terms will be presented afterwards. For effective document representation, we introduce several concepts.

Definition 1: A *virtual page* consists of a number of ordered tuple physical pages of a given document to be considered.

A layout structure of a document can be represented by a *Labeled Directed Weighted Graph* (LDWG). This graph contains geometrical information about all the ordered elements of a virtual page of a document and their relative positions.

Definition 2: A LDWG is a 2-tuple graph G=(V, E), where V is a set of vertices, each of which represents a block in the virtual page and E is a set of edges (v_i, v_j, R), where v_i, v_j are vertices in V, and R is one of the labels H (Horizontal), V (Vertical) or D (Diagonal).

A labeled directed edge between two vertices with a label H, V or D is defined if there corresponds two adjacent blocks such that they are next to each others horizontally, vertically or diagonally, respectively to indicate their adjacency relations and their relative positions between blocks. Since a block can be subdivided, the corresponding vertex in the graph G can contain another LDWG. In chapter 3, we define formally the concepts of horizontal relation, vertical relation, diagonal relation and adjacency.

An example is given in Figure 1.4. A page consists of three blocks: v_1, v_2 and v_3. In that, v_1 is horizontal with v_2 and v_1 is composed of v_{11} and v_{12}, which are vertically

adjacent. Then the internal representation of these blocks in this page is given in terms of a labeled directed weighted graph.

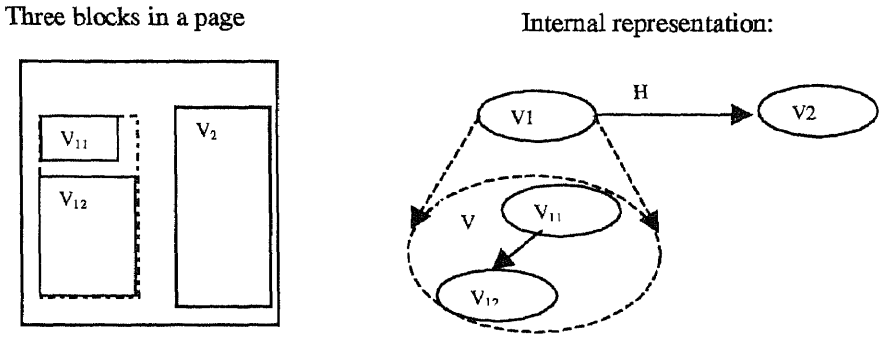


Figure 1.4 LDWG for a page of blocks

Definition 3: A *vertex* represents a physical block in a virtual page. It is defined as a block. There are two types of blocks, namely a simple type or a composite type. A block of simple type is a basic geometric rectangular area, which corresponds to a simple logical entity. (For instance, v_2 in Figure 1.4 is a block of simple type). A block of the composite type is a rectangular area, which comprises several blocks of the simple or composite types. (For instance, v_1 in Figure 1.4 is a composite block.)

The internal structure of a vertex stores: physical information about the block, (such as, Xmin, Ymin, Xmax, Ymax and BlockType), hierarchical relation among block and its subblocks.

A vertex's attributes include the following:

a. String BlockName;

   This is the name of this block. It is unique within one document.

b. Int Xmin, Ymin, Xmax, Ymax;

These four integers store the physical position in term of left_top point and right_bottom point of this rectangle.

c. enum blockType;

This item is an enumeration type. Its value should be one of TEXT_BLOCK, IMAGE_BLOCK, TABLE_BLOCK, GRAPHICS_BLOCK or OTHER_BLOCK.

d. Boolean stationary;

If it is true, then an absolute match for height and width for this block is done during matching.

e. Vector subBlocks;

If this block is simple block, then this vector is NULL. Otherwise, all the sub-blocks under this block are stored in this vector.

f. Boolean whetherContinuing;

If true, then it indicates that the block is contained in more than one rectangle.

The logical view and physical view of hierarchical relation of document's layout structure, pages and blocks is shown in Figure 1.5 and Figure 1.6 respectively. The internal information stored in layout structure is shown in Figure 1.7.

Definition 4: An *attribute* is simple if it is atomic. That is, the attribute can not further be divided into a set of attributes. An attribute is composite if it can be subdivided into a set of atomic attributes or composite attributes.

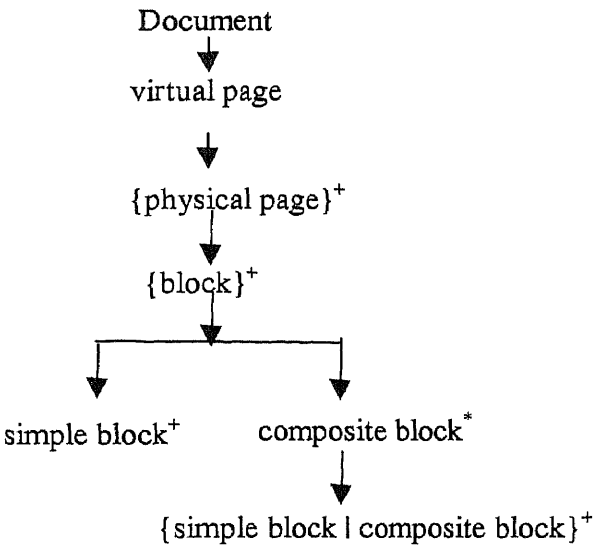


Figure 1.5 Hierarchical view of layout structure

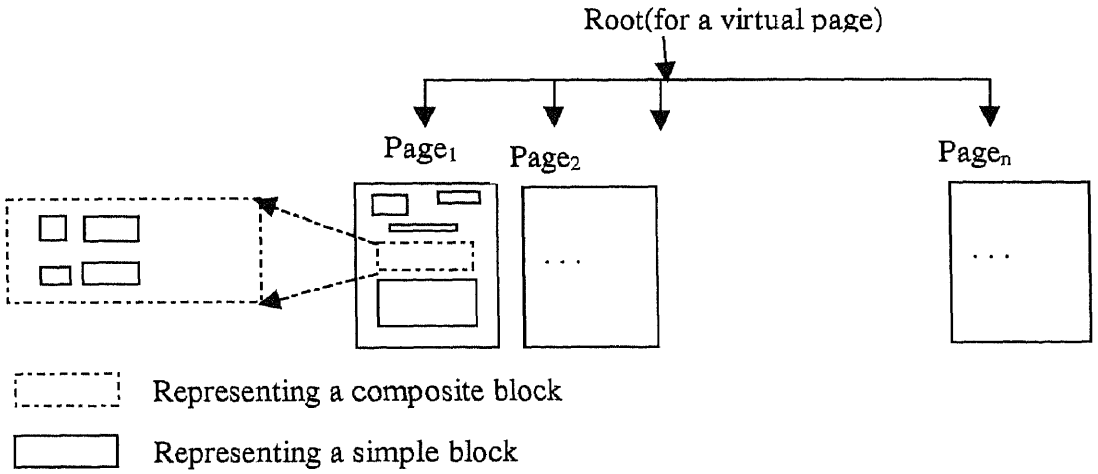


Figure 1.6 Image file after segmentation

For example, FirstName is a simple attribute for a person. Address is a composite attribute, which can be decomposed into StreetName, ApartmentNumber, CityName and ZipCode. The internal representation of layout structure of a document is shown in Figure 1.7.

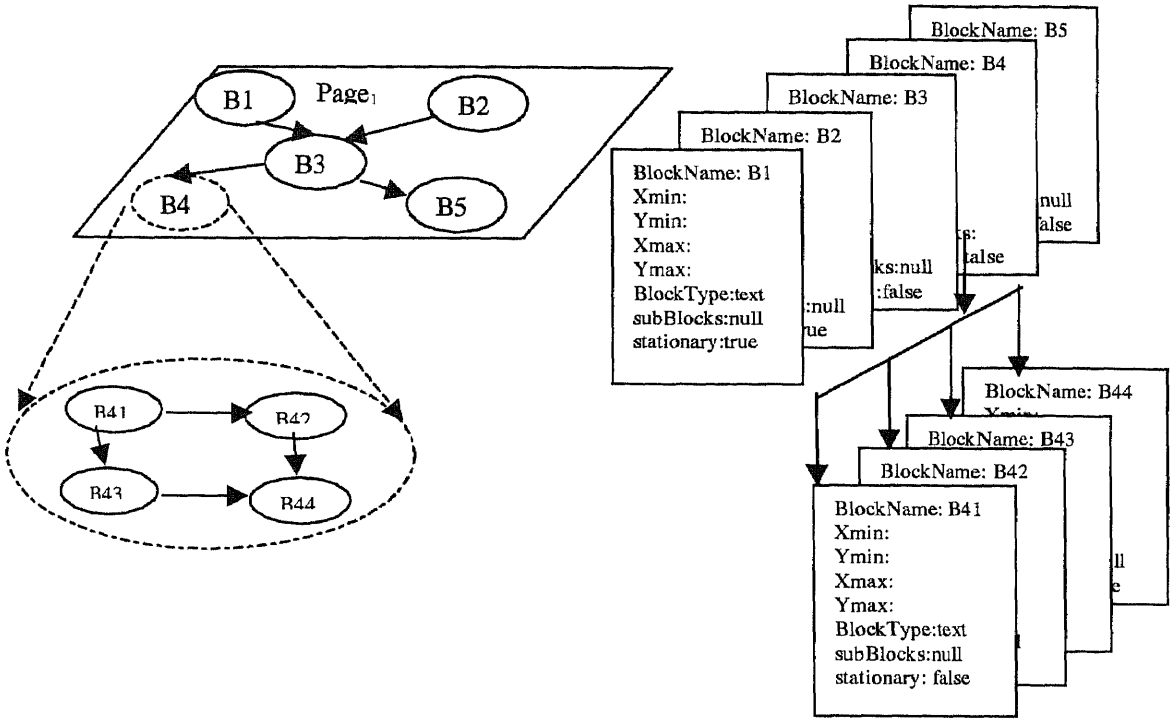


Figure 1.7 Internal representation of *layout structure*

Definition 5: A *Frame template* F=[<A_1:T_1>,...,<A_m:T_m>], where A_i is an attribute over the attribute type T_i, describes the properties of a document class.

A *frame template* is used to keep a record of logical meanings for a document. For a class of documents, there corresponds uniquely a frame template, which describes the important properties of the document type of its class. These frame templates of different document classes are organized as a hierarchical structure, which is called the *document type hierarchy*, based upon the generalization and specialization relation among frame templates and their inheritance properties among them.

A frame template is composed of a group of attributes. Each attribute may be of simple or composite type. For example, a frame template of the memo type may consist of the attributes, From (sender), To (receiver), Date, Subject, Content, etc. as shown in

Figure 1.8. If the attribute "sender" is treated as a simple type, then the name of the sender is a string. However, if it is of composite type, then the attribute "sender" can be further decomposed into FirstName, MiddleName and LastName attributes.

A frame template definition for memo type is given in Figure 1.8.

Attribute Name		Type
Logo		String
Address	StreetName	String
	CityName	String
	ZipCode	Number
DepartmentName		String
To		String
From		String
Date		Date
Subject		String
Content		String
CC		Email

Figure 1.8 Frame template definition for memo type

Conceptually, the frame template in TexPros is quite similar to the relational schema in the relational database. However the frame template is far more complicated than the relational schema. The frame template allows the composite data type, hierarchical relations and repeating groups, whereas the relation schema contains only simple attributes. By allowing an attribute name to be a composite type, the composite attribute supports both a detailed search and a multi-level search. For example, finding out all the persons with LastName "Smith" is a detailed search. A multi-level search would be finding all the names, which have "Smith" as either FirstName, MiddleName or LastName.

Each attribute has an one-to-one correspondence relation with the block defined in LDWG of this document type.

In general, each attribute has the following data members.

a. pageNum;

A page number is the identification of the page (of a document), in which, there exists a block having the attribute name.

b. blockName;

This name is the unique identifier of the block with the attribute name.

c. attributeName;

A user-defined, meaningful name represents well the logical meanings of the content in a block.

d. attributeType *T*i;

Attribute type *T*i is either one of the basic types or one of the common document attribute types. The basic types are number and string of characters. The common document attribute types are generic types, which are applicable to all kind of documents, such as date, name of a person, mailing address, email address and phone number. The common document attribute types are stored in the system's knowledge base as a part of background knowledge learnt from experiences. The system provides a tool, which allows users to review and update (delete, modify and add) these common types into the knowledge base. We will discuss this part in a later chapter.

e. whetherRepeat;

It is a boolean type to indicate whether this attribute is allowed to repeat.

f. weight;

It is a real number. It indicates the degree of importance of this block in classify a document into a type.

g. whetherSC;

It is a boolean type. If it is true, then only a simple attribute name and the attribute type are associated with a basic block, which is inseparable. If it is false, then the composite attribute can further be decomposed into sub-attributes according to the definition of attributes given in the childAttributes, which are, in turn, the objects of attribute class.

h. childAttributes;

They are the sub-attributes of an attribute of the composite type.

Figure 1.9 depicts an example of the internal representation of a logical structure for the layout structure shown in Figure 1.7.

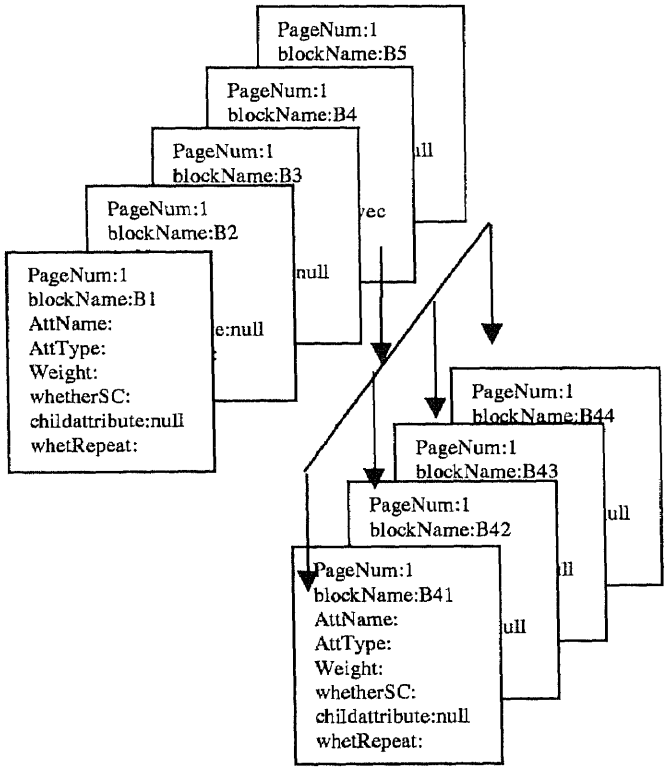


Figure 1.9 Internal representation for *logical structure*

We group together documents sharing the same frame template into a document class. The frame template is referred to as the document type (or simple type) of that class.

Definition 6: *(Document Type Hierarchy)* We organize frame templates as a hierarchical structure, which is called the *document type hierarchy* (DTH), based upon the generalization and specialization relation among frame templates and their inheritance properties among them. DTH is an one-to-many parent-children relation between document types. Each child has at most one parent. A child document type will inherit all the attributes from its parent.

Let A and B be the document types. For example, A is the memo type, and B is the meeting memo type. If type B inherits all the attributes defined in type A, then we say that type A is the parent of type B; and type B is a child of type A.

With the DTH, it is convenient for users to define a series of document types in a logically hierarchical relation, which is more manageable. The DTH plays a major role of retrieving documents in the browser and retrieval phase in the browsing subsystem [9, 10]. Say, if a user's inquiry is found to be related to one of the attribute names of a parent document type, then by default, this attribute name must occur in every its children document type. Consequently, this inquiry must query all the documents of these children document types.

A document type hierarchy describes the hierarchical relation over the document types from the logical point of view. The hierarchical relation may *not* be true for the layout structures of these document types. A hierarchical, logical relation over the

document is shown in Figure 1.10. For example, the meeting memo type, which is a child of the memo type, inherits all the attributes of the memo type. But the layout structures of documents of these document types may *not* hold a hierarchical relation for these documents with the hierarchical, logical relation.

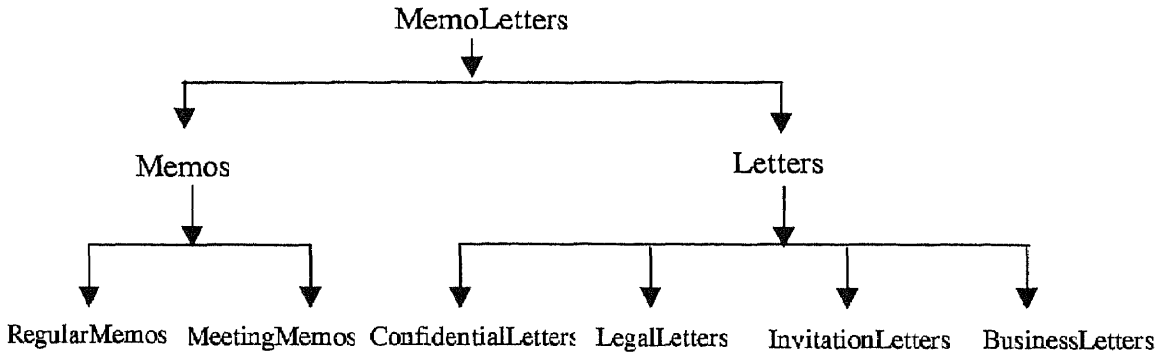


Figure 1.10 Example of a document type hierarchy

Definition 7: A *frame instance fi* of a document is an instance fi = [<A_1: I_1>, ..., <A_m:I_m>] over its document type representing by its frame template **F**=[<A_1:T_1>, ..., <A_m:T_m>], where A_i is an attribute; T_i is an attribute type and I_i is a value of the attribute type T_i extracted from the document.

A *frame instance* is an instantiation of a frame template, which represents the logical structure of a particular document under the logical definition of this document type. An example of a frame instance of the memo type is given in Figure 1.11.

1.2.5 System Scenarios

In the Automatic Document Classification and Extraction System (ADoCES), we use a *Labeled Directed Weighted Graph* to represent the *layout structure* and *frame template* to

Attribute Name		Value
Logo		IJPRAI
Address	StreetName	6 Mt. Hope
	CityName	Hull
	ZipCode	HU6 7RX
DepartmentName		CS
To		AI researchers
From		IJPRAI
Date		10/10/98
Subject		call for papers
Content		welcome
CC		M.A@comp.org

Figure 1.11 Example of frame instance of memo type

define the *logical structure* of a document. These are used as aids for classifying documents and extracting information from them. The *classification* of a given document consists of the following tasks:

1) Get the geometric structure based on the image file of the document, which is also called *document analysis*;
2) Transform the obtained geometric structure into the internal representation, called LDWG;
3) Find the document type for the given document by matching the graph of the given document with one of those sample graphs of the document type stored in the knowledge base. After the document type of the document is identified, it is assigned a unique *frame template*, representing the *logical structure* of the document.

In short, the input of the classification subsystem is the image file of a document (possibly generated from the OCR). The process of classification outputs a particular frame template representing the document type.

The document *extraction* subsystem has the following tasks: extract the proper content of a given a document based upon the layout structure of the document and the frame template, which represents the type of the document, and fill the content into each attribute defined in frame template. Therefore, the frame instance of this document is obtained. This is also called *document understanding*.

The ADoCES is a general-purpose system, which is domain-independent. In the very beginning, there is no specific knowledge for any particular domain in the knowledge base, only general knowledge applicable to all kinds of the documents, such as common data types. From the users' viewpoint, the system has two stages, the learning stage and operational stage as shown in Figure 1.12 and 1.13, respectively.

The learning stage is activated either at the very beginning of using this system by the users or when a given document failed to be identified as a type. During the learning stage, users use several document samples of the same document type to train the system in order to build up the specific knowledge and store the layout structures, the logical structure and the relation between the layout structure and logical structure in the knowledge base according to the document type. For the training purpose, users have to identify a document type for each of the documents; to verify the segmentation done by the system automatically; to define the logical equivalence among the blocks by using the same block name for those blocks in the different layout structures; and to give proper attribute names for the structured part of the document, which will be composed in the

logical definition of the document type. The system will save the multiple layout structures in the internal representation under this document type; build up the relation between the layout block and the logical definition; write the thesaurus and derive the global logical definition for this document.

The training document types are defined only after the users use this system for their specific purpose; therefore, it is domain independent. The knowledge stored in the knowledge base can be dumped and reloaded easily for further usage.

The reason the process failed to identify a document type for a given document is either that there is no sample layout structure of the document type in the knowledge base or there is no match found between the layout structure of the given document and the sample layout structures of all the document types. In this case, the learning stage of the ADoCES could also be activated to learn the recognition of the new case by means of training it by examples.

In the operational stage, with all the necessary knowledge in the knowledge base, the system doesn't need to interact with users. The system will do the classification and extraction automatically. First, the system will try to use the layout match for classifying a document. If a perfect match is found, then it will process the extraction based on the knowledge of relation between the layout structure and logical structure. If a perfect match is not found, it will try to match the logical equivalence of layout structures. If no match can be found, (it means that the current knowledge is inadequate for the system to do the correct classification and extraction), then the system needs to be re-trained with the unsolved document.

The following components of the ADoCES are used in the learning stage.

(1) Document analysis subsystem:

Initially, this subsystem is responsible for dividing the original document into a number of segments (or blocks), based on the "logical closeness". This process is called the segmentation. The output of the subsystem is the separated blocks within one document.

Learning stage:

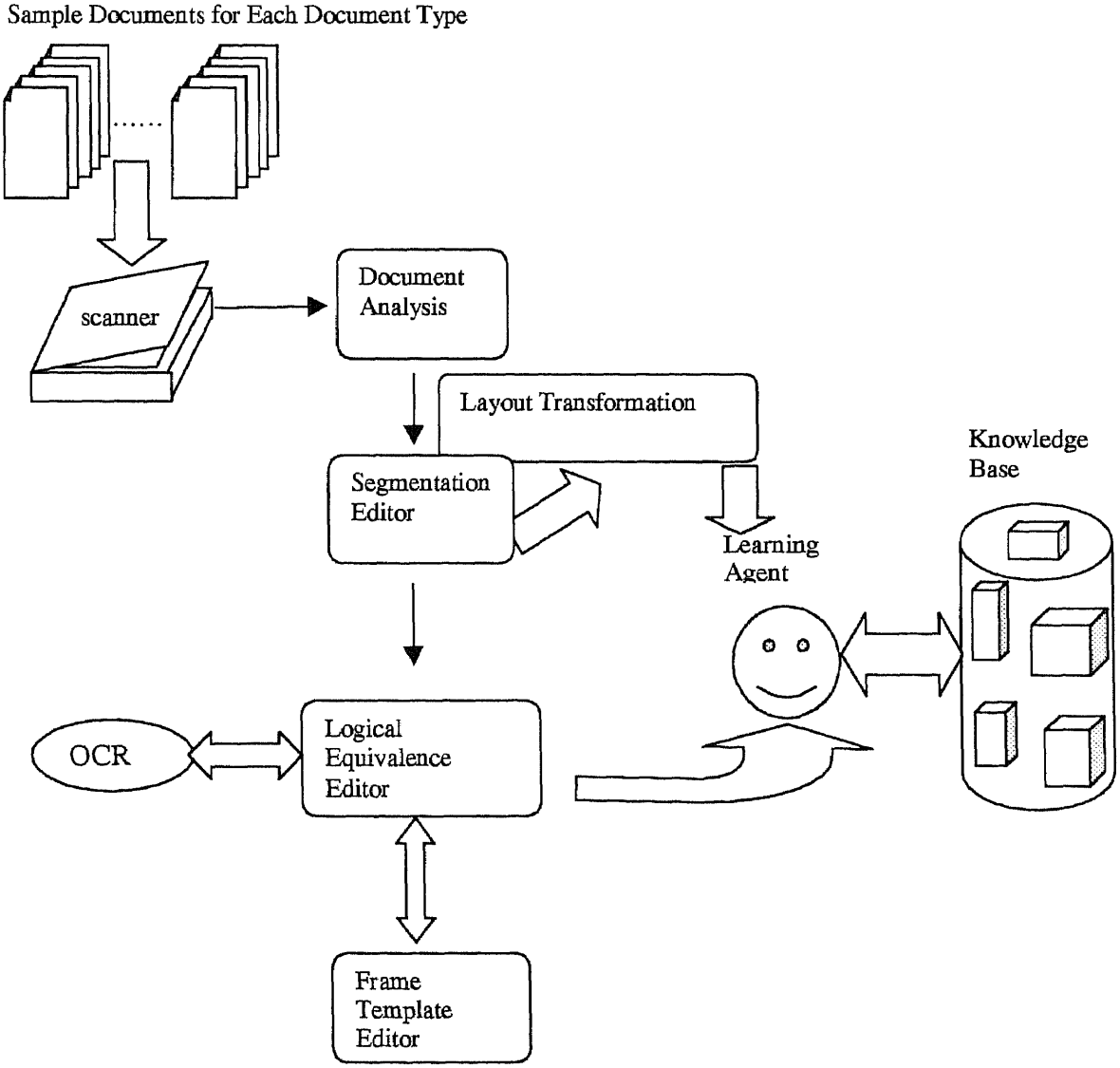


Figure 1.12 Learning stage

(2) Logical equivalence analysis subsystem:

This subsystem consists of a segmentation editor and a learning agent.

- segmentation editor:

  This segmentation editor is used if the document analysis subsystem does not generate a satisfactory output. It is also used if the structure of a document, which is a rare case, cannot be segmented correctly (or unsatisfactorily) by applying the general segmentation rule.

- segmentation learning agent:

  The segmentation learning agent is only activated whenever the segmentation editor is used. This learning agent maintains a history of the complete editing activities applied on a document. The incorrectly segmented document will be revised by applying recursively a sequence of editing operations on the previous ones. This agent will be activated whenever a document cannot be properly classified. The proper segmentation could be obtained after applying repeatedly the editing operations on the incorrectly segmented document. Thus, the document can be correctly classified.

(3) Logical equivalence defining subsystem:

It is used to define the "logical equivalent" parts of the different layout structures of the same document type and define the logical definition for each block after the segmentation. In addition, to construct the relation between the layout structure and the logical structure, these definitions for the blocks are used to derive the frame template for a document type automatically. We shall discuss this in detail in chapter 3.

(4) Layout transformation and synthesis subsystem:

This subsystem transforms the block segmentation of a document into a string representation and an internal representation -- Labeled Directed Weighted Graph. We discuss this subsystem in chapter 3.

(5) Frame template editor:

Once the logical equivalent editor has created the relation by assigning attributes of the logical structure to each block in the layout structure, the complete attribute names of this document type can be derived automatically. A user can use this frame template editor to display the global view of the logical structure of a document type, to give the detailed definition of a frame template, such as the data type of an attribute, and to set a flag to indicate whether an attribute is repeatable. However, this editor is not allowed to delete an existing attribute or add a new attribute into the frame template, in order to keep the relation between the layout and logical structure in a consistent manner.

(6) Learning agent:

In the learning stage, the system is trained by examples. The string representations, Labeled Directed Weighted Graphs and the document type definitions are stored in the system knowledge base, as the basis for the document classification and extraction. The learning agent is the only interface between the layout transformation and synthesis tool and the system knowledge base. The agent adds new knowledge into the knowledge base; and

is also activated whenever the system requires consulting the system knowledge base for additional knowledge.

Most components in the operational stage have the same functionalities as they have in the learning stage. The operational stage is used to classify the given document into its document type by finding an exact match, if possible, between its layout structure and a sample layout structure of the type, and by finding the frame template of its logical structure. We shall simply describe the classification and extraction subsystems that are only used in the operational stage.

(1) Classification subsystem:

In the operational stage, after getting the string representation and a LDWG of an incoming document to be classified, this classification subsystem uses three-level matching to match this incoming document with sample documents under each type stored in the system knowledge base. After finding a sample document which has the most "similar" match, the system assigns a particular type to this incoming document and then passes this result to the extraction subsystem to construct a frame instance. If the degree of "similarity", i.e. the degree of matching of two structures, is above the pre-defined threshold ("goodness") and the distances between the LDWG and the other sample documents are "far enough" ("uniqueness") away, then the type of this document is positively identified and the document is assigned the document type and the classification is completed successfully. If the degree of "similarity" is below the threshold, then the system will consult the segmentation history and try to revise the

Operational stage:

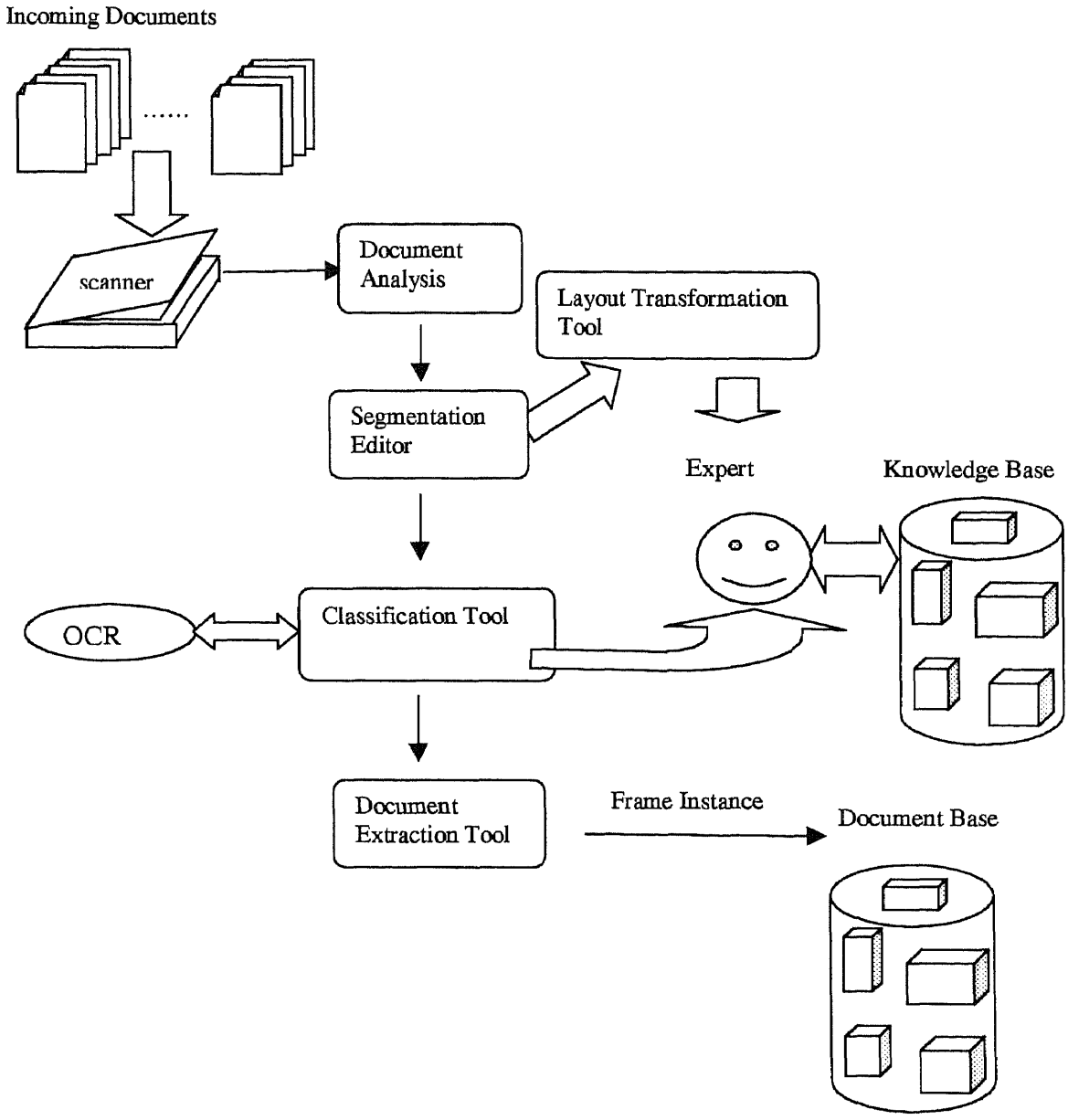


Figure 1.13 Operational stage of a system scenario

segmentation using the history of the editing activity on a previous similar document. If it helps, then this system still can solve this problem based on the existed knowledge. Otherwise, the system requires re-training to learn this new

case. This may result in a new layout structure to be added into the knowledge base under a document type or a totally new document type with the new layout structure and logical structure to be added into the knowledge base. If the distances between the LDWG and the other samples are not "far enough" apart, which means that the "uniqueness" is not very sure, then the system consults the "similarity table", which stores the "similarity" between any pair of existing document types.

(2) Extraction subsystem:

After the subsystem has identified a Labeled Directed Weighted Graph of the document type and has assigned the document with a document type, which are combined to be the best mask for this incoming document, the incoming document enters in the extraction stage. Then the extraction is proceeded automatically simply based on the relation between the layout structure and the logical structure.

1.3 Organization of this Dissertation

A methodology for segmenting a document based on the "logical closeness" is presented in chapter 2. In chapter 3, we shall give a detailed discussion on the internal representation of the layout structure, including the definition of the LDWG of a document, the methodology of transforming a document into its corresponding LDWG after the segmentation. The logical representation of the document will be covered in chapter 4. The rules of classifying a document will be given in chapter 5. We propose the content extraction of the document in chapter 6. Knowledge base architecture is

discussed in chapter 7. In chapter 8, we will summarize our research achievement and propose future research.

CHAPTER 2

LOGICAL EQUIVALENCE ANALYSIS

Human beings are used to grouping related information together. An author of an article uses the same font, format and even spacing between lines for information which are logically related and equivalent. In the same way, the readers of this article understand perfectly this formation rule used by the author. Therefore, a reader can group lines of contents together to form regions; pay attention to various parts of interest and skip some parts, region by region, of the article. It is very difficult for the reader to track the information selectively if the article is written in one font or in a plain text, line after line. This analysis suggests that it is possible and meaningful for us to do the logical segmentation by taking into account this rich text information.

In general, the reader divides a given text into segments (blocks), each of which satisfies the predefined logical closeness, without reading and understanding the content of the text. We can do it only by common sense. Therefore, this is a general approach applicable to all structured documents. In addition, this approach of dividing any given text into segments can be done automatically, augmented with capabilities for users to edit the segmentation or correct the erroneous results based on the automatic segmentation result. During the correcting procedure, a segmentation learning agent records the history of editing the analysis result (done by the user) in the knowledge base. This historical knowledge could be applied in the next segmentation of the same document type.

A document image is composed of several blocks, each of which represents a coherent component of the document. The coherent component means a set of text lines with the same typefaced font and a constant line interval.

2.1 Logical Closeness Analysis

The task of segmentation is to separate the original document image file into several rectangular areas, also called *blocks*. A block is the smallest unit of maximal homogenous area, such as text, graphics and image. Trying to find the best methodology to use for segmentation is one of the most widely discussed and the most important topics in document processing research. Several methods are proposed for solving this problem. Among those methods, the Run Length Smoothing Algorithm (RLSA) [51] was the most popular and widely referred method in the initial analysis of the document image file. Some researchers proposed their methods based on RLSA for improving the efficiency [43]. A knowledge based method [48] was also proposed to divide the image automatically into nested rectangles corresponding to meaningful blocks. Researchers in Japan [99] proposed using the Neighborhood Line Density (NLD) to do the character or graphic segmentation based on the phenomenon that "Characters consist of many more strokes than graphics; consequently stroke density is high. On the other hand, graphics strokes are isolated from each other." Recently, a way of using fractal signatures [100] was proposed in document analysis to separate an image into several blocks. A.K. Jain and B. Yu [98] proposed a bottom-up approach to implementing efficiently page segmentation and region identification based on the connected component extraction. This method can accommodate moderate amounts of skew and noise. In [2], an example

of understanding of a newspaper is given. The layout structure and logical structure are represented in terms of tree representations. The layout structure can be transformed into logical structure, using given four transformation rules.

Basically, those approaches can be separated into two kinds. One is the top-down approach, which divides the document into major regions and each of the major regions is further divided into sub-regions based on a predefined segmentation criteria. The other is the bottom-up approach, which progressively refines the data by layered grouping operations. In general, the top-down approach is much more appropriate for processing "neat" documents that have a specific format such as memos, letters and papers, but it is not good enough to deal with documents of irregular format, such as a newspaper. This approach is relatively fast and effective. On the other hand, the bottom-up approach is time-consuming. However, because it analyzes the image components from the most inner regions to the outer regions, this method is applicable to many document domains and it is possible to develop methods that are applicable to a variety of documents.

These methods process documents on their corresponding original image files. Their aim is to separate the file into blocks based on the lining spacing, gray levels, pixels and this kind of original and physical information. In the real world, the line spacing is an important criterion, which indicates the closeness between two adjacent blocks. However line spacing can not be used as the only criterion to the segmentation because authors may adjust the spacing of the document based on their own opinion of aesthetics or particular purpose, such as emphasizing a part of the content of this document. Therefore, a general segmentation only based on the physical information is not good enough to be used later by the other components of the document processing. It is essential to analyze

their logical closeness. Discussing another way of segmentation is not our focus of this dissertation. Segmentation is the very first step of document processing, and its role is to serve the later processing. We propose an image file of a given document be segmented based on the logical closeness of the contents. General segmentation will be enhanced greatly if it proceeds further analysis on the logical relationship of the contents in each block. Optical Character Recognition (OCR) technology recently has become efficient and effective. From the output of OCR, we can obtain easily the rich text and separated blocks. The rich text information including the text with its font and attributes can be used to conduct further analysis on the text and furthermore to find the "structured" parts of the document.

In order to do segmentation of a given document based on the logical grouping, the text needs to be post-processed after the original document has been read by the OCR. In brief, this automatic segmentation component consists of two major tasks.

Input: Output information from the OCR package.

For text only (ignoring graphics and image regions), further process is as follows:

a. Group all the lines with the same font and even line intervals as much as possible to form the initial segmentation.

b. For adjacent areas, do the logical association analysis:

   1) combine the lines to form a region if they are in the same format;

   2) combine adjacent text regions into one.

For each line, we search for special characters, such as the separators, except those characters from "A" through "Z", from "a" through "z", from "0" through "9","?" and "." The characters such as ":","●"..., are considered to be potential separators.

This method, which is a kind of bottom-up method, is used to get the logical unit from the document. The data structure of OCR is shown in Figure 2.1. (Rather than developing OCR by ourselves, we used the OCR developed by MaxSoft™ .) From this structure, we can get detailed information, such as the height, coordinate, attribute (*italic*, underline or **bold**), about each character, each word in a line and each line of the text.

2.2 Segmentation Methodology

The OCR package outputs the "reading" result in a hierarchical structure. Firstly, it has the block type. Different processing techniques will be applied for different block types. According to the contents of the blocks, there are several block types: TEXT_BLOCK, IMAGE_BLOCK, TABLE_BLOCK, GRAPHICS_BLOCK and OTHER_BLOCK.

For TEXT_BLOCK, proceed with post_processing - analysis to determine the logical closeness and to divide the block further according to their potential meanings.

If the block type is text, then further information is stored as line, word and character.

After OCR, the system gets the lines of texts (characters, words, etc.) and the height (character size) of characters, and font (*italic*, underline, **bold** and font type (serif, sans serif, mono)) of characters.

A GeneralFontHeightTable (GFH_table) in the descending order of height is created as Figure 2.2. In this table, the font sizes used in all the TEXT_BLOCKS are sorted in a descending order at first. That is from the larger size down to smaller size.

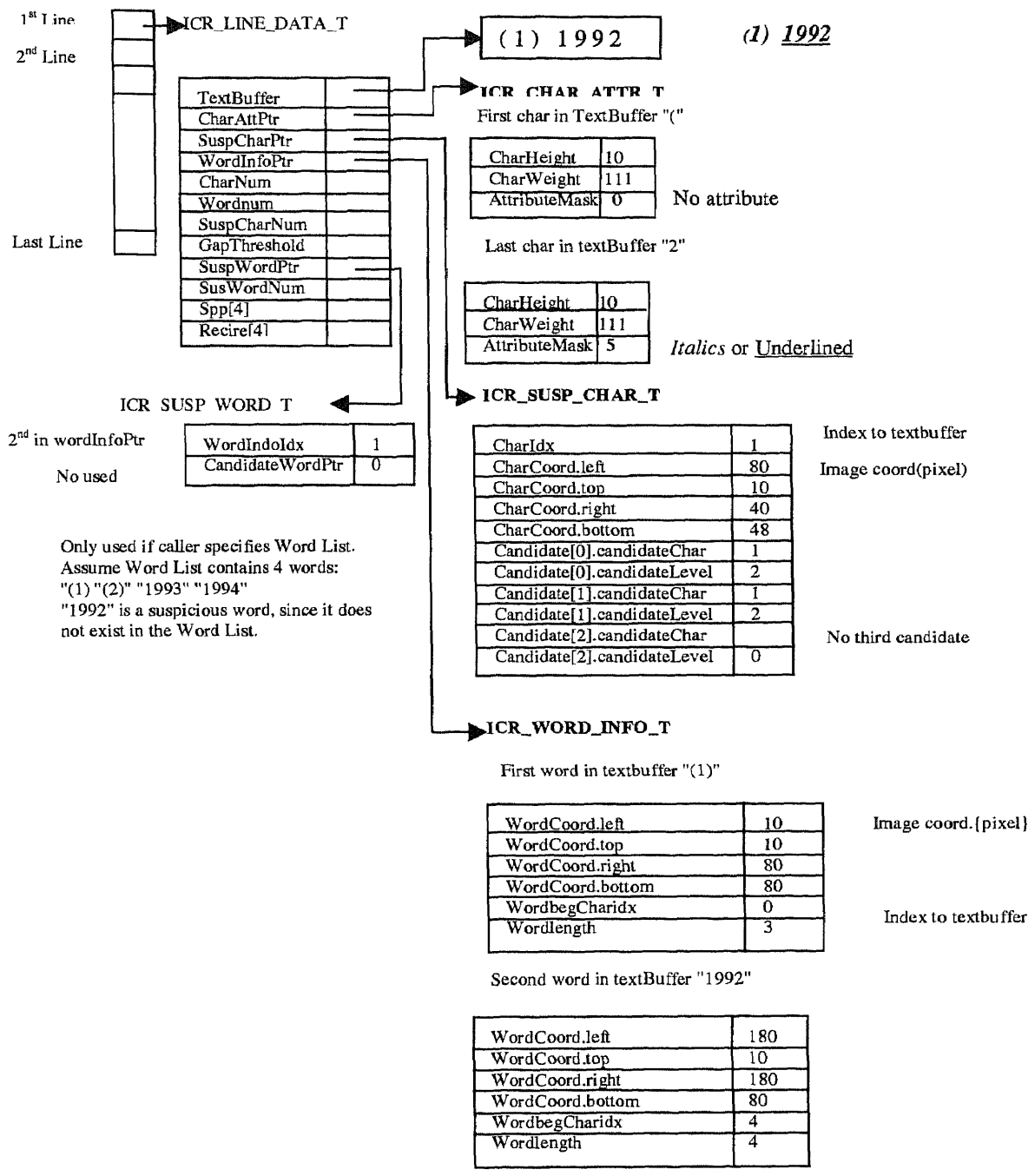


Figure 2.1 Data structure maintained by OCR package

Second, within the same font size, attributes are sorted in order of **bold**, *italic* and underline.

Analyzing the texts, the corresponding blocks are divided further into FREE_TEXT_BLOCK and ATTRIBUTE_TEXT_BLOCK.

H	F	A
14		1
13	B	2
12	I	3
10		4
8		5
6		6

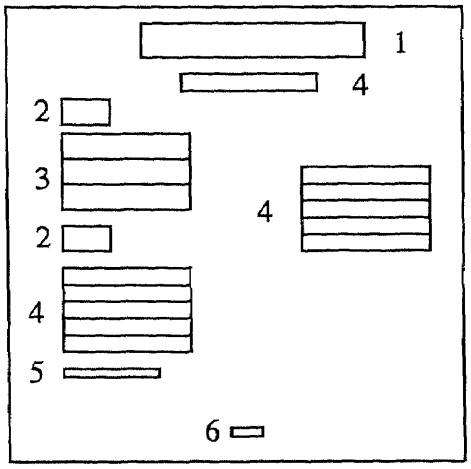


Figure 2.2 GeneralFontHeight Table

The *FREE_TEXT_BLOCK* consists of the information about first_line_text, full_line_text and last_line_text, as shown in Figure 2.3.

If the lines are of the same height and same font and have even spacing and they are holding the pattern of first_line_text(full_line_text)*last_line_text, then these lines are combined into one block and this block is classified as FREE_TEXT_BLOCK.

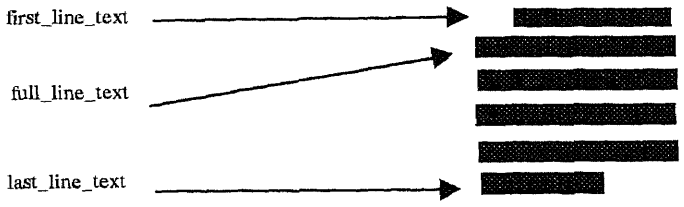


Figure 2.3 Pattern of FREE_TEXT_BLOCK

ATTRIBUTE_TEXT_BLOCK needs further analysis. All the lines which are in the same fonts and height, are combined together.

For the block types IMAGE_BLOCK and GRAPHICS_BLOCK there are three attributes to be stored.

1. Caption: The caption of an image or graphics is extracted from the document automatically.
2. Description: A user enters a brief description of his/her view of the image or graphics.
3. Signature: The signature of an image block or a graphic block is defined in terms of x-signature and y-signature, which are used to store the attribute values of this image or graphics. The signature can be used to compare two images without the need of comparing them pixel by pixel. An advantage of using the signature is that the system does not have to store the original image and therefore space efficiency can be achieved. The other advantage is that for matching two images the system does not need to compare the corresponding pixels of two images, which would be time-consuming.

The signature is used only for a block which is **stationary**, such as a logo block, which appears consistently in the upper left of a letter. The caption and description are used for both stationary and non-stationary graphic and image blocks.

The method of computing the signature of an image must be simple and not time-consuming. There must be a unique way of representing the original image, not influenced by a little noise.

Our method of finding the signature of an image is as follows. Generally speaking, an image or a graphics is projected on X-axis and Y-axis, and then we find a function of the variations.

As shown in Figure 2.4, an interpolation method could be used to find the function based on the discrete values obtained from the image itself. The X-signature of this image is $a_1x^4+b_1x^3+c_1x^2+d_1x+e_1$ and the Y-signature of this image is $a_2y^4+b_2y^3+c_2y^2+d_2y+e_2$.

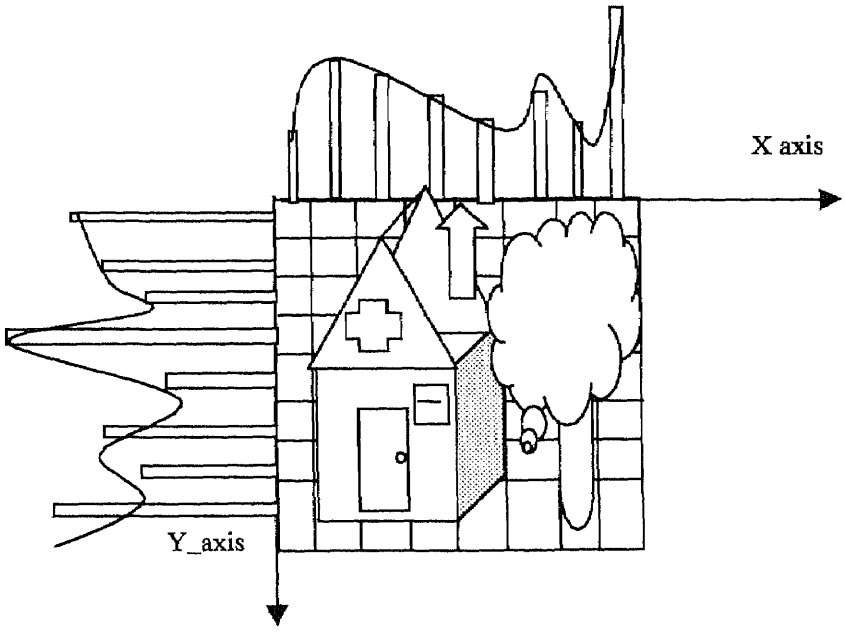


Figure 2.4 Projection of the image

Finally, for the block type, TABLE_BLOCK, we simply store the caption of the table.

We give the data structure for different block types as in Figure 2.5.

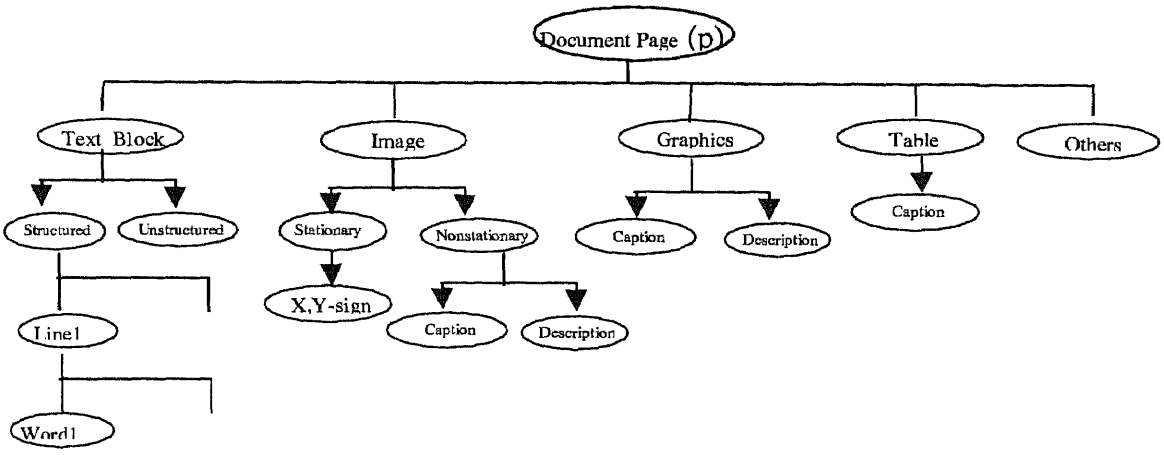


Figure 2.5 Data structure for different block type

2.3 Logical Equivalence Editor

The logical equivalence editor is used only during the learning stage. It is the main tool used to make a "supervised learning". During a user's usage of this tool, we provide an easy and convenient way for the user to train the system. In the background, the system "learns" and stores the knowledge in the system knowledge base. This editor is the only interface for users to define the relation between the layout structure and logical structure.

Figure 2.6 illustrates an example of defining the logical equivalence of two different layout structures of the same document type obtained after segmentation. First, users select the type which they want to define for this time, all the layout structures under this document type show up in the editor. Second, users only need to use the *same* color to fill out the logical equivalent part of these two different layouts and then give an attribute name for each different logical unit. An example of JournalArticle is given in Figure 2.7.

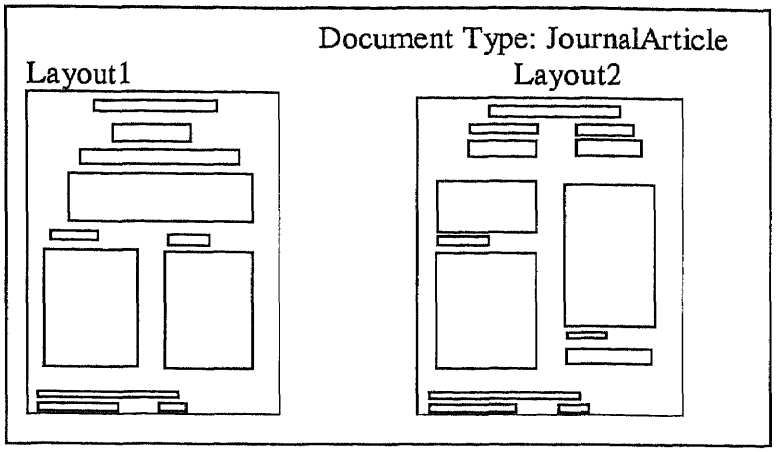


Figure 2.6 Logical equivalence editor interface

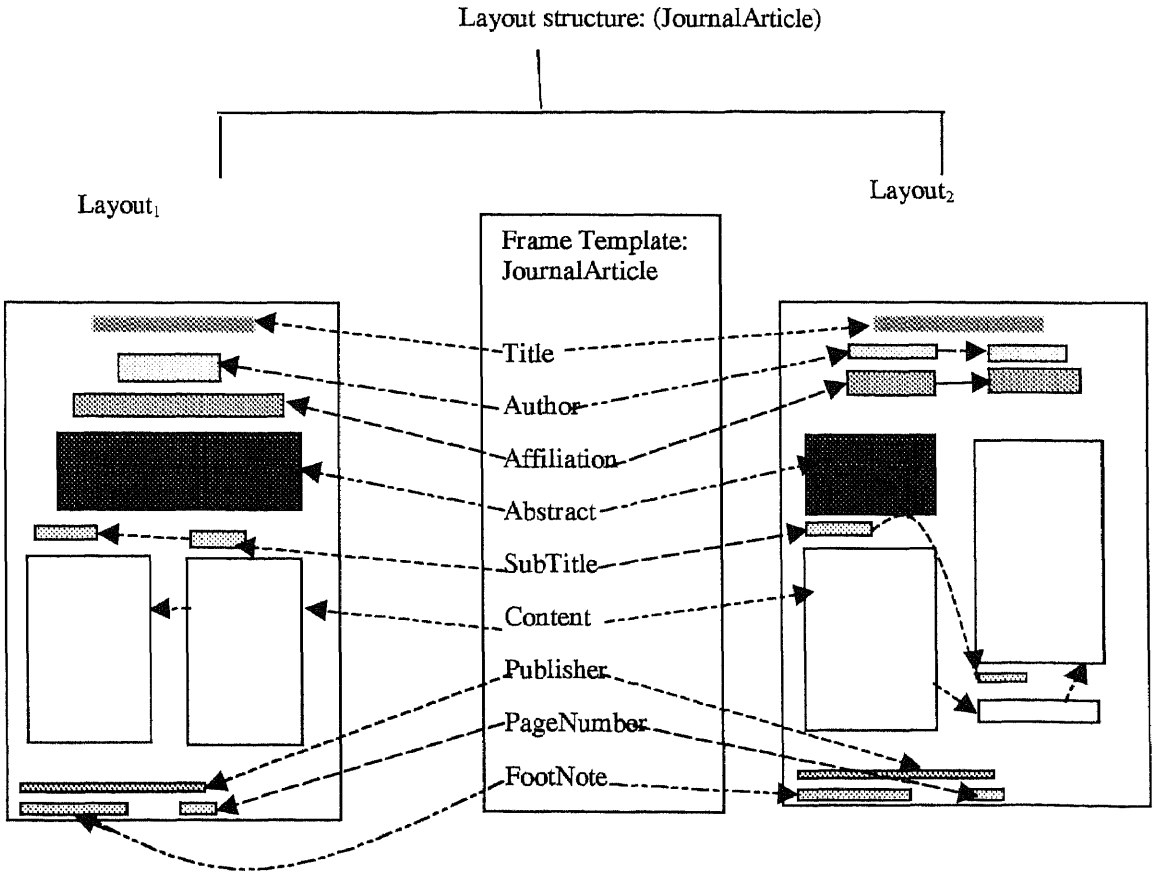


Figure 2.7 Defining of the relation between layout structure and logical structure

2.4 Recording the Segmentation Editing History

In the system scenarios for the learning stage as shown in Figure 1.12, the logical segmentation based on the results of the OCR may still be incorrect and no segmentation criterion can be omnipotent to solve segmentation for all documents. Therefore, we propose a way to compensate if a rare case occurs. By recording the editing history of the users, the system has this mechanism to learn from the interaction with the end users. In order to meet the later requirement on logical match within a block, we allows users to use the segment editor to redefine the segmentation area, such as, combining two blocks into one or splitting one block into several blocks.

For example, assume that the process of segmentation yields initially six blocks as shown in Figure 2.8 and this segmentation result doesn't yield the correct logical structure of this document type. Then the users may use the segmentation editor to do a sequence of editing.

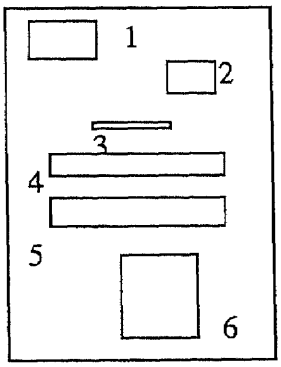


Figure 2.8 Result of initial segmentation

Assume that the editing history is as follows:

. combine block 4 and 5

. split block 6

Then the new blocks are as shown in Figure 2.9.

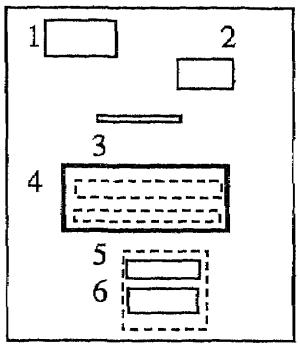


Figure 2.9 Segmentation result after editing

The dot lines represent the initial blocks and the lines represent the blocks after editing.

During editing the blocks, a user may "undo" any editing. An undo is an operation used to restore the original structure before the previous operation applied to the structure, based on the history of segmentation editing. Before the history is saved into the database, we should simplify the history by erasing those undos and the corresponding contents of undos.

For example, if the sequences of operations, performed by the user are as following:

Op 1

Op 2

undo

Op 3

The final result should be only Op1 and Op3.

For example, Op 1 is "combine block 4 and 5", Op 2 is "split block 6"; the undo means un-do the last operation, then after undo we only get the Op 1, which means "combine block 4 and 5". Similarly, the following operation sequences should get Op3.

Op 1

Op 2

undo

undo

Op 3

CHAPTER 3

DOCUMENT LAYOUT REPRESENTATION

Layout structure representations are digital forms of the stored original document images. This not only saves the store space, but also provides a good structure suitable for further analysis and processing such as document classification and extraction.

One of the most important laws abided by the author and readers is the similarity law. That is, a human being has the tendency to group the similar components together. In chapter 2, we have discussed our methodology of segmentation. In the document segmentation procedure, an image document is transformed into a specification of the geometry of the maximal homogeneous regions (logical units). In the mean time, each region is classified into (and labeled as) a particular type, such as text, image, graphics or table.

In this chapter, we shall present an appropriate and efficient representation of the blocks and their spatial relations in a document image file.

3.1 Internal Layout Representation

A geometric page layout of a document image is a specification of the geometry of the maximal homogenous regions and the spatial relations of these regions. Formally, a geometric page layout P = (R, S), where R is a set of regions, and S is a set of labeled spatial relations on the region set R.

A binary image of a page is represented as a Labeled Directed Weighted Graph (LDWG), which is defined as LDWG=(N, E), where N={N_i} is a set of blocks, and

$E=\{e(N_i, N_j, H/V/D)\}$ is the set of edges labeled as H, V or D relations between two blocks to indicate the spatial relation between block N_i and N_j.

Depending on different applications, the definitions for region *R* are not the same. Usually, regions are defined as *rectangle* for regular regions or *polygon* for irregular regions. Irregular regions usually show up in newspapers or magazines for special needs to attract the reader's attention. But in office documents, it suffices to use rectangles to represent the regions. Considering a rectangular region, there are several specifications. First, a rectangle can be expressed in terms of a starting point, length and width; second, a rectangle can also be represented by the left-top point and the right-bottom point.

In our application, the coordinates of the rectangle are used in order to compute the relative position between two regions. Therefore, the second specification is more suitable in our case. We define a rectangle in term of (X_{min}, Y_{min}), the coordination of the left_top point, and (X_{max}, Y_{max}), the coordination of the right_bottom points, as shown in Figure 3.1.

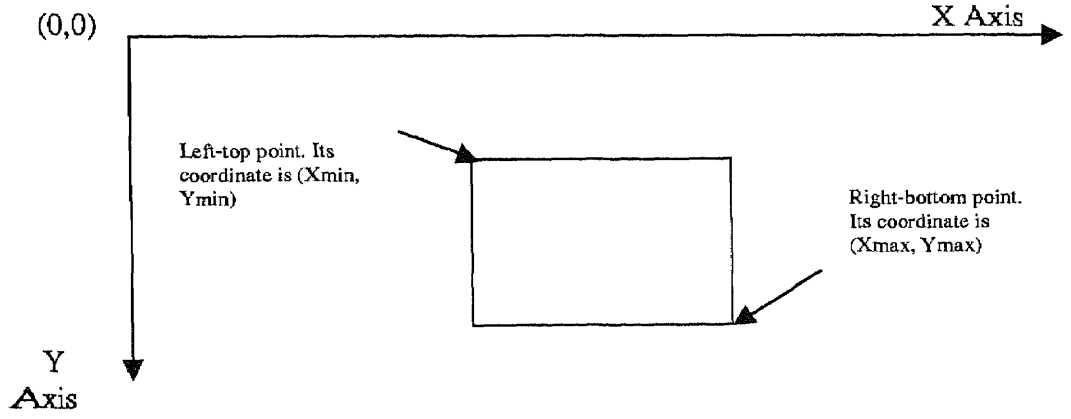


Figure 3.1 Rectangle definition

The labeled spatial relation S is a set of triples (R_i, R_j, L), where R_i is one region and the R_j is the second region and L is a label indicating the spatial relation of these two regions. The triple says that region R_i has a relation L with respect to R_j.

Similarly, depending on the different applications, the spatial relation may be defined in different ways. Some applications define the simple spatial relation, such as overlap, exclusive and inside, to indicate their relative positions; some applications record the difference on x-axis or y-axis of these two blocks besides their relative position.

Based on documents collected from the offices, experiments show that only the relative position is useful for classifying a document to a particular type. Therefore, the system doesn't need to record the difference of the two rectangles in x-axis or y-axis. And two areas rarely have the overlap relation. A hierarchical relation is used instead of the inside relation, and the exclusive relation is further divided into the horizontal, vertical and diagonal relations.

3.1.1 Relative Locations of Segmented Blocks in a Layout Structure

Given any document, its layout structure consists of the following information. Each rectangular block has a pair of (X, Y)-coordinates to represent its positional information. The (X, Y)-coordinates of a rectangular block are the pair of its starting point (X_{min}, Y_{min}) and its end point (X_{max}, Y_{max}), where $X_{max} = X_{min}$ + the width of the rectangular block; and $Y_{max} = Y_{min}$ + the height of the rectangular block.

Let {A_X_{min}, A_X_{max}}and {B_X_{min}, B_X_{max}} be the x-coordinates of blocks A and B, respectively. Let { A_Y_{min}, A_Y_{max} } and { B_Y_{min}, B_Y_{max} } be the y-coordinates of blocks A and B, respectively.

Definition 1: Blocks A and B are in a *horizontal* position and A is on the left of B if and only if

$A\_X_{max} \leq B\_X_{min}$ and (($A\_Y_{min} \geq B\_Y_{min}$ and $A\_Y_{min} \leq B\_Y_{max}$) or ($A\_Y_{min} \leq B\_Y_{min}$ and $A\_Y_{max} \geq B\_Y_{min}$)).

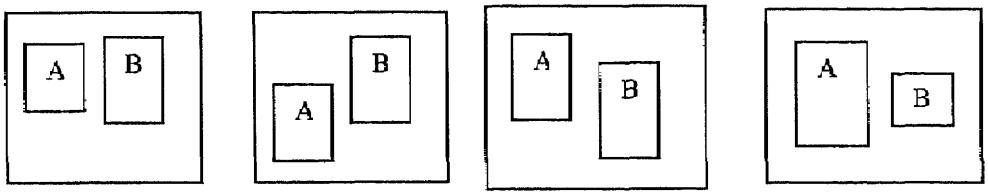


Figure 3.2 Four cases of horizontal position

The condition $A\_X_{max} \leq B\_X_{min}$ guarantees that Block A is on the left of B. The condition ($A\_Y_{min} \geq B\_Y_{min}$) and ($A\_Y_{min} \leq B\_Y_{max}$) specifies the first two cases and rules out that the B is completely above the block A, which is also known as diagonal case (as shown in Figure 3.3 a). Likewise, the condition ($A\_Y_{min} \leq B\_Y_{min}$) and ($A\_Y_{max} \geq B\_Y_{min}$) specifies the next two cases and rules out the block A is completely above the block B, which is also known as diagonal case (as shown in Figure 3.3 b).

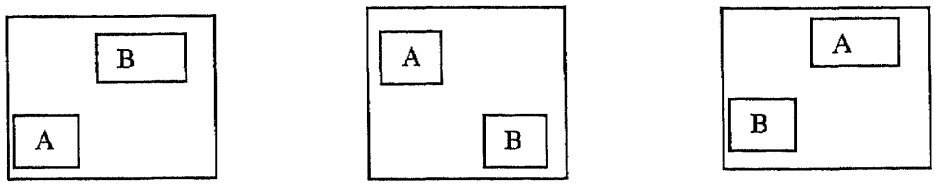


Figure 3.3 Ruled out cases

Definition 2: Blocks A and B are in a *vertical* position and A is on the top of B if and only if

$A\_Y_{max} \leq B\_Y_{min}$ and (($A\_X_{min} \leq B\_X_{min}$ and $A\_X_{max} \geq B\_X_{min}$) or ($A\_X_{min} \geq B\_X_{min}$ and $A\_X_{min} \leq B\_X_{max}$)).

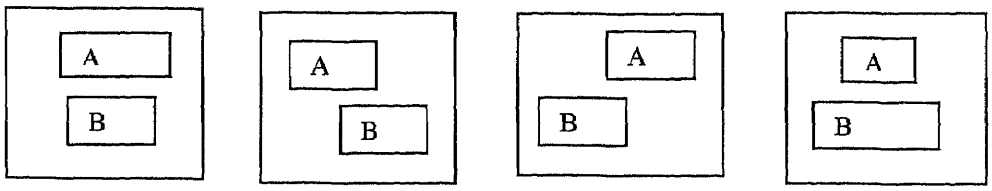


Figure 3.4 Four cases of vertical position

The conditions ($A\_X_{min} \leq B\_X_{min}$ and $A\_X_{max} \geq B\_X_{min}$) and ($A\_X_{min} \geq B\_X_{min}$ and $A\_X_{min} \leq B\_X_{max}$) rule out the diagonal cases as shown in Figure 3.3 b and Figure 3.3 c. The first condition guarantees that the block A cannot be completely on the left side of the block B. The second condition guarantees that the block A can not be completely on the right side of the block B. A is on top of B for $A\_Y_{max} \leq B\_Y_{min}$.

Definition 3: Blocks A and B are in a *diagonal* position and A is on the *upper_left* of B (called D_L relation) if and only if

$A\_X_{max} \leq B\_X_{min}$ and $A\_Y_{max} \leq B\_Y_{min}$.

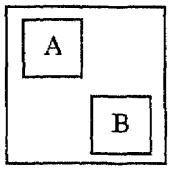


Figure 3.5 Diagonal position (upper_left)

Definition 4: Blocks A and B are *diagonal* position and A is on the *upper_right* of B(D_R relation) if and only if

$A\_X_{min} \geq B\_X_{max}$ and $A\_Y_{max} \leq B\_Y_{min}$.

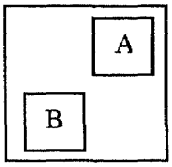


Figure 3.6 Diagonal position (upper_right)

Definition 5(*H-adjacent*): We call the two blocks A (A_X_{min}, A_Y_{min}, A_X_{max}, A_Y_{max}) and B (B_X_{min}, B_Y_{min}, B_X_{max}, B_Y_{max}) are *H-adjacent* if and only if:

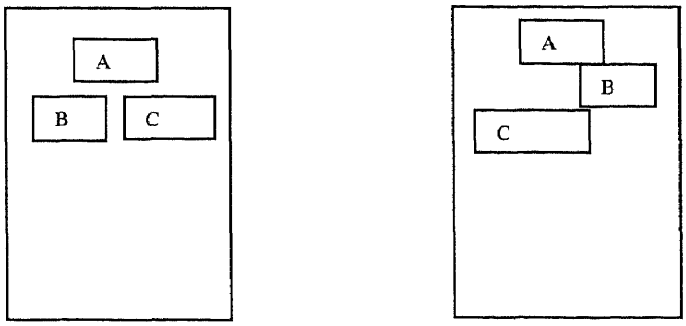
1) They are in a horizontal position; and
2) There is no other block, say block C(C_X_{min}, C_Y_{min}, C_X_{max}, C_Y_{max}), such that A and C are in horizontal position and C and B are also in horizontal position.

Definition 6(*V-adjacent*): We call the two blocks A (A_X_{min}, A_Y_{min}, A_X_{max}, A_Y_{max}) and B (B_X_{min}, B_Y_{min}, B_X_{max}, B_Y_{max}) are *V-adjacent* if and only if:

1) They are in a vertical position; and
2) There is no other block, say block C(C_X_{min}, C_Y_{min}, C_X_{max}, C_Y_{max}), such that A and C are in vertical position, and C and B are also in vertical position.

Definition 5 states that a block A is horizontally adjacent to block B if and only if they are in a horizontal position without any block between them. Definition 6 states that

a block A is vertically adjacent to block B if and only if they are in a vertically position without any block between them. In Figure 3.7(a), A and B are V-adjacent, A and C are V_adjacent and B and C are H-adjacent. In Figure 3.7(b), A and B are V-adjacent and B and C are V-adjacent, but A and C are not V-adjacent.



(a) (b)

Figure 3.7 V-adjacent blocks

Definition 7: *(Gray area)* Let Blocks A and B be in a diagonal position. If A is on the upper_left of B, then the gray area of the blocks A and B (denoted as gray_area(A,B)) is an area with the upper_left coordinate (A_X_{max}, A_Y_{max}) and the bottom_right coordinate (B_X_{min}, B_Y_{min}) as shown in Figure 3.8(a).

If A is on the upper_right of B then the coordinate of the gray area will be (B_X_{max}, A_Y_{max}) for its upper_left coordinate and (A_X_{min}, B_Y_{min}) for its bottom_right coordinate as shown in Figure 3.8(b).

Definition 8*: (D_L adjacent)* Blocks A and B are D_L adjacent if the blocks A and B have D_L relation and there is no other block that touches (lays on) their gray area (denoted as gray_area (A, B) in Figure 3.8 a).

Definition 9: (*D_R adjacent*) Blocks A and B are D_R adjacent if the blocks A and B have D_R relation and there is no other block that touches (lays on) their gray area (denoted as gray_area (A, B) in Figure 3.8 b).

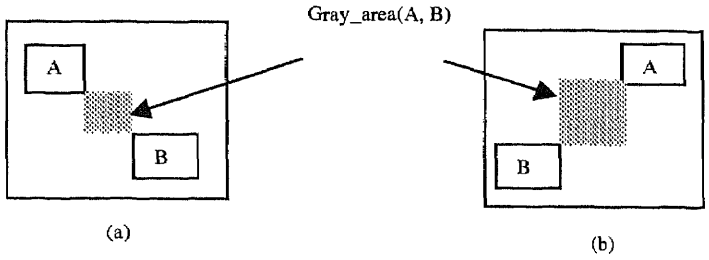


Figure 3.8 Gray area of two diagonal blocks

In the reminder of this section, we shall give several examples to illustrate the definitions of adjacent relations and the gray_area.

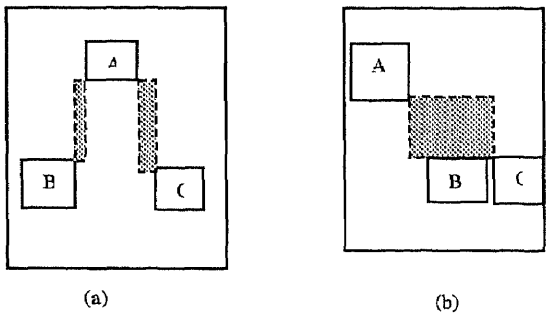


Figure 3.9 Adjacent blocks (Example 1)

In Figure 3.9 (a), since there is no block which touches the gray_area(A, B) and the gray_area(A, C), A is D_R-adjacent to B and A is D_L-adjacent to C respectively. However, B is H-adjacent with C and A is not V-adjacent to B or C. In Figure 3.9 (b), since there is no block, which touches the gray_area(A, B), A is D_L-adjacent to B. B is

H-adjacent to C. A is not D_L-adjacent to C, because block B touches the gray_area(A, C).

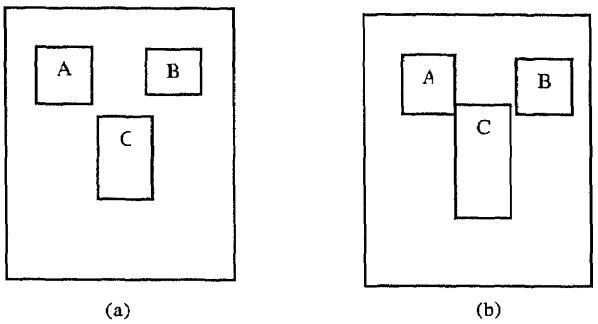


Figure 3.10 Adjacent blocks (Example 2)

In Figure 3.10 (a), A is D_L-adjacent to C because there is no block which touches the gray_area(A, C). A is H-adjacent to B. B is V-adjacent to C. In Figure 3.10 (b), A is H-adjacent to C and C is H-adjacent to B, but A is not adjacent to B.

In the Figure 3.11(a), A is D_L-adjacent to C and A is H-adjacent to B; B is D_R-adjacent to C. In the Figure 3.11 (b), A is H-adjacent to B and A is V-adjacent to C; B is V-adjacent to C.

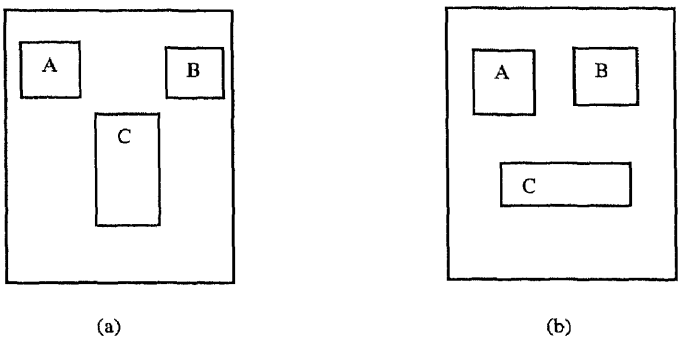


Figure 3.11 Adjacent blocks (Example 3)

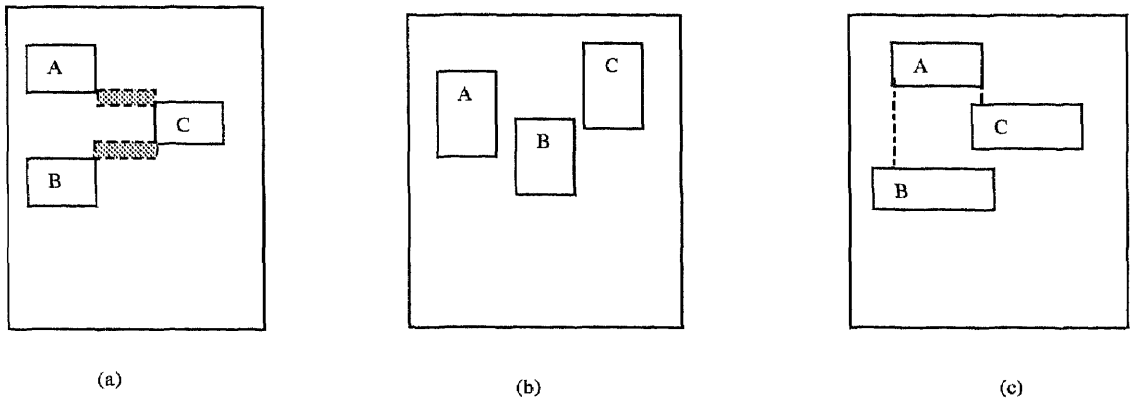


Figure 3.12 Adjacent blocks (Example 4)

In the Figure 3.12(a), A is D_L-adjacent to C and C is D_R-adjacent to B.

In the above Figure 3.12 (b), A is H_adjacent to B, B is H_adjacent to C, but A is not H_adjacent to C. In the above Figure 3.12 (c), A is V_adjacent to C, C is V_adjacent to B, but A is not V_adjacent to B.

In the Figure 3.13 (a), A is H-adjacent to B and B is H-adjacent to C, but A is not H-adjacent to C. In Figure 3.13 (b), A is H-adjacent to B. B is D_L-adjacent to C because no block touches the gray_area(B, C). However, A fails to be D_L-adjacent to C, because B touches the gray_area(A, C).

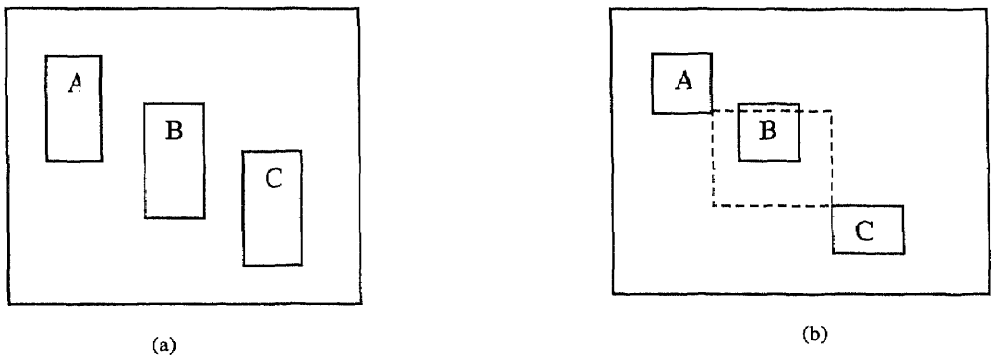


Figure 3.13 Adjacent blocks (Example 5)

3.1.2 Attributes of the Block

Given a document, it is segmented into various blocks to form its layout structure. Each block has the following data members:

a. (X_{min}, Y_{min}) and (X_{max}, Y_{max})

They are used to record the absolute position of a block in a page. (X_{min}, Y_{min}) is the coordinate of the absolute leftmost top position of a block in a physical page. (X_{max}, Y_{max}) is the coordinate of the absolute rightmost bottom position of a block in a physical page. They are used to derive the relation between the blocks, the width and the height of the block or realigning the original image.

b. blockName

It is the unique identifier of a block. This name is not allowed to be reused in the same document.

c. weight

It is only used during the match for the classification purpose. The weight of a block is to determinate the significant factor of the block in a document type.

d. type

It is the type of this block. It may be one of the following types: text, image, graphics and table.

e. whetherSC

It is a boolean type to indicate this block is either a simple block or a composite block. If it is a composite one, then the blockNames of its sub-blocks are stored as the member of its childBlocks.

f. childBlocks

As stated above, it contains the further segmentation information (i.e. the block name of all the subblocks) of the current block.

g. stationary

When the block type is image or graphics type, this boolean is used to indicate whether this block has an absolute position and size in this document. Some blocks which contain such as logos or images may be stationary and play an extraordinary role in classifying a document as a particular type. A block is considered to be non-stationary, if its size is flexible, its position is not stationary, or its occurrence is optional.

h. caption

Besides tables, it is used when the block type is either graphic or image which is non-stationary (i.e. the stationary is false). This means that only those non-stationary images or graphics and tables within the document are entitled to have a caption, which usually appears below or above the graphic or image. If there is no caption in the original document, then this field will remain blank.

i. descriptions

This item is also used when the block type is either a graphic or an image, which is non-stationary (i.e. the stationary is false). But the difference between the item caption and item description is that the content of description is entered by users. The content of the description for an image or graphic contains the subjective expression of the user, after seeing the image. We provide this item since there is no good and effective way to do the image

match, this item description is provided to allow users to enter some keywords, which could be used to retrieve this image later on.

j. x_signature and y_signature

These two items are in fact two functions which represent the projection of the original image on X-axis and Y-axis respectively. These two items are used for the stationary graphics and image type (i.e. the stationary is true), such as a logo or a little image which is always in a specific position.

3.1.3 Representing Document Layout Information

From the output of the segmentation, the (X, Y)-coordinates of each block can be obtained. Each rectangular block has a pair of (X, Y)-coordinates to represent its positional information. The (x,y)-coordinates of a rectangular are the pair of its starting point(X_{min}, Y_{min}) and its end point (X_{max}, Y_{max}), where X_{max} = X_{min} + the width of the rectangular block; Y_{max} = Y_{min} + the height of the rectangular block. In this section, we shall describe a way of representing the layout structure of this document in terms of an internal data representation, which we called the Labeled Directed Weighted Graph (LDWG). For example, given the documents as shown in Figure 3.7a and 3.7b (on page 61), their layout structures can be represented by the LDWGs as shown in Figure 3.14 (a) and 3.14 (b), respectively.

Consider a layout structure of a given document as shown in Figure 3.15. The corresponding internal representation, which we called the Labeled Directed Weighted Graph (LDWG) is shown in Figure 3.16. For each block of a given document, there

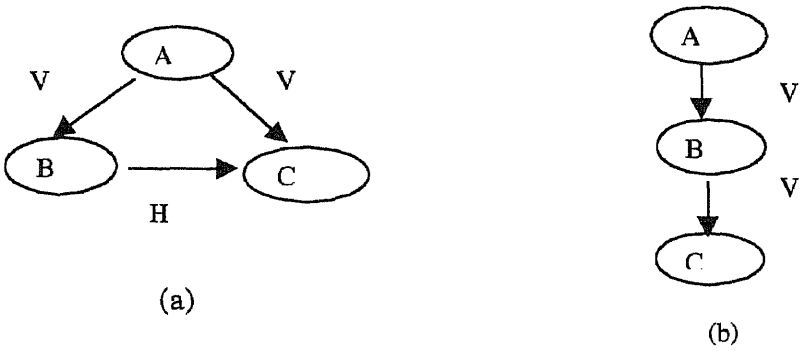


Figure 3.14 Labeled Directed Weighted Graphs
for 3.7a and 3.7b respectively

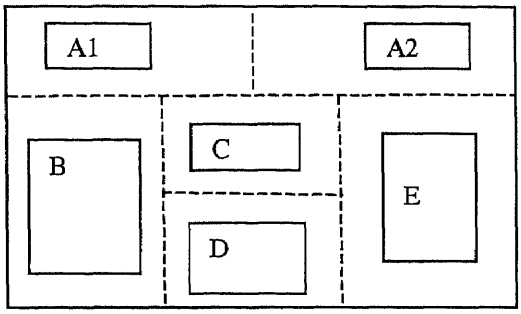


Figure 3.15 Segmentation of a document

corresponds a node in the LDWG. We designate a node as the root of the graph. Usually, the rooted node of the graph corresponds to the most upper left block of the first physical page of the document because the indegree of this node is 0. Two horizontally adjacent blocks of the document are represented by a labeled directed edge labeled by H (stands for horizontal) connected between their corresponding nodes in the graph. Likewise, two vertically adjacent blocks of the document are represented by an edge labeled by V (stands for vertical) connected between their corresponding nodes. The directions are from left to right if they are horizontally adjacent or from top to bottom if they are vertically adjacent. In other words, if a block A is in the left side of and is horizontally adjacent to B, then the labeled directed edge has a label H and is going from the node A

to the node B. If a block A is vertically adjacent to the block B and A is above B in the layout structure, then the labeled directed edge has a label V and is going from the node A to the node B. Associated with each node, there is an assigned value and the value is referred to as the weight of the node.

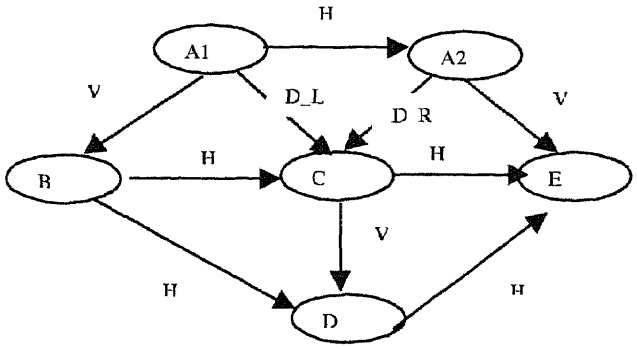


Figure 3.16 LDWG of the document in Figure 3.15

In Figure 3.16, each node of the graph represents a block structure in a given document. Only those blocks which are adjacent have an edge between them. The direction is from left to right for those horizontal blocks, or from top to bottom for those vertical blocks.

In the following, we shall discuss the cutting method, which derive a string representation for the segmentation and then will introduce the algorithm for transforming the blocks into a Labeled Directed Weighted Graph (LDWG) based on this string representation. Assume that each of the blocks has a pair of (X_{min}, Y_{min}) and (X_{max}, Y_{max}) coordinates.

3.1.3.1 Derive String Representation: For all the rectangular blocks, find the minimum Y_{max}, which can divide horizontally the blocks into two groups. This cut is called a

Horizontal cut, denoted as H_c. One group is on the top of the other. Store the cutting relationship of these two groups. If there is only one block in each group, then output this string in the form of H_c(block1, block2); otherwise use a group number to indicate the group and output as H_c(group1, group2). Apply the process to each of the remaining groups until there is only one block in each group; then replace the group number with the new string in the form of H_c(block1, block2). If there is no minimum Y_{max} then in turn, try to find the minimum X_{max}, which divides the blocks vertically into two groups, denoted as V_c. Apply the process to each of the remaining blocks until there is no further possible division of the block. Figure 3.17 depicts the layout structure of a document using the block segmentation technique. In Figure 3.17, without any loss of generality, we use alphabet letters 'A' -- 'Z' to name the terminal blocks (that is, a block can not be further divided into sub-blocks) and we use numerics 1, 2, 3, ... to name the intermediate groups. Each group can have more than one block.

In Figure 3.15, the first cut is the horizontal cut that separates all of the blocks into two sets. Block A_1 and A_2 are in the upper set and block B, C, D and E are in the lower set. Because there are more than one block in each set, each of them get a group number 1 and 2 and output string H_c(1,2) and the blocks in group 1, 2 will be further separated recursively based on the predefined cutting order and cutting criteria. In group 1, after the second cut is made to separate the A_1 and A_2, there is only one block in each set, therefore no group number is needed and the block name A_1 and A_2 is used directly and the previous group number 1 is replaced by use the string $V_c(A_1,A_2)$. Likewise, the group 2 follows the same procedure. Figure 3.17 illustrates the cutting procedure of the example in Figure 3.15.

The pseudo codes of this cutting algorithm is given as followings.

Input: A given document partitions into blocks after segmentation. A linked list of the blocks is given, each of them is in the form of a pair of a starting point (x_{min}, y_{min}) and an ending point (x_{max}, y_{max}).

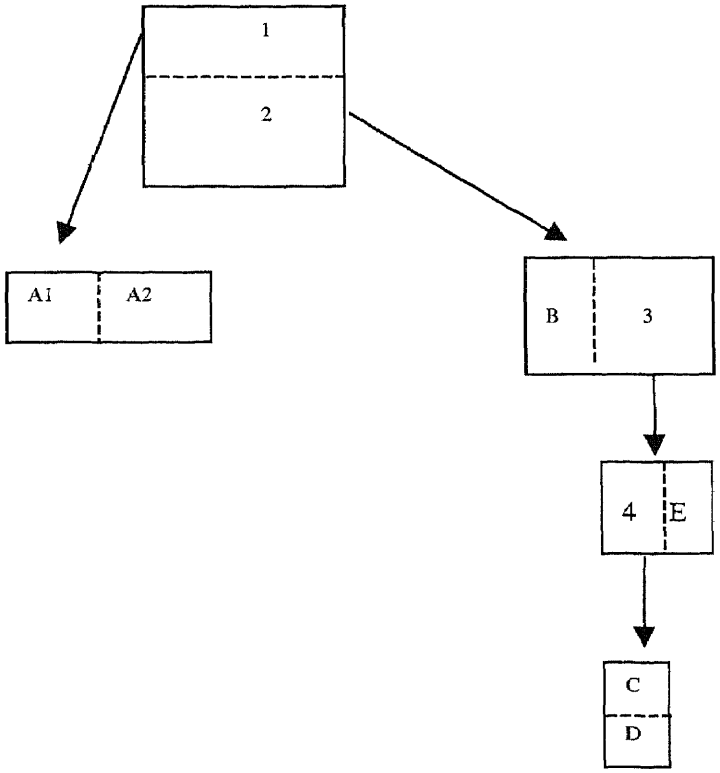


Figure 3.17 Procedure of segmentation of document blocks

Output: Corresponding to the given document, a string representation is obtained.

Function Body:

String_Rep (list of blocks)

If the list is null (no block), then exit;

If this list contains only one block, then output the name of this block and exit;

else /* contains multiple blocks */

```
{
    If (there exists a minimum Y'max to divide the list of blocks
    into two sets Ylist1, Ylist2, one set containing all the blocks, which satisfy
    Ymax ≤ Y'max and the other set containing all the blocks, which satisfy
    Ymin ≥ Y'max)
    then{/*1*/
        divide the block horizontally (i.e. horizontal cut) into two sets-
            Ylist1, Ylist2;
        if (there is one block in the list)
        then use the block name
        else {
            get group number for the set(s);
            replace the input with Hc(num1, num2);
            }
        call String_Rep(Ylist1);
        call String_Rep(Ylist2);
            }/*1*/
    else if (there exists a minimum X'max to divide the list of the blocks
    into two sets Xlist1, Xlist2, one set containing all the blocks, which satisfy
    Xmax ≤ X'max and the other set containing all the blocks, which satisfy
    Xmin ≥ X'max.)
    then {/*2*/
        divide the block vertically (i.e. vertical cut) into two sets-
```

```
                    Xlist1, Xlist2;
            if (there is one block in the list)
            then use the block name
            else {
                    get group number for the set(s);
                    replace the input with Vc(num1, num2);
                    }
            call String_Rep(Xlist1);
            call String_Rep(Xlist2);
                    } /*2*/
    }/ *else */
```

The stepwise derivation of string representation for the example in Figure 3.15 is illustrated in Figure 3.18. The final string representation is in Figure 3.19.

As we observe from the above procedure, the method is unique for partitioning the document into blocks in a certain order, based on a pre-defined order (try to find Y_{max} first, if not found, find X_{max} next), and deriving its string representation. It is important to keep the history of stepwise generating the resultant graph for any given document as shown in Figure 3.18. A document can be partitioned into several blocks, each of the blocks can be, in turn, partitioned into smaller blocks. That is, the nested-segmentation of the document into various blocks, each of these blocks may contain a number of smaller blocks. However, the resultant graph does not reflect the nest-segmentation at all. For example, in Figure 3.16, it does not show that blocks C and D are nested to form block 4;

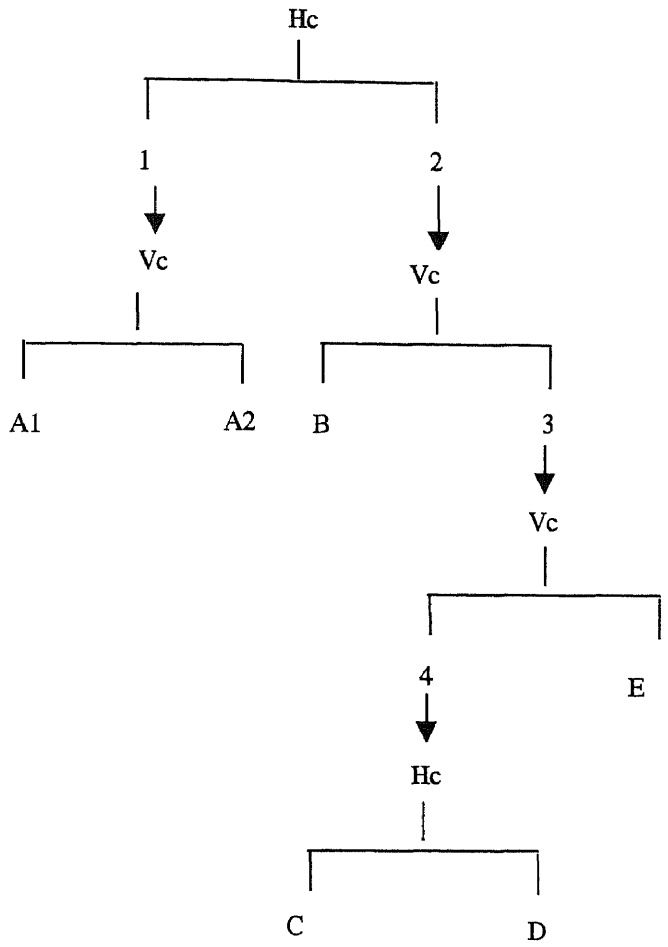


Figure 3.18 Stepwise derivation for string representation

H_c (1, 2)

H_c (V_c(A_1, A_2), 2)

H_c (V_c(A_1, A_2), V_c(B, 3))

H_c (V_c(A_1, A_2), V_c (B, V_c (4, E)))

H_c (V_c(A_1, A_2), V_c (B, V_c (H_c (C, D), E)))

Fig 3.19 String representation for a nested-segmentation

which is, in turn, nested with block E to form block 3.The block 3, which contains the blocks C and D, and E are nested with B to form the block 2. And finally the block 2 is horizontally adjacent to the block 1, which contains block A1 and A2. This nested segmentations can be traced only through the history of stepwise generating the resultant string representation for a given document. However, the final string representation $H_c(V_c(A_1,A_2), V_c(B, V_c(H_c(C,D), E)))$ does represent the nested-segmentation of any document, as shown in Figure 3.19. Therefore, given a document, the layout structure of its image, can be described using the representation of its layout structure in terms of a LDWG. Only a history of stepwise generating the resultant graph for a given document represents the layout structure if it is nested. But the layout structure (even if it is nested) and its string representation are in a one-to-one correspondence. In other words, we could use the string representation to describe a history of stepwise generating a resultant Labeled Directed Weighted Graph for describing a layout structure of a given document. We state the following properties.

Properties:

1. Given a string representation, we can construct an equivalent and unique nest-segmentation of a document;
2. A history of step-wise generating a resultant graph can be represented in terms of a unique string representation;
3. Two resultant graphs are equivalent if and only if they represent the same nested segmentations of a document. (This allows us to draw freely the nodes and edges without taking location of the nodes into consideration.)

3.1.3.2 Transform into a Labeled Directed Weighted Graph: In the previous section, we introduced a methodology of deriving the string representation of a given segmentation. It is much more effective if the technique for matching between two layout structures could use the string representations of block segmentation of documents before going into the graph-graph matching directly. Then the one-to-one correspondence between a layout structure and its string representation is critically important. In this section, an algorithm for transforming the segmentation into a LDWG based on the string representation is presented. Given the following blocks information, as shown in Figure 3.16, the corresponding string representation is $H_c(V_c(A_1,A_2), V_c(B, V_c(H_c(C,D), E)))$, where H_c represents a horizontal cut and V_c represents a vertical cut. A horizontal cut (denoted as H_c) is to divide a given block into two sets such that they are vertically adjacent. Likewise, a vertical cut (denoted as V_c) is defined.

Although the string representation can not reflect the exact position of the original blocks, it can be used for matching two images to exclude the different type of blocks.

Next, we use the string representation and the original image to derive the LDWG. The algorithm is given as followings.

A pointer to the characters, one at a time, of the input string representation, scanning from the left most character to the right most character. If a block name is encountered, then pass it onto the stack. If a V_c or H_c operator is encountered, then push it into the stack. Check the top three elements in the stack, see whether they are in order that an operator is in the bottom of two operands (The operands can be a single block or a group); that is: they become one unit which is ready to be drawn. Pop the three elements (that is, the operands and the operator) from the stack. Then use an intermediate number

to replace that unit and push it into the stack. Continue the process until all the characters in the string are processed and all the vertices are drawn. A stack is used to allow the algorithm to draw the blocks in the opposite sequence in which they are cut.

The algorithm is in three parts: TranLDWG(), draw_graph() and those functions that return the blocks according to their relative position. Procedure TranLDWG is the main body. It is responsible for checking whether the drawing condition is satisfied, then call the draw_graph. In turn, draw_graph calls those functions, within which the blocks' relative positions are computed to get the vertices and edges and then to draw them. The algorithm is written in pseudo codes which are given as follows.

Procedure TranLDWG(string)

```
{
/*The stack is used for processing the string representation, the link-list records the
intermediate result during this processing. */
Initialize the empty stack and the empty linked list;
Scan the string from left to right;
While (not reaching the end of string)
{       if the character is '(' or ')' or ',' , ignore;
        if the character is an operator (Vc or Hc)
                then push it into stack;
        if the character is an operand (either a block name or a group name)
                then {  push it into stack;
                        /* check the top 3 elements in the stack */
                        if they are in sequence of an operator followed by two operands
```

```
                then {
                        pop up these 3 elements;
                        draw_graph (operator( operand, operand));
                        get a group number and write into the linked list;
                        push this group number into stack;}
}
} //end of TranLDWG
```

Function draw_graph(P(X,Y))

```
/* P stands for the operator;
X and Y stand for the two operands;
The operand may be either a single block or a group which contains several blocks */
{
Vector V1, V2;  /* store the blocks' name */
        if P is Hc {
                        V1= BOTTOM_BLOCKS( X );
                        V2= UPPER_BLOCKS(Y);
                        if both V1 and V2 are single blocks
                                then draw node V1 and node V2 and an edge from V1 to V2
                                and labeled as V;
                        else /* one of X, Y is group or both X and Y are groups */
                                for each blocks in V1 compare with those blocks in V2
```

```
                    one by one using the coordinates of blocks to determine a
                    directed edge with label H, D_L or D_R;
                        draw vertices and edges between them;
            }
        if P is Vc {
                V1= RIGHT_MOST_BLOCKS(X);
                V2= LEFT_MOST_BLOCKS(Y);
                if both V1 and V2 are single blocks
                    then draw node V1 and node V2 and an edge from V1 to V2
                    and labeled as H;
                else /* one of X, Y is group or both X and Y are groups */
                    for each blocks in V1 compare with those blocks in V2
                    one by one using the coordinates of blocks to determine a
                    directed edge with label V, D_L or D_R;
                        draw vertices and edges between them;
                }
}//end of draw_graph
```

Function Vector LEFT_MOST_BLOCKS(Vector X)

```
{//return those blocks which are the most left among all the blocks in X;
 //The concept of left_most blocks is explained in the next paragraph.
    if X is a single block, then return X;
    if X is in the form of Vc(A, B)
```

```
            return LFET_MOST_BLOCKS(A);
        if X is in the form of Hc(A, B)
            return  (LEFT_MOST_BLOCKS(A) + LEFT_MOST_BLOCKS(B));
}//end of LEFT_MOST_BLOCKS
```

Function Vector RIGHT_MOST_BLOCKS(Vector X)

```
{//return those blocks which are the most right among all the blocks in X;
        if X is a single block, then return X;
        if X is in the form of Vc(A, B)
            return RIGHT_MOST_BLOCKS(A);
        if X is in the form of Hc(A, B)
            return  (RIGHT_MOST_BLOCKS(A) + RIGHT_MOST_BLOCKS(B));
}//end of RIGHT_MOST_BLOCKS
```

Function Vector UPPER_BLOCKS(Vector X)

```
{
        if X is a single block, then return X;
        if X is in the form of Vc(A, B)
            return  (UPPER_BLOCKS(A) + UPPER_BLOCKS(B));
        if X is in the form of Hc(A, B)
            return  (UPPER_BLOCKS(A));
}//end of UPPER_BLOCKS
```

Function Vector BOTTOM_BLOCKS(Vector X)

{

if X is a single block, then return X;

if X is in the form of V_c(A, B)

return (BOTTOM_BLOCKS(A) + BOTTOM_BLOCKS(B));

if X is in the form of H_c(A, B)

return (BOTTOM_BLOCKS(A));

}//**end of BOTTOM_BLOCKS**

/* The end of the algorithm */

We will use the example in Figure 3.15 to illustrate how to use the algorithm above to transform a string representation into the LDWG in Figure 3.16.

In the very beginning, the string is $H_c(V_c(A_1,A_2), V_c (B, V_c (H_c (C,D), E)))$ and the stack and the linked list are empty. During the procedure of scanning the string from left to right, the stack will reach a state of $H_cV_cA_1A_2$. At this time, the drawing condition is satisfied and draw_graph ($V_c(A_1,A_2)$) is called. In draw_graph($V_c(A_1,A_2)$), the operator is V_c, V_1 returns the RIGHT_MOST_BLOCKS(A_1), which is A_1 itself and V_2 returns the LEFT_MOST_ BLOCKS(A_2), which is A_2 itself. Therefore the vertex A_1 and vertex A_2 are drawn and the edge from A_1 to A_2 labeled as H is drawn too because both A_1 and A_2 are single blocks. Then the draw_graph returns to the TranLDWG; pops up these three elements, gets a group number 1 and puts it back into the stack.

After this first step, the string becomes $H_c(1, V_c(B, V_c(H_c(C,D), E)))$ with the input pointer indicating the first V_c and the contents in the stack is H_c1. Next step, the operators and the operands are put into the stack sequentially until the content of the stack is $H_c1V_cBV_cH_cCD$ and the drawing condition is satisfied again. At this time, the vertex C and D are drawn and an edge from C to D labeled V is drawn too.

A new group number 2 replaces the substring $H_c(C, D)$. The string becomes $H_c(1, V_c(B, V_c(2, E)))$ and the content in the stack is $H_c1V_cBV_c2$. Next, operand E is put into the stack, then V_c2E satisfies the drawing condition again. The function draw_graph $(V_c(2, E))$ is called. The operator is V_c, then V1 returns RIGHT_MOST_ BLOCK(2), which are the C and D and V2 returns LEFT_MOST_ BLOCK(E), which is E. Then the two vertices in the group 2 will be compared with E one by one. Therefore the vertex E is drawn and two edges between C, E and D, E are drawn separately. The stepwise transformation is given in Figure 3.20.

String: $H_c(V_c(A_1,A_2), V_c(B, V_c(H_c(C,D), E)))$
Stack:
Graph:

Figure 3.20 (a) Initial Status

String: $H_c(1, V_c(B, V_c(H_c(C,D), E)))$
Stack: H_c1
Graph:

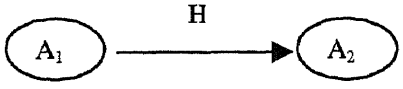


Figure 3.20 (b) After $V_c(A_1,A_2)$ is processed

String: $H_c(1, V_c(B, V_c(2, E)))$

Stack: $H_c 1 V_c B V_c 2$

Graph:

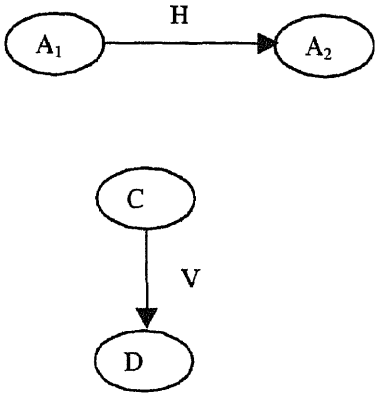


Figure 3.20 (c) After $H_c(C, D)$ is processed

String: $H_c(1, V_c(B, 3))$

Stack: $H_c 1 V_c B\ 3$

Graph:

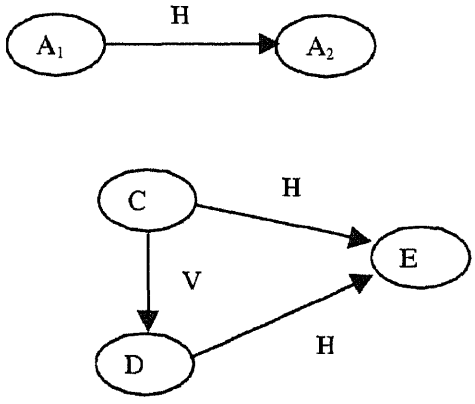


Figure 3.20 (d) After $V_c(2, E)$ is processed

String: $H_c(1, 4)$

Stack: H_c14

Graph:

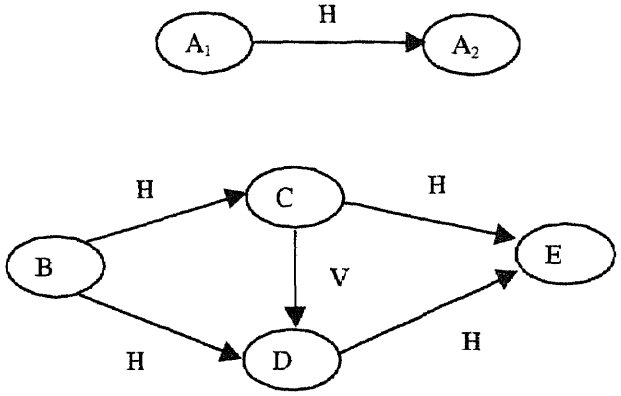


Figure 3.20 (e) After $V_c(B, 3)$ is processed

String: 5

Stack: 5

Graph:

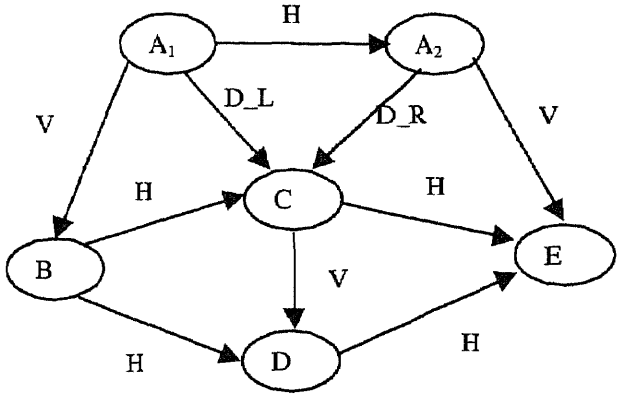


Figure 3.20 (f) After $H_c(1, 4)$ is processed

Figure 3.20 Stepwise transformation for example in Figure 3.15

In step f, the string becomes 5 and the pointer points to the end of the string and the content in stack is the group 5. The algorithm reaches the ending condition and the program finishes successfully.

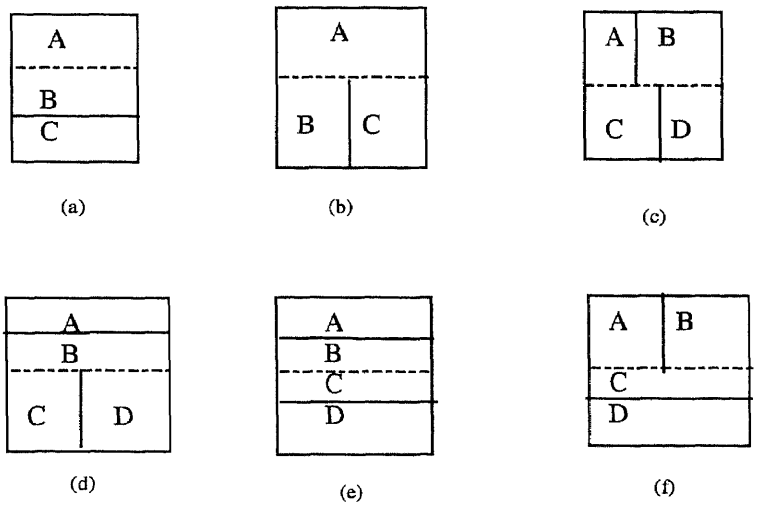


Figure 3.21 Possibly adjacent positions of blocks and groups

Now a brief explanation is necessary for the four functions to compute the relative positions of the blocks. The main principle is that when the two groups (or two blocks) are cut by a horizontal cut, only those BOTTOM_BLOCKS in the above group can possess V-adjacent with those UPPER_BLOCKS in the below group. As shown in Figure 3.21 (a) -(f), the dot-line is the horizontal cut and the line in the rectangle indicates that there are more than block in this group. In (a), block A will be returned from the BOTTOM_BLOCKS(A) and B will be returned from the UPPER_ BLOCKS (group 1). In (c) block A and B will be returned from the BOTTOM_ BLOCKS(group 1) and blocks C and D will be returned from the UPPER_ BLOCKS (group 2) because all of them possess possibly adjacent with each other.

From the above analysis, we can conclude that the string representation is helpful in representing the global structure of the original image, but the detailed relative positional information should be also kept in the LDWG.

Lemma 1: Given a document, the process of stepwise generation of its layout structure can be represented by a history of Labeled Directed Weighted Graph.

Lemma 2: Given a document, its layout structure with the process of stepwise generating it, and the string representation are in one-to-one correspondence.

Theorem 1: The string representation of a Labeled Directed Weighted Graph for a layout structure of any given document is a map for navigating the entire graph by visiting every node once and passing through each of the labeled directed edges once.

Lemma 2 states a quite important and meaningful fact. To identify whether two documents have the same layout structure, we need to determine whether their internal representations (LDWGs) are equivalent. Since the graphs can be partially represented by the string representations, matching two graphs is done after a comparison of their string representations first. Furthermore, a document class may have several different layout structures and therefore it can be represented by a group of graphs. Various document classes may have some common properties, and therefore a document type hierarchy (DTH) is constructed [5-8].

3.1.4 Computing the Weights of Blocks

Given a document, users can identify its document type by taking a glance at its features. The document is classified into its document type based on the users observation on certain significant and referential physical features of the document. For example, a document of memo type is characterized by the blocks containing the information of the receiver, sender, date, and the subject's nature. A document of letter type is characterized by the blocks containing name and address of the receiver, "Dear Sir", "Sincerely Yours" and others. A document of journal paper type is characterized by various blocks containing the name of the journal, title, authors, their affiliations and others. Based on these observations, it is reasonable to differentiate the degree of significance of a block from others for classifying the document as its type. Hence, we assign a weight to each block of a given document to express the degree of significance of each block.

Then the next question is how to compute the weight of each block for a LDWG. We provide two ways. First, users may set up the weight of each block. Second, the system can obtain the final proper set of weights after "learning" using the Perceptron Learning Algorithm (PLA).

In the machine learning [77] area, parameter adjustment (PA) is one of the simplest forms of learning and one of the earliest and still one of the best known PA systems is Perceptron [78,79]. It is essentially a hill-climbing, gradient-decent search algorithm.

The Perceptron learning method follows the current-best-hypothesis (CBH) scheme. In this case, the CBH is defined by the current values of the weights. The initial value are assigned randomly, usually from the range [-0.5, 0.5]. Then the weights are

updated to try to make them consistent with the examples. This is done by making small adjustments in the weights to reduce the difference between the actual values and predicated values.

The model is abstracted in Figure 3.22.

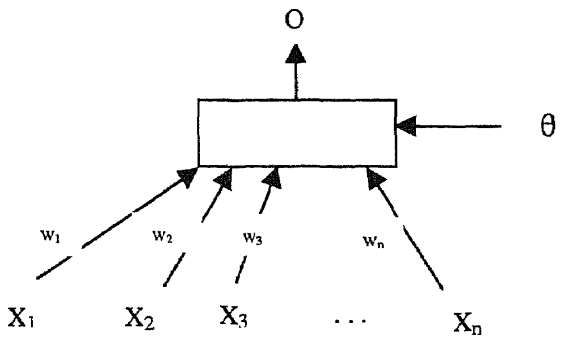


Figure 3.22 Original Perceptron model

With input parameters as $X_1, X_2, \ldots, X_n$ and predefined threshold θ, the output is

$$O = \begin{cases} 1 & if \quad \sum_{i=1}^{n} w_i x_i > \theta \\ 0 & Otherwise \end{cases}$$

The only thing that varies is the weights. At the beginning, the weight $w_i(t+1)$ for X_i starts with a random weight. In the learning procedure, those random weights will be adjusted from training examples and will "get better" towards the final set of weights. The Perceptron uses an error-correction learning algorithm in which the weights after an erroneous response are adjusted, as follows:

$$w_i(t+1) \leftarrow w_i(t) - \eta \bullet \Delta w_i(t)$$

η is the learning rate, $\eta \in (0,1]$.

and $\Delta w_i(t) \leftarrow (t_x - a_x) \bullet X_i$, in which t_x represents the training examples and a_x stands for the actual activities. No change is made after a correct response; but after a mistake all the feature values are either added or subtracted from the weights, depending on whether the system's output is too high or too low on the previous trial.

Next, we apply the PLA in the document samples in order to get the proper set of weights for each document type existing in the system. Given a document, the classification finally (after the string matching and layout matching) determines to which document type an incoming document belongs by finding a *unique* sample document with a set of real-valued weights $w_{1i}, \ldots, w_{ki}$ under document type i such that

$$\sum_{j=1}^{k} w_{ji} * M(X_j, X_j') >= Threshold$$

where

X_j is the j^{th} attribute of the sample document;

X_j' is the j^{th} attribute of the incoming document;

k is the number of attributes of the sample document;

$i \in$ {Document type defined in current document sample base};

$M(X_j, X_j')$ is the matching result between attribute X_j in the sample document and X_j' in the incoming document.

$$M(X_j, X_j') = \begin{cases} 1 & \text{if } X_j = X_j' \\ 0 & \text{if } X_j' \text{ missing} \\ -1 & \text{if } X_j \neq X_j' \end{cases}$$

A document type for a document is determined by taking a linearly weighted summation, which functions as a *linear threshold unit* (LTU). Our classifier is a linear

classifier. The Perceptron Learning Algorithm (PLA) [78] is an algorithm for learning such a set of weights for an LTU. For training purposes, the examples, which are used to train the system, are separated into positive examples and negative (or counter) examples. In general, for each document type, there exists a logical structure (with a definition in the form of attribute names) in the sample base of the system. Given a document type, although all the sample documents stored under this type share the same logical structure, some attribute names may not occur in some samples. In this case, we assign the null attribute name. These examples will be used to determine the significance of blocks in the logical structures of the given document type. That is, the weights for each block are generated using PLA. For training purposes, *all* sample documents are divided into positive examples and negative examples. For each document type i, all the samples of the same type i are considered as positive examples, and the rest of the samples which are not of the type are considered as negative examples.

In the training stage, using the initial set of weights, some positive samples, which should be classified as this document type, may be excluded; and some negative examples, which should be excluded, may be classified incorrectly as this document type. Both of these cases are called *misclassification*. Those misclassified documents are used to adjust the current weights in order to get the final correct weights. It starts with a initial set of weights and iteratively refines the weights to minimize the number of misclassified examples. The prerequisite for using the PLA is that the data are "linearly separable" between the two pattern classes.

The goal of using the learning algorithm is to find a set of weights for all the blocks in each document type that satisfies the following:

$$\sum_{j=1}^{k} w_{ij} * F(X_j, X_j') >= Threshold$$

for all the positive examples of this document type i, and

$$\sum_{j=1}^{k'} w_{jp} * F(X_j, X_j') < Threshold$$

for all the negative examples of document type p except type i in the current system document base.

There are many variations on the Perceptron theme. For our needs, we augment and modify the original PLA in two places. First, instead of randomly assigning the initial values of the weights, we use statistic methodology to assign the initial weights. Second, we extend the number of pattern classes beyond two. This is a good way to find a set of weights, in comparison with manual engineering of weights.

Instead of storing one set of weights, we use a two dimensional matrix to record each set of weights for every document type T_i as below in Figure 3.23.

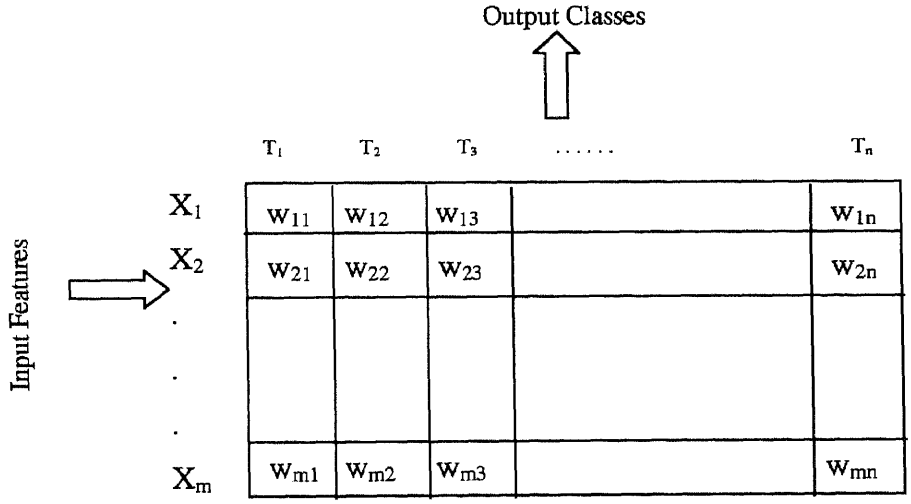


Figure 3.23 Weights matrix of document types

In the above matrix, the row is the universal attribute for all the document types defined in the document sample base. The column represents each document type.

Although there may be several samples under each type, we assign a set of weights for each type instead of each sample. In this augmented Perceptron, the column number is the number of the document type in this system. When an incoming document arrives, all of its attributes X_j' are compared with the attribute X_j of the sample documents one by one. The sum for each column is added up and the largest total picked as the systems output. If this output is larger than the predefined up_threshold ("goodness") and the second largest value is less than the low_threshold ("uniqueness"), then this classification is successful and the document type is output. Otherwise the error-correction algorithm subtracts from the feature values from the corresponding C values for the column that gave the wrong answer, and adds the feature values to the column that failed to give the right answer. When the system makes a mistake, all the feature values are subtracted from the weights in the column that gives the incorrect response, and the feature values are added to the column that should have given the response (but failed to do so). Other columns are left unaltered. Weights that are too large are thereby reduced and those that are too small are increased.

The Augmented Perceptron Learning Algorithm is as follows:

Input: A document type i, a set of positive examples (sample documents of the same document type i) and a set of negative examples (training documents other than the given document type i).

Output: A set of weights, each weight is associated with a block for a document type.

1) Pick a document type i, then all the sample documents are positive examples of this type i and all the others are used as negative examples.

2) Initialize the weights in column i based on statistical computation on all the positive examples as follows:

2.1) Given type T_i, find out the occurrence of attribute X_j in all the samples S_1, ...S_z

$$F(X_{ji}) = \sum_{p=1}^{z} N_{pi}^{j}$$

where X_{ji} is the attribute X_j of document type T_i. For each attribute $p \in [1..z]$ where z is the number of positive examples for document type i and $N^j_{pi}=1$ iff the attribute p occurs in the sample document.

2.2) The initial value of W_{ji}, $j \in [1..m]$, is decided based on the frequency of this attribute in this document type.

$$W_{ji} = \frac{\sum_{p=1}^{z} N_{pi}^{j}}{\sum_{q=1}^{m} \sum_{p=1}^{z} N_{pi}^{j}}$$

where m is the number of all the attributes in document type i.

3) For **all** the positive and negatives examples test to see whether the goal is achieved. if yes, exit; else do the following:

3.1) if classification is correct, ignore, go next;

3.2) Adjust the weight if a misclassification occurs as follows for the two cases:

a) if the input sample is a positive example that is misclassified as negative,

if this input sample misses some attributes, then increase the weight of all the other attributes except the missing attribute in the current matrix by (threshold - the weight of the missing attribute);

if this input sample has more attributes than those in the matrix, then increase the different matched attributes' weights;

b) if the input sample is a negative example that is misclassified as positive,

decrease the weights of all the matched attributes in the input sample.

Repeat the process until the final condition is met.

//end of the augmented PLA

Sometimes, in order to accelerate the tuning of the weights, the system allows users to adjust the weight of a block manually.

3.2 Building the Similarity Table

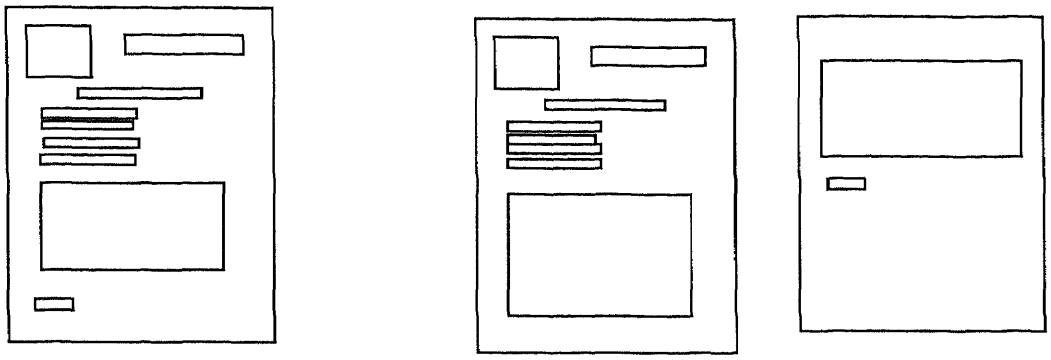
The similarity table is a fast-reference table that helps to decrease unnecessary matching or comparisons. It can be used to record the "similarity" of each pair of layout structures in the document sample base. Suppose that, the values "0.9-1" indicates very similar, and "0.1-0.2" indicates not similar at all. Then, the similarity table is a n*n table, in which S[i, j] is the degree of similarity between layout structures i and j.

The matrix is symmetric, so S[i,j]=S[j,i] and S[i,i]=1. Therefore, only half of the matrix is needed to store.

3.3 Multi-Page Normalization

To deal with the multi-page document, we propose the virtual page and multi-page normalization method to normalize the fraction size of blocks.

Considering the following two memos. One is in the sample base, and the other is a two-page document to be classified.



(a) One page memo (b) Two-page memo

Figure 3.24 Sample memo and one multi-page document to be classified

The two documents are the same under the view of a virtual page.

After the document is processed through the OCR, it is segmented. We can tell what the types of the blocks are.

With the IMAGE_BLOCK and GRAPHIC_BLOCK (whose positions are stationary in the documents), compute the size percentage based on one page.

For the ATTRIBUTE_BLOCK (which are text based, but with structure), take the size percentage based on the page.

But the FREE_TEXT_BLOCK, the percentage is left open. Its size percentage will be assigned as what remains after removing the other types. That is 100 - the sum of the percentages of all other types.

Size_%(FREE_TEXT_BLOCK)=

100-[Size_%(GRAPHIC_BLOCK)+Size_%(IMAGE_BLOCK)+Size_%(ATTRIBUTE_BLOCK)];

CHAPTER 4

DOCUMENT LOGICAL REPRESENTATIONS

4.1 Introduction of Frame Template

Data Modeling for document management systems has gained quite a bit of attention. Horak[81], Croft and Stemple [82] represented the structures of documents based on the Office Document Architecture (ODA). ODA is part of the standards for document interchange developed by the International Standardization Organization (ISO) and the European Computer Manufacturers Association (ECMA). It distinguishes between the *logical* and *layout structures* of a document. The logical and layout structures are made up of hierarchies of *logical objects* and *layout objects*, respectively. The logical and layout objects are classified according to their *type*, which is the document *class*. The logical structure associates the content of the document with a hierarchy of logical

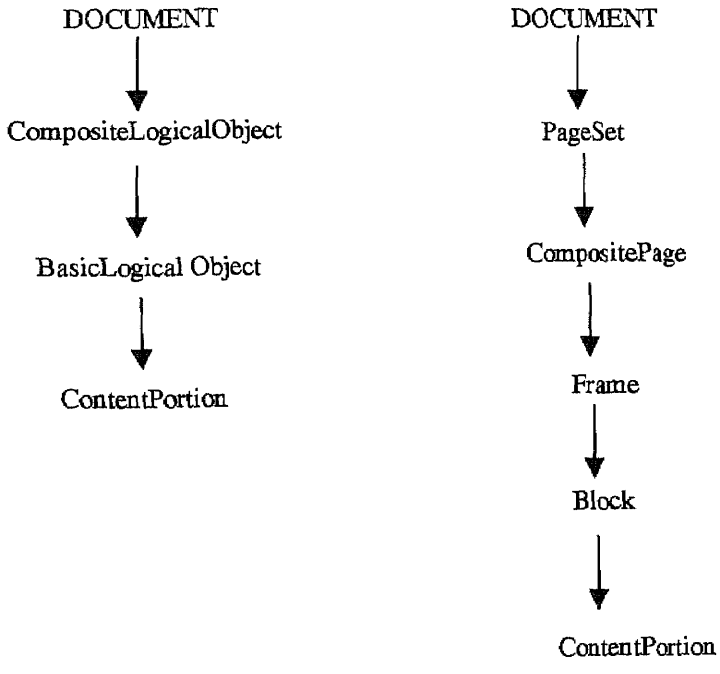


Figure 4.1 Simplified ODA document structure

objects. Examples of logical objects are summaries, titles, sections, paragraphs, figures and tables. The layout structure associates the same content with a hierarchy of layout objects. Examples of layout objects are pages, columns, and footnote areas. ODA requires that each document has a logical structure and a layout structure, together with a set of logical-layout, logical-logical and layout-layout relationships. A simplified ODA document structure and a type hierarchy of ODA objects are depicted in Figure 4.1 and Figure 4.2, respectively (excerpts from [82]). There is a distinction between *composite* and *basic* logical object types. Composite logical objects comprise other composite logical objects or basic logical objects. Basic logical objects are associated with *content portions*, which contain the contents of a document. Included in the layout object types are *page sets, composite pages, basic page, frame,* and *block.*

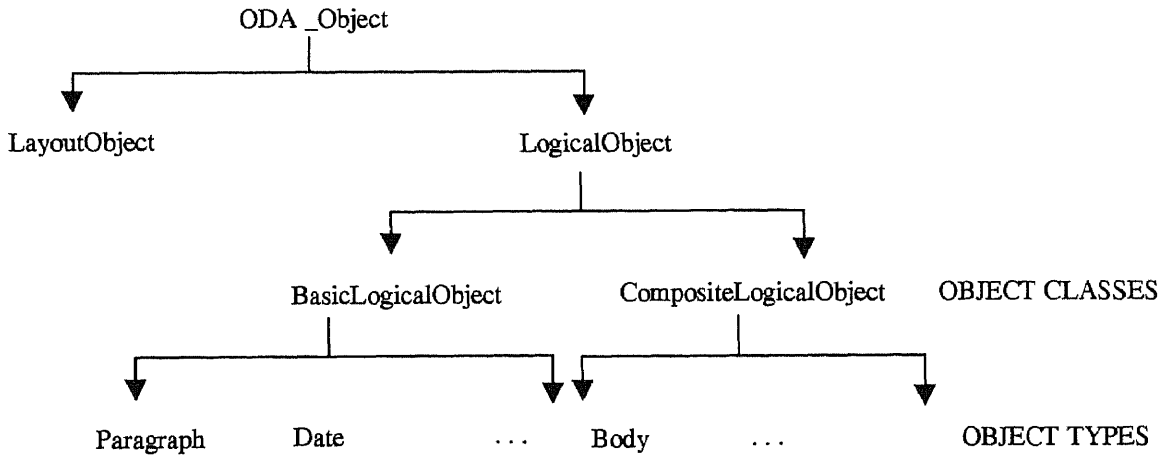


Figure 4.2 Type hierarchy of ODA objects

Bertino, Rabitti and Gibbs [83] extended the ODA standard by including a conceptual structure, which allows a system to specify a document in terms of its conceptual component types. A conceptual component type is defined by a set of attributes. It

represents a portion of a document used for some specific purpose (e.g., the sender of a memo). Figure 4.3 shows an example of conceptual structures of document types Generic_Letter and Business_Product_Letter (Figure 4.3 is an excerpt from [83]). In the figure, the attributes inside the box represent the Generic_Letter document type and those outside the box are included to specify the representation of the Business_Product_Letter document type. The authors argued that component types are more meaningful to the user than the logical and layout components in terms of retrieval where <attribute, value> pairs can be used in specifying queries. This enables the model to support a well-defined query language and techniques for query processing. Bertino et al. Described a distributed office system called MULTOS (MULTimedia Office Server) based on this ODA extension. (MULTOS is also described in [84].) Utilization of conceptual component types allows for the exploitation of the aggregation relationship abstraction [85]. For example, in Figure 4.3, the component type **sender** can be considered as an aggregation of conceptual component types **Name** and **Address**. A distinction of a concept of type [83] is made between a *strong component type* and a *weak component type*. A strong component type completely specifies the structure of its instances (e.g., in the relational model [86, 87, 88], a relation schema completely defines the structure of its instances (or tuples)). Thus, the component types are not divisible any further. MULTOS introduces the concept of a weak type to the conceptual data model. A weak type only partially specifies the structure of its instances; i.e., the instances can have more complicated attributes. We are thus able to define document types at different levels of detail.

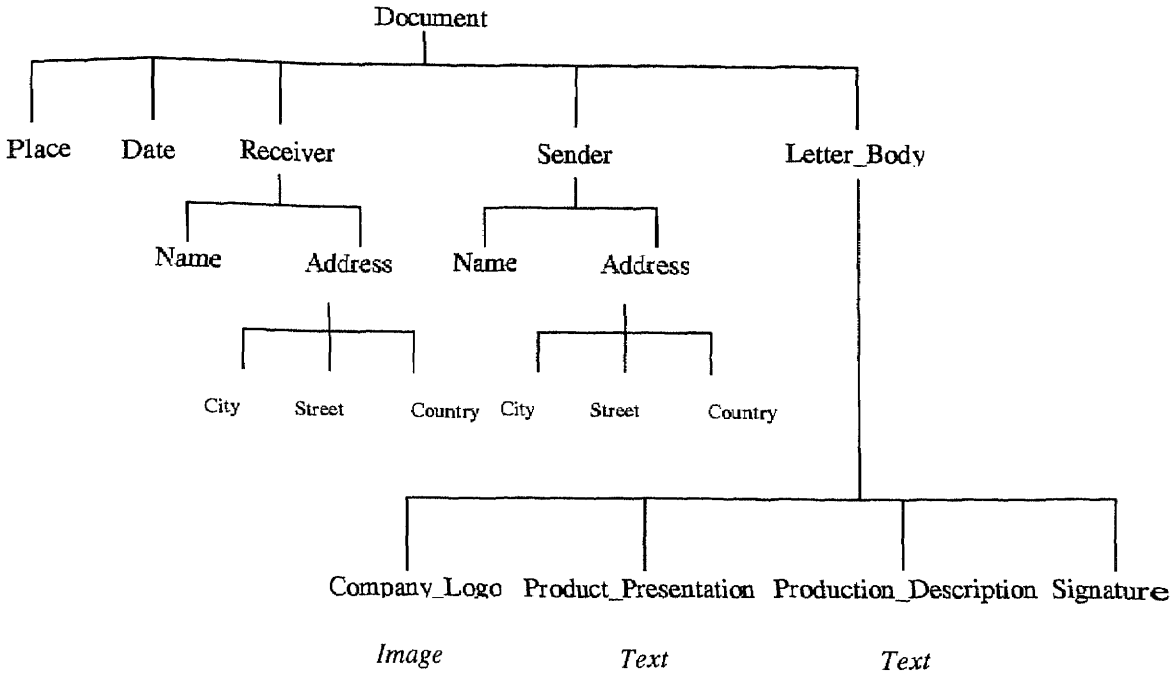


Figure 4.3 Example of conceptual structures of document types Generic_Letter and Business_Product_Letter

The document types shown in Figure 4.3 are defined at different levels (see the attribute **Sender** for example). This allows the use of *path notation* [89] in referencing a conceptual component type in a document. For example, to reference only the **City** component type of a sender (cf. Figure 4.3), the path name would be of the form **Sender.Address.City**.

Lutz et al. [33] developed a document classification system, called MAFIA (Mail-Filter-Agent), based on MULTOS. The MAFIA provides an automatic document classification system which utilizes the conceptual data model. The basic modeling principles discussed are those of *aggregation* [85], typing, and generalization [85]. The representation of documents is described through the aggregation of conceptual component types. Documents are defined at different levels of detail using the concept of

typing called the weak type. (Figure 4.3 illustrates the concept of the weak type. Note that the two component types Generic_Letter and Business_Product_Letter are defined at different levels of detail). MAFIA, however, is a system only used for electronic mail.

Hoeper [90] extended ODA to support multimedia documents by integrating synchronization properties and temporal relationships into it. The presentation of multimedia documents is considered to be a set of actions temporarily related to each other, which are executed in a special intended sequence defined by the user. The scheduling is called synchronization of actions.

Woelk, Kim and Luther [91] presented an object-oriented approach to describing multimedia documents. The basic object-oriented aspects that are required in the standard object-oriented paradigm are the notions of instantiation and generalization. Woelk et al. extended these two notions by augmenting the notions of aggregation and relationships to capture the data modeling requirements of multimedia applications. Information in a document is considered, in the first place, to form an aggregation (*part-of*) hierarchy of component node types. A component node, in addition to its place in the aggregation hierarchy, is also considered to be a part of a generalization hierarchy. A generalization hierarchy, in terms of subtyping, defines a component node N as a subtype of a component node M such that M can reuse the attributes defined for N; M becomes a specialization of N. In addition, each of M and N can be an aggregation of component node types. The component node types can result into a *dag* structure since any node can have a relationship with any other node in the aggregation hierarchy and generalization hierarchy. The paper elaborates on augmenting the basic data modeling requirements by utilizing the concept of a *token object* which provides a single mechanism for

representing diverse types of data and relationships among these diverse types of data. However, augmenting the notions of instantiation, generalization and aggregation into one concept of a token object increases the complexity of property inheritance and constraints management [91]. Property inheritance and constraints management are more complex in this system than in conventional object-oriented systems since the data model discussed here supports the notions of instantiation, generalization and aggregation.

Christodoulakis et al. [92] represented multimedia documents using two structures: a logical structure representing the logical components of the documents such as titles, sections, paragraphs and so forth, and a physical structure specifying the components of the layout presentation of the documents on an output device such as the screen of a workstation. A mapping from the logical to the physical structure of a document is provided to specify which components of the logical structure are mapped onto which components of the physical structure. The argument given for separating the logical structure from the physical structure is that the same logical structure shared by two different documents can be presented through different mappings. The authors implemented this technique of describing multimedia documents into the MINOS multimedia information system.

Our work differs from the above approaches in several ways. First, we do not model a document using logical, physical, layout or conceptual structures. Instead, we combine these structures and incorporate them into a frame template. The idea of combining the logical and layout structures into a frame template allows the user to store the synopsis, as opposed to the original document, into the template. (In other words, we do not distinguish between logical, physical, layout or conceptual structures of a

document. Rather, we concentrate on the information that the user considers to be significant from the document.) We call the synopsis of a document a frame instance. Each frame instance is composed of a set of attributed-value pairs. (The frame instance results from instantiating the document's frame template.) The information contained in the frame instance represents the most significant information (i.e., the synopsis) of the document pertinent to the user. Various frame instances can be grouped into a folder based on the nature of their contents. One motivation for considering a frame instance rather than the original document is that the frame instance describes the document in a succinct manner. Also, a user may not be concerned with all the information contained in a document. When retrieval occurs, the information contained in frame instances suffices to satisfy the user's needs.

Our document model is a dual one – it provides a separate treatment of the structural organization of documents from the real-world folder organization perceived by the user. The structural organization of documents is depicted by a document type hierarchy, which is used for classifying various documents based on the generalization abstraction among the frame templates. The folder organization, on the other hand, mimics the user's document filing system.

4.2 Frame Template, Document Type Hierarchy and Operations on Documents

In document processing, retrieval by content is always the dream of the users. But because of the huge number of documents, the time in comparing the content with the query is not feasible. In addition, users also need to find particular sets of documents

under some conditions, vague query and document synthesizing. Most of the operations for document processing are impossible without the definition of document structure.

On the other hand, some of those requirements can be achieved by a series of database operations with the price that users have to define the detailed data definition and be familiar with the database operations, which is still not applicable for most computer users who may not be database specialists. In addition, the traditional relational database technologies have the following points which are not applicable in document processing:

a) No information about hierarchical relation among documents.

b) Less flexibility in defining data type patterns.

c) No composite attribute definition.

d) No multi-valued and repetitive definitions.

e) No vague inquiry.

f) No thesaurus.

In another word, document processing is neither properly supported by free text processing, nor by traditional relational database operations. We propose the weak-type concept, which is different from free text and the strong type provided by database.

In order to represent the logical structure of a document, we propose to use frame templates. The frame template of a document type is composed of a series of attributes, which may be simple type or composite type. Simple type is not divisible and composite type can be divisible further into simple type or composite type again. Only the simple types are associated with data types.

1) Hierarchical relation

We introduce the concept of DTH to characterize the hierarchical relation over different document types. It is a parent-children relation between two document types. It is an one-to-many relation, which means that a parent may have several children. Each child has at most one parent. Therefore, a child document type will inherit all the attributes from its parent.

Let A and B be the document types. For example, A is the memo type, and B is the meeting memo type. If type B inherits all the attributes defined in type A, then we say that type A is the parent of type B; and type B is a child of type A. Users may create a new frame template from a parent-template. In this way, users may add new attributes besides those inherited from parent-template, but they can not delete or update an inherited attribute because those inherited attributes may also be inherited by other children of the same parent.

A frame template is always able to be modified or deleted if and only if

1) There is no frame instance related with this frame template;

2) There is no child frame template inherited from this frame template.

If there are some frame instances which have been filed and stored into the frame instance base, then the modification or deletion operations on the frame template will cause some attributes in the frame template to have no values or wrong values assigned. This inconsistency should be avoided. In such a situation, if the user still wants to modify or deletes the definition, then all the related frame instances must be removed from the frame instance base.

If there are some children frame templates related to this parent template and some attributes of the parent template are to be modified, then first check whether there are any frame instances related to any of these frame template level by level. If there are any, then refer to the above. Otherwise, any changes made in parent frame template will bring about the corresponding changes in the children frame templates.

2) Multi-value and repeating group

To meet the expression of the complicated document logical structure and allow the maximum flexibility, each attribute in the frame template is associated with a boolean type whetherRepeatable to indicate whether the attribute is repeatable. If the attribute is simple type, then it means multi-value is allowed for this attribute; if this attribute is composite type, then is means this is a repeating group.

For example:

Frame template: ProceedingsArticles
TitleOfArticle
AuthorsOfArticle {NameOfAuthor LastName
FirstName
InitialName
AffiliationOfArticle NameOfOrganization
AddressOfOrgnization RoomNos
BuildingName
Street
City
State
Zip
Country
{OfficePhoneNos}
EmailAddress}
PageNos PageFrom
PageTo
ContentDescriptionOfArticle

Figure 4.4 Frame template definition with multi-value and repeating group

In an article of a proceeding, there is only one title and there may be one or several authors with one article. In the above example, AuthorsOfArticle is a repeating group and OfficePhoneNos is an attribute which allows multi-value.

We use the following structure of boolean to express the above structure. (T) represents True and (F) represents False.

TitleOfArticle(F)
AuthorsOfArticle(T) NameOfAuthor(F) LastName(F)
FirstName(F)
InitialName(F)
AffiliationOfArticle(F) NameOfOrganization(F)
AddressOfOrgnization(F) RoomNos(F)
BuildingName(F)
Street(F)
City(F)
State(F)
Zip(F)
Country(F)
OfficePhoneNos(T)
EmailAddress(T)
PageNos(F) PageFrom(F)
PageTo(F)
ContentDescriptionOfArticle(F)

Figure 4.5 Internal structure to record multi-value and repeating group

CHAPTER 5

CLASSIFYING DOCUMENTS

In this chapter, we shall discuss the automatic document classification and information extraction. We shall investigate how to classify an incoming document as a particular document type based on the predefined document types which are stored in the document sample base and how to choose a "most-similar" sample as the template of further extraction. Then, this incoming document is assigned to a document type, the logical structure represented by a frame template and a collection of associations between each component of the layout structure and the logical structure. In the extraction phase, the information extraction can be accomplished simply by traversing the Labeled Directed Weighted Graph of the incoming document, taking into account the internal connections for the associations between each component of the layout and logical structures, and then extracting information from the document according to the corresponding parts.

The process of classifying an incoming document proceeds through three-level matching -- string matching, layout matching and logical matching. We want to achieve time-efficiency by narrowing down the compared set of document samples level by level. In the first two levels of matching, we only focus on the matching on the physical view. Only in the third level, the logical meanings are verified node by node.

Firstly, the string representation matching is used to determine the match between two string representations of a sample Labeled Directed Weighted Graph of a document type from the sample base and the Labeled Directed Weighted Graph of the incoming document. String matching is used mainly for two reasons. 1) By excluding the search for

a match between any two layout structures without traversing different layout structures, it reduces the sequenced time if only matching their two string representations is used instead. 2) Instead of finding the possibly common sub-pattern of two layout-structures, it is easier to find the common sub-string from two string representations. Then we can eliminate the search of a match between the layout structure of a given document and all the other sample layout structures which are definitively different from it.

Secondly, perform the layout structure matching between two Labeled Directed Weighted Graphs. The process of finding a match between two Labeled Directed Weighted Graphs helps to resolve any vague and ambiguous match from the first step. After this stage, no matter if the matching is a perfect match or an approximate match, blocks of incoming documents have their effective matched blocks in sample documents.

Finally, logical matching is applied to decide finally whether the incoming document is of a document type. Therefore, if there exists a case that the layout structure of an incoming document is the same as the layout structures of two different document types from the sample document base, the incoming document still can be classified correctly based on the logical matching (i.e., matching of two logical structures).

A global scenario of classifying an incoming document is given as follows:

An incoming document accompanied with its own string representation (denoted as $string_x$) and its Labeled Directed Weighted Graph (denoted as $LDWG_x$) is an input of the classification subsystem. A possible structure in the document sample base is shown in Figure 5.1. The $string_x$ is compared with the string representations of a document type, one by one, to find a string of the document type, which is "most similar" to the $string_x$. Then apply the same string matching between the $string_x$ and the string representations

for all the other document types. For each of the string representations which are "most similar" to the $string_x$, if the similarity_degree between the $string_x$ and a string

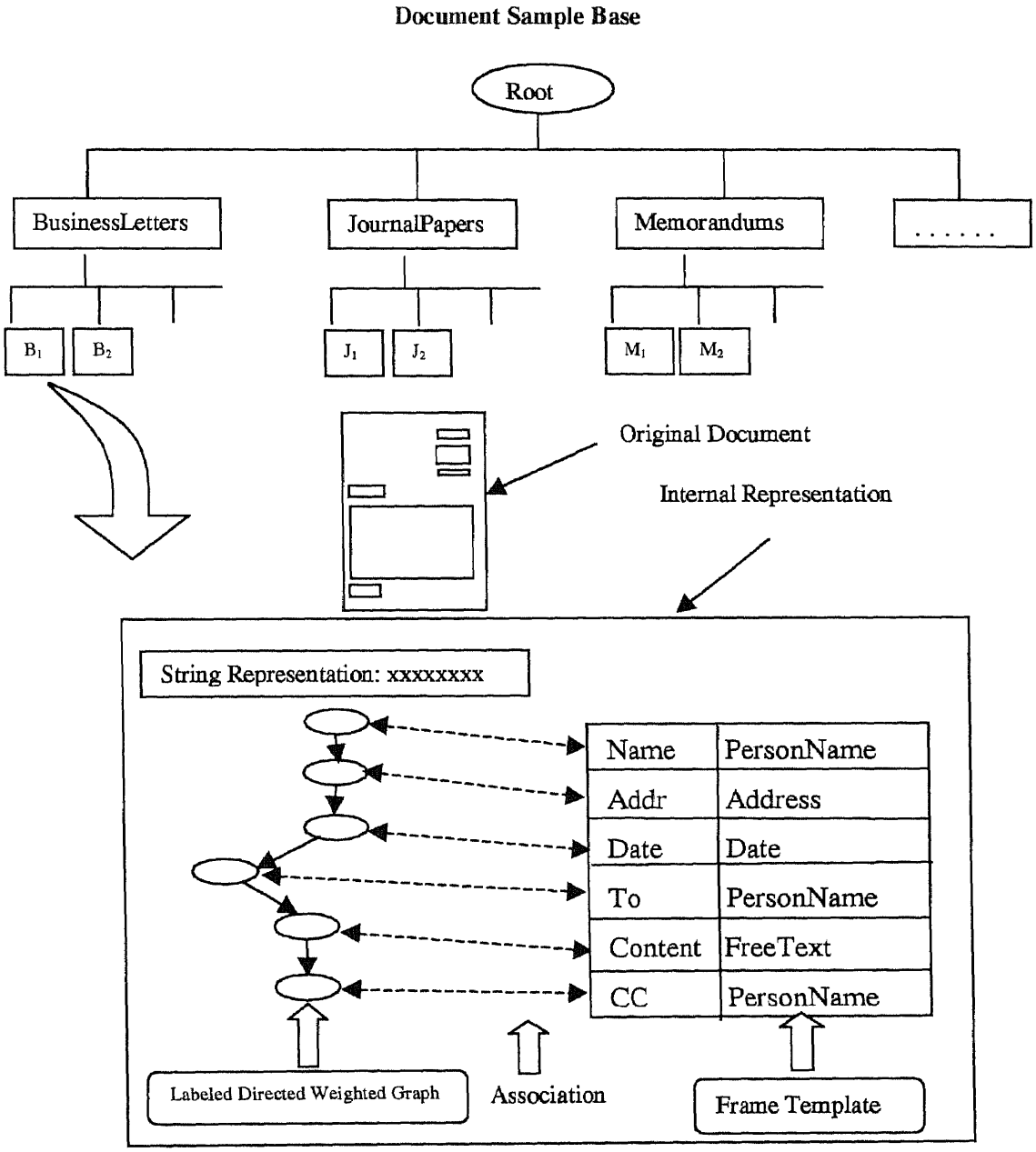


Figure 5.1 Structure of document sample base

representation is larger than the pre-defined threshold, then the layout structure of the string representation will be used to compare with the layout structure of the $string_x$. By traversing the layout structure represented by a LDWG, compare each corresponding component of the LDWG and each component in the $LDWG_x$. Finally, compare the logical structures of the $string_x$ and the string and check the compatibility of their corresponding data type of the labeled components.

The main body of the document classification is as follows:

Input: The string representation and derived LDWG of the incoming document.

Output: A document type is assigned to the incoming document after classification.

Function Classification($String_x$, $LDWG_x$)

```
{       //predefined three threshold θ1, θ2 and θ3
        /* String matching */
        for each document type T
        {
                max_similarityT=0;
                for each different string representations (StringA) of T
                { S=string_match( Stringx and StringA);
                if (S> max_similarityT )
                        replace max_similarityT by the new similarity S;
                 if max_similarityT equals to 1 then break; //break for next document type
        }
        /* For each document type T, find a string, which has the maximum similar
        degree with Stringx */
```

/*Layout matching */

for each document type T with $max\_similarity_T$ is larger than the predetermined threshold θ_1

$degree_D$=layout_match($LDWG_x$, $LDWG_A$); /* Labeled Directed Weighted Graph traversal */

for each document type T with $degree_D$ is larger than the predetermined threshold θ_2

$degree_L$=logical_match($LDWG_x$, $LDWG_A$);

assign the maximum of logical_match $degree_L$ to $Mdegree_L$;

if the $Mdegree_L$ is large than the predefined threshold θ_3

then assign the document type T to the incoming document; //succeeds

else active learning stage.

}//end of Classification

5.1 String Representation Matching

String representation matching is used to eliminate any sample layout structures, which are clearly different from the layout structure of a document to be classified, before applying the time-consuming graph matching. The Approximate String Match (ASM) [101, 102] algorithm is modified for our application. To conduct the string representation matching, we introduce three edit operations, i.e., merge, split and substring.

As we mentioned before, in this level matching, we do not care about the logical meaning of each node. Therefore, we use node name x instead of their original node names.

The *merge* operation is used to replace a part of the incoming string, which is a group, by the corresponding part of the sample string, which is a single block. For example, consider the incoming $string_x$ $H_c(\ V_c(x,x),\ V_c(x,\ V_c(H_c(x,x,),x)))$ and the sample string $H_c(x,\ V_c(x,V_c(H_c(x,x,),x)))$. To match these two strings, Vc(x,x) is replaced by x. as shown in Figure 5.2.

Incoming $String_x$: $H_c(\ V_c(x,x),\ V_c(x,\ V_c(H_c(x,x,),x)))$
Sample String: $H_c(x,\quad V_c(x,\ V_c(H_c(x,x,),x)))$

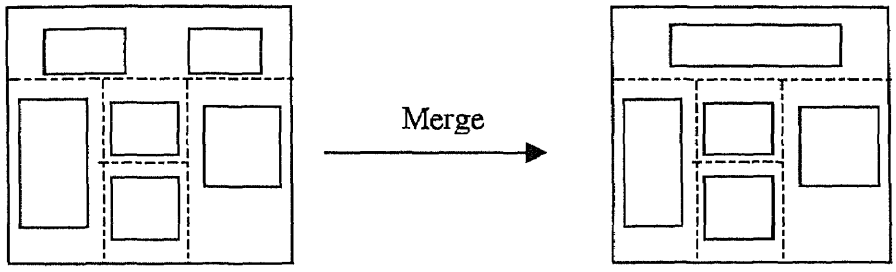


Figure 5.2 Merge operation

The *split* operation is the opposite of the merge. The split operation occurs when the corresponding part of the $String_x$, which is a single block is replaced by a part of a sample string, which is a group. For example, given a $String_x$ $H_c(x,\ V_c(x,\ V_c(H_c(x,x,),x)))$ and a sample string $H_c(\ V_c(x,x),\ V_c(x,\ V_c(H_c(x,x,),x)))$, x is inserted into the layout structure of the $string_x$ to form Vc(x,x). It is illustrated in Figure 5.3.

Incoming String$_x$ $H_c(x, V_c(x, V_c(H_c(x,x,),x)))$
Sample String:: $H_c(V_c(x,x), V_c(x, V_c(H_c(x,x,),x)))$

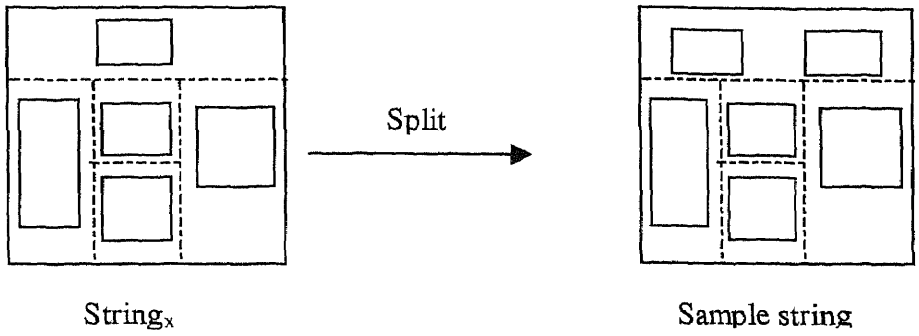


Figure 5.3 Split operation

The *substring* operation is to find a substring within a given string. By doing so, we can determine whether there is a match between this substring and another string. This is the most meaningful part of string matching. It is much easier to find a substring within a string than to find a subgraph within a Labeled Directed Weighted Graph. For example, given an incoming string $V_c(x, H_c(x,H_c(x,x)))$ and a sample string $H_c(x, H_c(V_c(x, H_c(x, Hc(x,x))), x))$. We can use KMP algorithm to find this substring in O(n), but it is very difficult to find the subgraph within the whole graph. Figure 5.4 illustrates the subgraph of a layout structure.

This improved algorithm [103] to match a substring with a string is invented by D. E. Knuth, V. R. Pratt and J. H. Morris at the same time. So it is also called KMP algorithm for simplicity. This algorithm can finish the string matching in O(n+m) if the length of string is n and the length of the substring is m. Its important improvement is that during the procedure of matching, whenever the matching is missed, instead of

Incoming $String_x$: $V_c(x, H_c(x,H_c(x,x)))$
Sample String: $H_c(x, H_c(V_c(x, H_c(x, Hc(x,x))), x))$

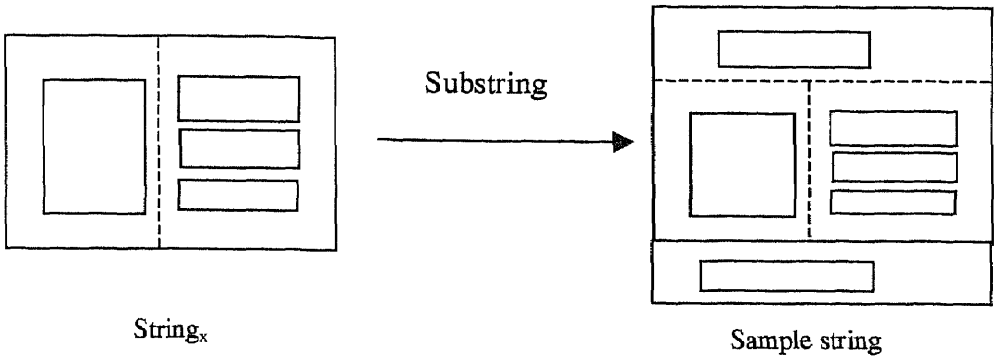


Figure 5.4 Substring operation

turning the pointer back, the matching continues in the same direction and slides a distance based on the "previous partially matched" substring.

Suppose that string s is $s_1s_2...s_n$ and the pattern string t is $t_1t_2...t_m$. In order to improve the traditional matching algorithm, the problem to be solved is: when the character s_i in s is not matching with the character t_j in t, which is the next character in t to continue the comparison with s_i?

Assume that character t_k ($k<j$) is the right character which the comparison should continue with s_i. Then the front (k-1) characters in t must satisfy the following equation and there is such a number k' with k'>k and k' satisfies the following equation:

$$t_1t_2...t_{k-1} = s_{i-k+1}s_{i-k+2}...s_{i-1} \quad (5\text{-}1)$$

In addition, the partially matched result is $t_{j-k+1}t_{j-k+2}...t_{j-1} = s_{i-k+1}s_{i-k+2}...s_{i-1}$ (5-2)

From the above two equations, we can easily get $t_1t_2...t_{k-1} = t_{j-k+1}t_{j-k+2}...t_{j-1}$. (5-3)

In other words, if there are two substrings which satisfy the above equation 5-3, then during the procedure of matching, when character s_i in s is not matching with character t_j in t, the pointer j of string t only needs to slide to k. Because the front k-1 characters $t_1t_2\ldots t_{k-1}$ in t must be the same as k-1 character back from pointer i in s $s_{i-k+1}s_{i-k+2}\ldots s_{i-1}$.

Let next[j]=k, the next[j] is the position that when a mismatching occurs between t_j and s_i , the position that the pointer j should begin with. Then the definition of next[j] can be described as follows.

$$\text{Next}[j] = \begin{cases} 0 & \text{when } j = 1; \\ \max\{k \mid 0 < k < j \text{ and } t_1t_2\ldots t_{k-1} = t_{j-k+1}t_{j-k+2}\ldots t_{j-1}\} & \text{when the set is not empty;} \\ 1 & \text{for any other cases.} \end{cases}$$

KMP algorithm to match a substring with a string is given as follows.

Input: Two strings --string s is the full string and t is a possible substring of s;

Output: The index of the first character in string s if t is a substring of s;

otherwise, return 0;

```
int index_KMP(char* s, char* t)
{
initialize two index pointers i and j ;

while (i not reaching the end of string s and j not reaching the end of string t)
{// If the characters in s and t are matched, then keep going, else slide a distance based on
// table next;
    if ((j == 0) or ( s[i] == t[j] ))
    //The corresponding characters pointed by i in s and j in t are equal.
```

```
        {i++; j++;} //keep going
        else j = next[i]; //find the next table, slide pointer j a distance of next[i]
}

if (j reaches the end of string t)
        return (i-strlen(t)); // return the index of the matched substring in s
else return (0);
}//index-KMP
```

The next table is built up based on the substring t itself and it has no connection with string s. From the previous discussion, next[j]=max{k| 0<k<j and $t_i \ldots t_{k-1}=t_{j-k+1} \ldots t_{j-1}$} if the set is not empty. The method of finding k can take advantage of KMP again.

```
//Build up the next table, which is called by the above function.
void get_next(char *t)
{
        //Initialize the variables
        int j, k;
        j=0; k=-1; next[0]=-1;
        while(j not reaching the end of t)
        {
                if((k==-1) || (t[j+1]==t[k+1]))
                {j++; k++; next[j]=k;} //keep going
                else {k=next[k];}
```

}

}//get_next

Assume that the string *s* = "ababc**abcac**bab" and the string t = "abcac". Next table is ready and as shown in Figure 5.5. The procedure of matching is illustrated in Figure 5.6.

j (pointer in *t*)	1	2	3	4	5	6	7	8
string *t*	*a*	*b*	*a*	*a*	*b*	*c*	*a*	*c*
next[j]	0	1	1	2	2	3	1	2

Figure 5.5 Next table of string t

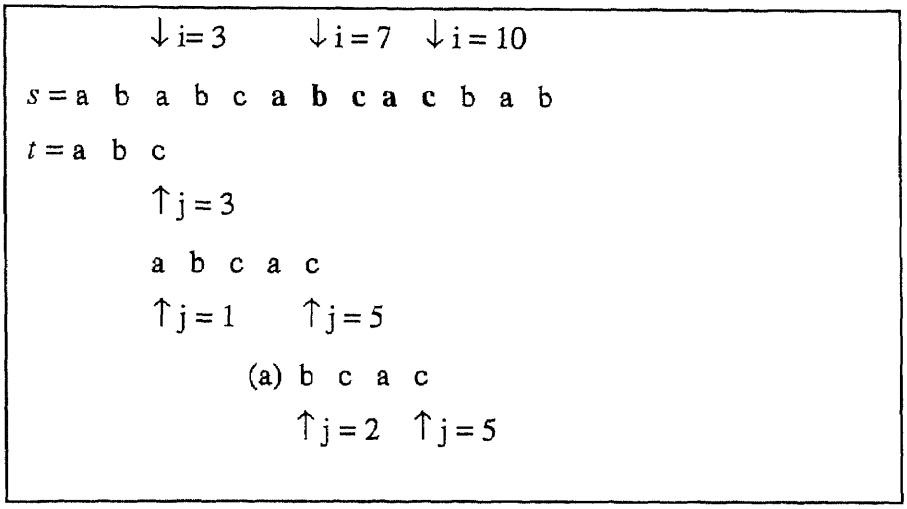


Figure 5.6 Illustration of KMP algorithm

Next, we shall present a way for computing the degree of similarity of two string representations. As we have mentioned in chapter 3.1.4, there is a weight associated with each block in a layout structure. Because the final matching between two logical

structures is based on the weights, the similarity of two strings should take the weight percentage into consideration when the approximate string match is applied.

In comparison with the $String_x$, within each document type, only the string with the maximum similarity_degree will be kept for this document type. In order to prevent the waste of computing time consumed by the unnecessary matching between the $String_x$ and the other strings of the same document type, two principles are observed:

1) If there is a match with the similarity_degree 1, then there is no need to match further the $String_x$ to the other strings within the same document type, because there are no two identical layout structures within the same document type to be kept in the sample base.
2) If the *possible maximum similarity_degree* of the next matching between the $String_x$ and the other strings within the same document type is already less than the current maximum similarity_degree, which is not 1, then no more matching is needed because further matching will only generate less similarity_degree.

For the split operation, when comparing a string $Op_A(A, B)$ with another string $Op_x(X, Y)$, at least one of the A or B is in the form of Operator($Operand_1$, $Operand_2$) and the corresponding character X or Y in $String_x$ is a single block. In this case, the matching will have a possible maximum similarity_degree between either X or Y in the $String_x$ and one of $Operand_1$ and $Operand_2$ with the *larger* weight.

Assume that $Weight_1$ of the $Operand_1$ is larger than $Weight_2$ of the $Operand_2$, then the similarity_degree between either X or Y in the String and the $Operand_1$ is atmost

$$\frac{1-\text{Weight}_2}{\sum_{i=1}^{n}\text{Weight}_i}$$

where n is the total number of blocks in the sample string.

For the substring operation, the new similarity_degree between the String_x and the string in the sample base will be computed by multiplying the current similarity_degree by the *factor* of

$$f = \frac{\sum_{j=1}^{m}\text{Weights}_j}{\sum_{i=1}^{n}\text{Weights}_i}$$

as the penalty, where n is the total number of the blocks in the sample string and m is the total number of blocks in the substring.

The methodology of string representation matching is as follows: use two pointers to scan the two strings from left to right, character by character. The two pointers continue to move to the right in the strings if the compared characters are the same. Otherwise, apply the approximate matching using one of the three operations, merge, split and substring.

```
String_match(String_A, String_x)
{
/* scan two strings from left to right */
while( neither of the two pointers reaches the end of string )
```

```
{       if (Op_A= =Op_x)
            if(Operand_A!= Operand_x)
            { if( split operation)
                  modify the similiarity_degree by multiplying it by the factor f;
            /*if it is merge operation, then keep going*/
            }
            /* if it is an exact match, then keep going */
        if (Op_A != Op_x)
        /* check whether String_x is a substring of String_A */
        substring(String_A, String_x);
        modify the similiarity_degree by multiplying it by the factor f;
}
```

5.2 Matching between Layout Representations

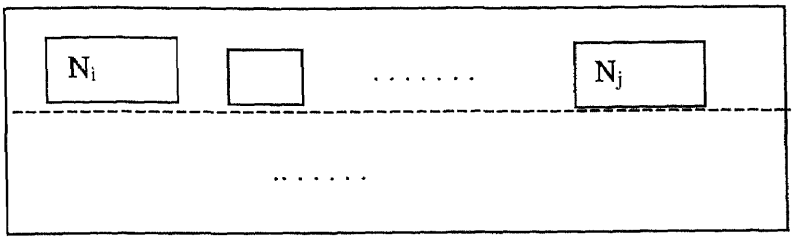
After string representation matching, only small set of samples are left from first step matching to apply further layout matching. In this chapter, we prove every Labeled Directed Weighted Graph is acyclic and then introduce the way of traversal of the entire graph by visiting every node once and passing through every labeled directed edge once in topological order.

Property: Every Labeled Directed Weighted Graph derived from a document's layout structure is acyclic.

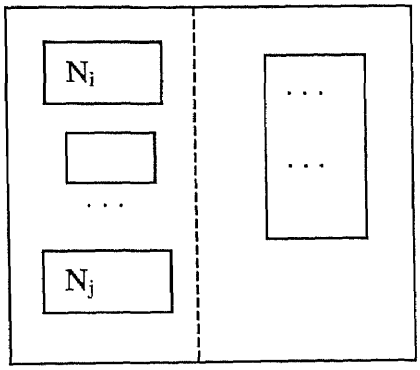
Proof: For simplicity, we discuss the following four typical cases.

If the LDWG is cyclic, which means there is at least one cycle in the LDWG, then there exists at least one pair of vertices N_i and N_j, for which there is a directed path from N_i to N_j and there is also a directed path from N_j to N_i. In order to prove the LDWG is acylic, we will prove that there is no such possible pair in LDWG.

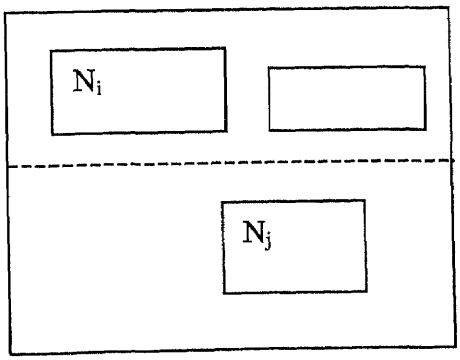
For a pair of vertices N_i and N_j, there have the following cases of spatial relation between these vertices as shown in Figure 5.7:



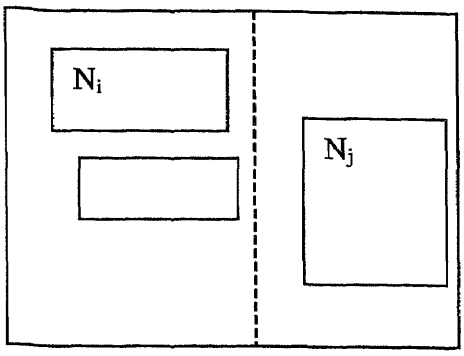
(a) N_i and N_j are cut in one group by a horizontal cut



(b) N_i and N_j are cut in one group by a vertical cut



(c) N_i and N_j are cut in two groups by a horizontal cut



(d) N_i and N_j are cut in two groups by a vertical cut

Figure 5.7 Four typical cases of spatial relation between vertex N_i and N_j

Note that, for two horizontally adjacent vertices, a directed edge is drawn from the left part to the right part. For two vertically adjacent vertices, a directed edge is drawn from top to bottom. From the above cases, we can see that there is only way path atmost if any, it is impossible to have two-direction path existing between a pair N_i and N_j. This is true for any pairs of vertices in the LDWG. Therefore, all LDWGs are acyclic.

Also, it is well known that a topological sorting can be found if and only if the directed graph is acyclic. Therefore, we can find a traversal of a LDWG in topological order.

Acyclic Directed Graph Traversal Algorithm:

Input: A rooted, acyclic, Labeled Directed Weighted Graph G is given.

Output: A layout of all the vertices with labeled directed edges of the given graph in a topological sorting order (i.e., a linear order to meet the prerequisite rules of the topological sort) is obtained.

Method:

Topological sort algorithm:

1. Read a list of edges between vertices of the graph and then to form its adjacency list and an indegree array
2. Place vertices of indegree 0 on Queue
3. While Queue is not-empty do

    3.1 Delete head of Queue x and number x;

    3.2 For every neighbor y of x do

        Decrease indegree(y);

        If indegree(y)=0 then add y to Queue.

In our case, for an acyclic Labeled Directed Weighted Graph, we can use the topological sort to generate a linear order of vertices of the graph, in such a way that each of the vertices is listed only once in the linear expression. The traversal of the graph ensures that we (1) visit all vertices of the graph and visit each vertex once only; and (2) traverse all edges according to the given directions, and traverse through each labeled directed edge once only.

Furthermore, no traversal is allowed to travel through the labeled directed edge in its opposition direction.

It should be noted that a list of triples is obtained, in which the triples ($Node_i$, L_{ij}, $Node_j$) are listed according to the order in which all the nodes $Node_j$ are visited from the node $Node_i$. Then follow by another sublist of triples ($Node_j$, L_{jk}, $Node_k$)'s for each of the nodes $Node_j$'s, to reach the other nodes $Node_k$'s if they are connected by labeled directed edges labeled with L_{jk}'s, which have not been marked.

In the following, we shall give the algorithms to determine an approximate, possibly exact, match of two given graphs.

Algorithm for an approximate match

Input: Two Labeled Directed Weighted Graphs in topological sorting order are given.

(Two lists of triples)

Output: Generate a message for an approximate match of the two graphs and all the pairs of matched blocks between an incoming document and a sample document.

Method: A triple is matched if and only if nodes are matched and their spatial relationship (as label on edge) is also matched.

Two graphs are said to have a match if their triples of the graphs are exactly matched. Two graphs are said to have an approximate match, if most of the triples of the graphs are matched.

5.3 Matching between Logical Representations

The layout structures of documents can be represented in terms of acyclic Labeled Directed Weighted Graphs. Since every acyclic Labeled Directed Weighted Graph can be represented by a layout of its vertices in a topological sorting order, which could be transformed into topological sorting list, the layout matching becomes the comparison between two topological sorting lists. Here, our assumption is that the non-hierarchical graph is used, in which the node does not contain any sub-graphs at all.

For a document, which derive slightly from its standard document of the same document type, the missed match between this document and the standard document

could be prevented by normalizing a given document using the segmentation history as discussed in chapter 2.4.

After the match between the layout structures of two documents at the first step, we proceed further a logical matching between the logical representations.

Although the layout structure matching (or layout matching) is required at the very first step of matching, the logical structure within each block may have different weight for classifying a document. Usually, a human being can classify a document by a glance at the document format. For any document with uncertainty or for further verification, he/she may take a further look at some keywords in a particular position. Our methodlogy of logical matching simulates this typical behavior of a human being.

Its main idea is that the blocks with larger weights in the whole document have higher priority than those blocks with smaller weights. If the partial summary of matched blocks with larger weights is already larger than the threshold, then the logical matching is finished before any further match of other blocks with smaller weights need to be performed. Hence, the matching procedure is as follows: from all of the unvisited block, select a block, whose weight is largest among the weights of the blocks; check the logical equivalence of two corresponding blocks and then check whether the returned match degree times the weight is larger than or equal to the predefined threshold for the matching. If it is, then the whole logical structure matching is completed. Otherwise, find the block among all the remaining unvisited blocks with largest weight. Repeat the same precedure as above until either the successful match is found or match fails with every vertices visited.

Input: Two Labeled Directed Weighted Graphs with all the effective matched pairs.

(One is an incoming document; the other is one of the sample documents passed from layout structure matching.)

Output: The incoming document is either logical matched or it is not matched.

Function: Logical matching of two LDWGs.

If either graph is an empty graph, exit;

Repeat

Find the block whose weight is the largest one among all of the blocks which have not yet been vistied. (This may not be the root);

//block1 is a block in sample document and block2 is matched block in

// incoming document after layout structure matching

Call the block_match(block1, block2);

// to compare the attributes of the corresponding blocks and return the

// degrees of block match.

If Σweight*block_match(block1, block2) > θ return "matched";

Until no more blocks unvistied;

Return "not matched".

```
Int block_match(b1, b2)
{//remember that b1 is the block in sample document;
    switch(b1)
    { b1 is a keyword:
```

```
            if b2 is exactly the same as b1 or b2 is equal to one of the terms in
            b1's thesarus, then return 1;
            else return 0;
        b1 is a value:
            if this value is defined by pattern, then
                if b2 is satisfied with one of the patterns defined for this type,
                    then return 1;
                    else return 0.
            if this value is free text,
                then return 1;
        }
}
```

5.4 Automatic Document Extraction

After the classification, the logical structure of the incoming document is identified. The assigned logical structure with a document type including its frame template definition, layout structure and the connections (or called the associations) between each component of these two structures are obtained. This means the system finds a mask for the previous unknown document. And in fact, based on the frame template, information is extracted from the document. Such information extraction from the document requires another traversal of its corresponding Labeled Directed Weighted Graph as shown in Figure 5.8.

In Figure 5.8, we can see that even though these two documents are not in exactly the same format, the extraction can still be done effectively. In addition, from this

example, it indicates that the cutting and LDWG's transformation method are unique both in sample document and incoming document. Its cutting method excludes the effects caused by different font size and different line spacing. Therefore, after segmentation and cutting and transforming into LDWG, these two LDWGs are "similar" and are easily matched. It resolves the problems that may arise in nested-segmentation method and tree matching in [6,7].

Some keywords, such as "To" in incoming document and "Receiver" in sample document are logical equivalent and their equivalent relation is supported by the thesaurus stored in the knowledge base [16].

For composite values, such as date, it maybe 07/01/99 in the pattern Month/Day/Year(2) in sample documents and it maybe May 11, 1999 in the pattern Month Day, Year(4) in the incoming document. These pattern recognition problems are supported by data-domain mechanism in knowledge base [17].

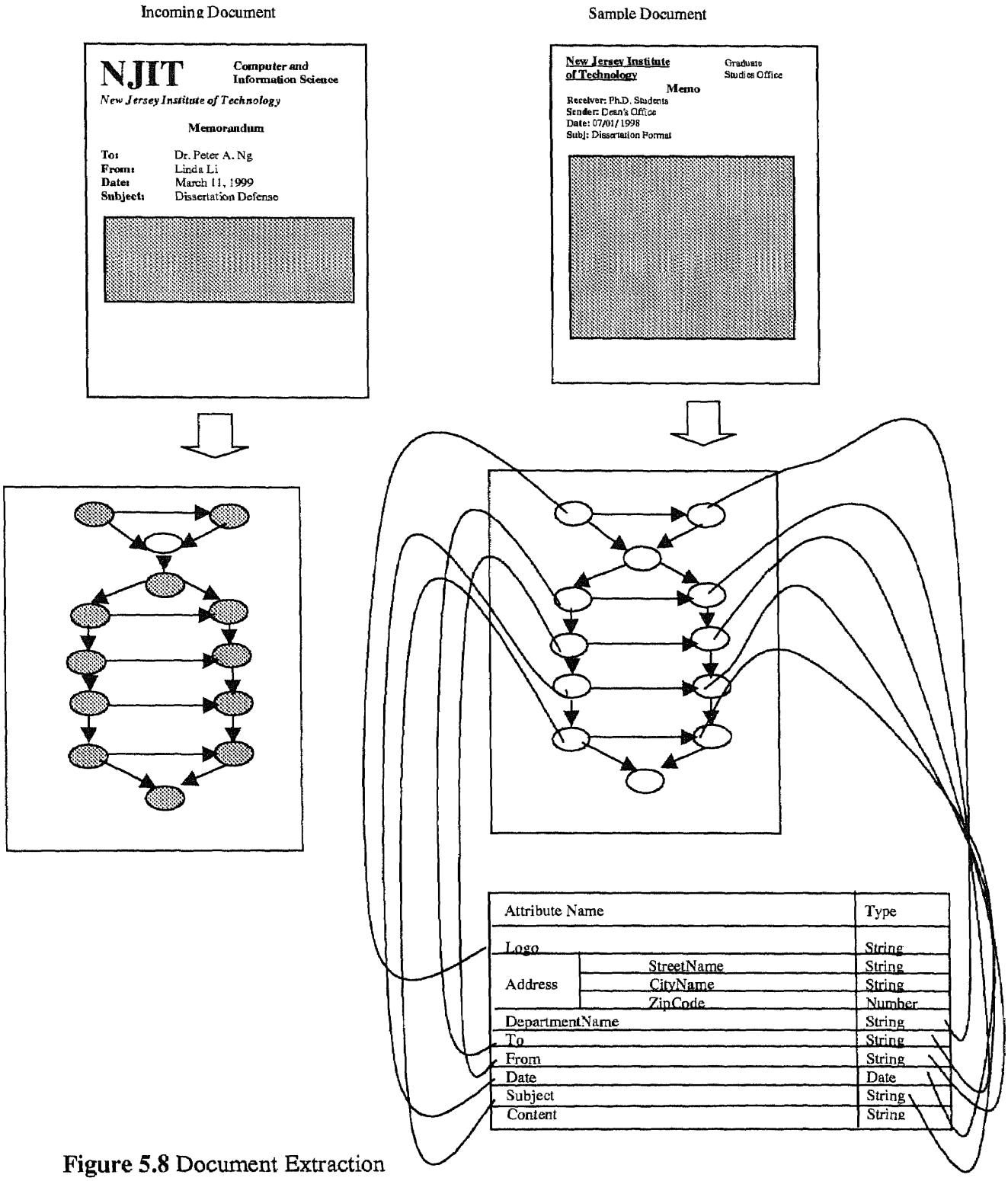


Figure 5.8 Document Extraction

CHAPTER 6

FURTHER ANALYSIS ON DOCUMENT CONTENT

After we classify successfully a document by identifying its document type and extract the necessary values from the document content, we want to classify further the document according to its content, such as classifying the memos into the meeting memos and QE memos. Each of the documents that belong to the same memorandum document type may have different layout structure, but they share the same logical structure and frame template. In addition, further classification cannot depend on the layout structure or logical structure anymore, but can only depend on the further analysis on the contents or values of some attributes.

Sharing the same frame template is the precondition for using the decision tree algorithm to do the further classification. The decision tree algorithm does not produce the criteria for the further classification. It is used to optimize the original decision tree for a particular problem. Its objective is trying to generate a smaller tree with a greedy divide and conquer algorithm. We can not guarantee it will find the smallest decision tree for a given tree, that is a NP complete problem. But we definitely can use this algorithm to find a smaller tree then the original one. Its main idea is to find which attributes have the most information and use them to obtain the "best divide" of the data into uniform subsets.

The Decision Tree Algorithm:

ID3 (D: Data, F: feature)

if D is empty then return (?)

else if all examples in D are in the same class, return (class (D))

else if F is empty, return (majority(D))

else

BestFeature: the feature f in F that minimizes Info(D/f);

Initialize a tree T with BestFeature as Root;

For i =1 to N bestFeature

$S:=D_{BestFeature= i;}$

i^{th} child of T:= ID3(S, F-BestFeature);

return(T).

A simple example of the use of this algorithm is given as follows. This example is only used to demonstrate the key idea of the Decision Tree Algorithm. It will be changed into a particular example on further classification. For simplicity, in this example, documents of memo type are classified into two classes: emergency or non-emergency based on the different value compositions of the three attribute names, To, From and Subject as in Figure 5.8.

	Subject	From	To	Emergency
1	Course	Student	Chair	+
2	Routine	Student	Chair	+
3	Course	Faculty	All	-
4	Meeting	Student	All	-
5	Course	Faculty	Chair	+
6	Meeting	Faculty	Chair	-

Figure 6.1 Further classification based on the value of attributes

How can we find the "best feature"? (In our case, "best attribute")

Notation:

Classes: 1,...,k for k class problem;

S: set of data;

P_j: proportion of S of $class_j$;

Attribute f;

Values of attribute f's is a set of 1,2,... , n_f;

$S_{f=i}$: subset of S where attribute f has value i;

From informtation theory, we get information content of a data set S:

$$Info(S) = -\sum_{j=1}^{K} P_j * \log_K P_j$$

For the given example, K=2

$$\begin{aligned} Info([1,2,3,4,5,6]) &= -\sum_{j=1}^{2} P_j * \log_2 P_j \\ &= -(P_1 * \log_2 P_1 + P_2 * \log_2 P_2) \\ &= -(0.5 * \log_2 0.5 + 0.5 * \log_2 0.5) \\ &= 1. \end{aligned}$$

$$\begin{aligned} Info([1,2,5]) &= -(P_1 * \log_2 P_1 + P_2 * \log_2 P_2) \\ &= -(1 * \log_2 1 + 0 * \log_2 0) \\ &= 0. \end{aligned}$$

Information content of S given f:

$$Info(S \mid f) = \sum_{i=1}^{nf} \frac{|S_{f=i}|}{S} * Info(S_{f=i})$$

For example:

$$Info([1,2,3,4,5,6] \mid From)$$

$$= \sum_{i=1}^{2} \frac{[1-6]_{From=i}}{6} * Info([1-6]_{From=i})$$

$$=0.5*\text{Info}[1,2,4]_{\text{From=Student}} + 0.5*\text{Info}[3,5,6]_{\text{From=Faculty}}$$

$$=0.5*(-(2/3*\log_2(2/3) + 1/3*\log_2(1/3)) +0.5*(-1/3\log_2(1/3)+2/3*\log_2(2/3))$$

$$=0.918$$

In the same way, compute the value for

Info ([1,2,3,4,5,6]|Subject)=0.549

Info ([1,2,3,4,5,6]|To)=0.459

Comparing these values we find that the Info ([1,2,3,4,5,6]|To) which is the smallest. Therefore, feature "To" is the "best" feature and it is chosen the root of the decision tree.

One of our needs is to sub-classify a document type into various meaningful subtypes. For example, a journal type of documents can be subdivided into a technical journal type, a management journal type, a computing journal type, a financial journal type, etc. Another example is to subdivide a letter type into a business letter type, a legal letter type, a personal letter type, etc. In the sample base, for each type of documents, we have various layout and logical structures of the document of the same type. This arises from the fact that a class of documents can be divided into various subclasses because of the nature of their contents and the physical appearances of the documents. For example, the nature of the content and the physical appearance of a legal letter is very much different from the nature of the content and the appearance of a business letter, which is

in turn, very different from the content and the appearance of a memo. The layout structures of subclasses of the documents of a class can be different because of its physical appearances. The logical structures of the subclasses of the documents of a class can be significantly different because the nature of their contents is different. In the previous chapters, our document classification is defined based on the combination of the layout structures with a decent deep matching of logical structures. In this chapter, we investigate and search for a way for dividing a class of documents into subclasses of documents, when the layout structures of the class and its subclasses are quite the same.

CHAPTER 7

CONCLUSION AND FUTURE RESEARCH

7.1 Conclusion

In this dissertation, a systematical methodology is established. The experimental system can successfully classify most of the existing office documents as predefined document types and extract significant information from the semi-structured documents to build up frame instances. Compared with the existing system and methodology, this system has the following characters.

1. Most existing systems use a rule base to do the logical identification for each unit in the document. But users have to write down those rules for their own application domain for themselves. Usually, for one document type, it requires hundred of rules. During the usage, whenever a rare case occurs, users have to add new rules or modify the existing rules. When users want to deal with a new document type, they have to repeat the whole procedure from the beginning. In contrast, we use "learn from examples". This methodology releases users from the work of creating the rules used by the system to establish document types. The knowledge and rules are stored in the samples. The system provides a way to take the knowledge from sample documents and apply it to a new unknown document. This methodology is applicable to most semi-structured documents widely used in office, library and press, such as: memo, letter, article, report, etc.
2. Absolute block positions and statistical methodology are another widely used way. But this method makes the assumption that the documents in one type

share similar font size, line spacing in each part, which is not true in the real world. In this dissertation, we established a Labeled Directed Weighted Graph (LDWG) to represent the layout structure of a document. In this representation, relative spatial relations are recorded for adjacent blocks instead of recording the fixed absolute position of each block. In this way, we provide much more flexibility in dealing with documents that use various font sizes.

3. String representation for a document is a novel part in this dissertation. String comparison is done to exclude those obvious different samples compared with the structure of an incoming document before the time consuming graph matching is applied. This is particular necessary when the system is huge and document sample base is large.
4. Existing segmentation methods are effective to separate a whole image file into different blocks such as: text_block, image_block, graphics_block and table_block. These methods do not perform further analysis on the content of the text blocks in order to separate the text block into logical units. In this dissertation, segmentation based on the logical closeness is another contribution in this research area.
5. The traditional Perceptron Learning Algorithm (PLA) is augmented in this dissertation to derive and adjust the weights associated with each block in the document layout structure automatically.
6. In this dissertation, we try to design a system which can be easily used by general users. The methodology used in this system simulates the behaviors of

a human being in doing the same task. We provide a user-friendly interface to get knowledge from the interactive usage of users. Users can feel the system improve itself and become "smarter" during their natural usage.

This system is independent with the OCR package so it can deal with the document in different languages and in handwriting.

As we have mentioned before, we are focus on dealing with semi-structured documents. In this dissertation, we focus on the extraction of the structured part of any given document, which is the most implicit representative information, without analyzing the content of the free text content part, which involves natural language processing -- another research area.

In addition, we use a rectangle instead of a polygon to represent a block for simplicity. This is acceptable in dealing with most documents used in the office. But once again, it is not applicable to an article in newspaper, which may not use a rectangles for composing purposes.

7.2 Future Research

Future research can be separated into two directions -- to enhance the system itself to make it have more capable and to explore the system with other research direction.

1. Enhance the system:

   We present a system which can process paper documents properly. We should consider dealing with electrical files in MicroSoft Words, Write, Postscript, HTML etc and email with the attachments to allow the paper document and electrical files

managed in the same system. We also assure that the system supports automatic routing or content-based retriveal based on some criteria.

2. Enhance the system with other research areas

   This system can be used as a personal system or a shared system. For instance, a digital library is a possible important application domain of this system, which is a sharing system. In a distributed system, security problems and data consistence problems are under research. In addition, when the document base is huge, data access may become a bottleneck. Therefore an efficient storage plan among multiple disks and parallel processing, multi-thread are also under research.

REFERENCES

1. Yuan Y. Tang, Chang De Yan and Ching Y. Suen, "Document Processing for Automatic Knowledge Acquisition," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 6, No.1 pp3-21, 1994.

2. S. TSUJIMOTO and H. ASADA "Understanding Multi-articled Documents," *10th International Conference on Pattern Recognition*, Atlantic City, New Jersey, 1990.

3. J. Higashino, H. Fujisawa, Y. Nakano and M. Ejiri, "A knowledge-based segmentation method for document understanding," *Proceedings of 8th International Conference on Pattern Recognition*, pp745-768, 1986.

4. John F. Cullen, Jonathan J. Hull and Peter E. Hart, "Document Image Database Retrieval and Browsing using Texture Analysis". *Proceedings of the Fourth International Conference on Document Analysis and Recognition*, pp718-721, August 1997.

5. Xiaolong Hao "Automatic Office Document Classification and Information Extraction", *Ph.D. Dissertation*, Department of Computer and Information Sciences, New Jersey Institute of Technology, Newark, New Jersey, October 1995.

6. X. Hao, J.T.L. Wang, M. Bieber, and P. A. Ng, "A Tool for Classifying Office Document," *Proceedings of the 5th IEEE International Conference on Tools with Artificial Intelligence*, pp427-439, November 8-11 1993.

7. X. Hao, J.T.L. Wang and P. A. Ng, "Nested Segmentation: An approach for Layout Analysis in Document Classification," *Proceedings of the Second IAPR Conferences on Document Analysis and Recognition*, pp319-322, Tsukuba Science City, Japan, October 1993.

8. ChingSong Wei "Knowledge Discovering for Document Classification Using Tree Matching in TexPros", *Ph.D. Dissertation*, Department of Computer and Information Sciences, New Jersey Institute of Technology, Newark, New Jersey, May 1996.

9. C.Y. Wang, Q. Liu, and P.A. Ng, "Browsing in an Information Repository," *Proceeding of 2nd World Conference on Integrated Design and Process Technology*, IDPT-Vol2, pp48-56, 1996.

10. C.Y. Wang, Q. Liu, and P.A. Ng, "Intelligent Browser for TEXPROS," *ISATED Proceeding of International Conference on Intelligent Information Systems (IIS'97)*, IEEE Computer Society Press, pp388-398, December 1997.

11. Xien Fan and Peter A. Ng, "Personal Document Management and Retrieval: A Knowledge-Based Approach", *Journal of Systems Integration*, vol. 8, No. 3, 1998.

12. Xien Fan, Peter A. Ng, "A Dual Model Approach for Modeling Office Documents," *Proceedings of International Workshop on Issues and Applications of Database Technology*, 1998.

13. X. Li, J. Hu, X. Fan, C. Y. Wang and P. A. Ng, "Automated Document Filing and Retrieval," in *Proceedings of the Third World Conference on Integrated Design & Process Technology*, 1998.

14. Xuhong Li, Jianshun Hu, Zhenfu Cheng, D.C. Hung and Peter A. Ng "Automatic Document Analysis and Understanding System," *The First International Conference on Enterprise Information System*, 1999.

15. Xuhong Li, Jianshun Hu, Zhenfu Cheng, Simon Doong, D.C.Hung, and Peter A. Ng "An Integrated Document Processing System: Document Classification and Information Extraction," *The Fourth World Conference on Integrated Design and Processing Technology, incorporating IEEE International Conference on System Integration*, 1999.

16. Jianshun Hu, Xuhong Li, Simon Doong, D.C. Hung and Peter A. Ng, "A Thesaurus Model for Document Processing System: The TEXPROS Approach," *The Fourth World Conference on Integrated Design and Processing Technology, Incorporating IEEE International Conference on System Integration*, 1999.

17. Jianshun Hu, Xuhong Li, D.C. Hung, Simon Doong and Peter A. Ng, "Managing Knowledge for an Intelligent Document Processing System", *The First International Conference on Enterprise Information System*, 1999.

18. Q. Liu, An Office Document System With the Capability of Processing Incomplete and Vague Queries, *Ph.D. dissertation*, Dept. of Computer and Information Science, New Jersey Institute of Technology, Newark, New Jersey, August 1994.

19. Q. Liu and P. Ng, "A Browser of Supporting Vague Query Processing in an Office Document System", *Journal of System Integration*, Vol. 5, No. 1, pp61-82, 1995.

20. Q. Liu and P. Ng *Document Processing and Retrieval: Text Processing*, Kluwer Academic Publishers, Norwell, Massachusetts, 1996.

21. Hwee Tou Ng, WeiBoon Goh and Kok Leong Low "Feature Selection, Perceptron Learning, and a Usability Case Study for Text Categorization", *SIGIR 97* Philadelphia, Pennsylvania.

22. F. Barbic and F. Rabitti. "The Type Concept in Office Document Retrieval". *Proceeding of VLDB 85*, pp34-47, Stockholm, Sweden, 1985.

23. A. Celentano, M.G. Fugini, and S. Pozzi, "Classification and Retrieval of Documents Using Office Organization Knowledge," *Proceeding of ACM Conference on Organizational Computing Systems*, pp159-164, Atlanta, Georgia, November 1991.

24. H. Fujisawa, Y. Nakano, and K. Kurino, "Segmentation Methods for Character Recognition: From Segmentation to Document Structure Analysis," *Proceedings of the IEEE, 80(7):* pp1079-1091, July 1992.

25. K.S. Jones, "Notes and References on Early Automatic Classification Work," *SIGIR Forum*, pp10-17, Spring 1991.

26. T.W. Malone, K.R. Grant, and K. Lai, "Semistructued Messages Are Surprisingly Useful for Computer-Supported Coordination," *ACM Transaction on Office Information System*, 5(2): 115-131, April 1987.

27. B. Pagurek, N. Dawes, G. Bourassa, G. Evans, and P. Smithers, "Letter Pattern Recognition," *Proceeding of IEEE Sixth Conference on Artificial Intelligence Application*, pp313-319, 1990.

28. B.W. Porter, R. Bareiss, and R.C. Holte "Concept Learning and Heuristic Classification in Weak-Theory Domains," *Artificial Intelligence*, pp229-263, April 1990.

29. E. Riloff. "Using Cases to Representing Context for Text Classification," *Proceeding of AAAI Spring Symposium on Cased-Based Reasoning and Information Retrieval*, 1993.

30. E. Riloff and W. Lehnert. "Information Extraction as a Basis for High-Precision Text Classification," *ACM Transactions on Information Systems*, 12:269-333, 1994.

31. S. Tsujimoto and H. Asada, "Major Components for a Complete Text Reading System," *Proceedings of the IEEE*, 80(7):1133-1149, July 1992.

32. S.J. Hong, "Developing Classification Rules from Examples," *Tutorial for the International Conference on Artificial Intelligence for Applications*, pp1-37, Orlando, Florida, March 1993.

33. E. Lutz, H.V. Kleist-Retzow, and K. Hoernig, "MAFIA-An Active Mail-Filter-Agent for an Intelligent Document Processing Support," *Multi-User Interface and Applications,* eds., S.Gibbs and A.A.Verrijn-Stuart, Elsevier, Science Publishers, North Holland, pp16-32, 1990.

34. J. Schmdit and W. Putz, "Knowledge Acquisition and Representation for Document Structure Recognition," *Proceedings of the 9th Conference on Artificial Intelligence for Applications: the CAROL project*, pp177-181, Orlando, Florida, March 1993.

35. R. Yasdi, "Learning Classification Rules from Database in the Context of Knowledge Acquisition and Representation," *IEEE Transactions of Knowledge Data Engineering*, 3(3): 293-306, September 1991.

36. J. P. Bixler, "Tracking text in mixed-mode document," In *Proceeding of ACM Conference of Document Processing Systems*, pp177-185, 1988.

37. A. Dengel and G. Barth, "High Level Document Analysis Guided by Geometric Aspects," *International Journal of Pattern Recognition and Artificial Intelligence.* Vol 2. No 4 pp641-655, 1988.

38. A. Dengel, "Document image analysis-expectation-driven text recognition," in *Proceeding of Syntactic and Structural Pattern Recognition.* (SSPR90), pp.78-87, 1990.

39. D. G. Elliman and I. T. Lancaster, "A review of segmentation and contextual analysis techniques for text recognition," *Pattern Recognition* Vol 23, No3/4, pp337-346, 1990.

40. F. Esposito, D. Malerba, G. Semeraro, E. Annese, and G. Scafuro, "An experimental page layout recognition system for office document automatic classification: An integrated approach for inductive generalization," *Proceeding of 10th International Conference on Pattern Recognition*, pp557-562, 1990.

41. H. Fujisawa and Y. Nakano, "A top-down approach for analysis of documents images." *Proceeding of SSPR90*, pp113-122, 1990.

42. J. Higashino, H. Fujisawa, Y. Nakano, and M. Ejiri, "A knowledge-based segmentation method for document understanding," In *Proceeding of 8th International Conference of Pattern Recognition.* pp745-748, 1986.

43. S.C. Hinds, J.L. Fisher and D.P. D'Amato, "A document skew detection method using run-length encoding and the Hough transform" In *Proceeding of 10th International Conference on Pattern Recognition.* pp464-468, 1990.

44. W. Horak, "Office document architecture and office document interchange formats: Current status of international standardization," *IEEE Computing* pp50-60, October.1985.

45. ISO 8613: Information Processing-Text and Office Systems-Office Document Architecture (ODA) and Interchange Format, *International Organization for Standardization*, 1989.

46. O. Iowaki, H. Kida, and H. Arakawa, "A segmentation method based on office document hierarchical structure" *Proceeding of IEEE International Conference on System Man Cybernetics*, Alexandria, Virginia, Oct., 1987, pp759-763.

47. H. Makino, "Representation and segmentation of document images," In *Proceeding of IEEE Computer Society Conference on Pattern Recognition and Image Processing*, pp291-296, 1983.

48. G. Nagy, S.C. Seth, and S.D. Stoddard, "Document analysis with an expert system" in E.S.Gelsema and L.N.Kanal, Eds., *Pattern Recognition Practice II*. New York: Elsevier, 1986, pp149-159.

49. R.G. Reynolds, J.I. Maletic, and S.E. Porvin, "PM: A system to support the automatic acquisition of programming knowledge," *IEEE Transaction on Knowledge and Data Engineering*. Vol 2, No 3 pp273-282, 1990.

50. C.Y. Suen, C.D. Yan, and Y.Y, Tang, "Document analysis and understanding: A method for automated acquisition of data and knowledge," *Center for Pattern Recognition and Machine Intelligence (CENPARMI)*, Concordia University Technology. Report, 1989.

51. K.Y. Wong, R.G. Casey, and F.M. Wahl, "Document analysis system," *IBM J Research Development*. Vol 26, No 6 pp647-656, 1982.

52. F. Esposito, D. Malerba, and G. Semeraro, "Automated acquisition of rules for document understanding," *International Conference on Document Analysis and Recognition (ICDAR)*, 1993.

53. F. Esposito, D. Malerba, and G. Semeraro, "A knowledge based approach to the layout analysis," *International Conference on Document Analysis and Recognition (ICADR)*, 1995.

54. T.Hu. *New methods for robust and efficient recognition of the logical structures in documents*, Ph.D. Thesis, No 1076, *IIUF-Universite de Fribourg*, Switzerland, 1994.

55. Thomas A. Bayer and Hanno Walischewski. "Experiments on extracting structural information from paper documents using syntactic pattern analysis" *Proceedings of the 3rd International conference on Document Analysis and Recognition*, Vol 1, pp476-479, 1995.

56. Andreas Dengel and Gerhard Barth. ANASTASIL: "A hybrid knowledge-based system for document layout analysis," *Proceedings of the 11th International Joined Conference for Artificial Intelligence*, pp1249-1254, Michigan, 1989.

57. F. Esposito, D. Malerba, and G. Semeraro, "Multi-strategy Learning for Document Recognition," *Applied Artificial Intelligence*, 8(33-84) 1994.

58. Bruno T. Messmer and Horst Bunke. "A network base approach to exact and inexact graph matching," *Technical Report IAM-93-021*, University of Bern, Institute for informatics and applied mathematics, 1993.

59. Bruno Timothy Messmer. *Efficient Graph matching algorithms for Preprocessed Model Graphs, Ph.D. Thesis* University of Bern, CH, Institute for Applied Mathematics, November 1995.

60. H. Fahmy and D. Blostein, "A Survey of Graph Grammars: Theory and Applications", *Proceedings of the 11th ICPR (International Conference on Pattern Recognition)*, 1992.

61. M. Okamoto and A. Miyazawa, "An Experimental Implementation of a Document Recognition System for Papers Containing Mathematical Expressions," *Structured Document Image Analysis*, pp36-53, 1992.

62. S.Y. Wang, T. Yagasaki, "Block Selection: A Method for Segmenting Page Image of Various Editing Styles," *ICDAR 95*, Vol 1, pp128-133, 1995.

63. Bloomberg, D, Chen, F., "Extraction of text-related features for condensing image documents," *SPIE*, vol 2660, 1996.

64. Doermann, D, Shin, C., Rosenfeld, A., Kauniskangas, H. Sauvola, J., Pietikainen, M. "Development of a General Framework for Intelligent Document Retrieval", *International Association for Pattern Recognition Workshop, Document Analysis Systems*, October. 1996.

65. Hull, J.J. "Document image matching and retrieval with multiple distortion-invariant descriptors," pp383-400, *IAPR workshop, Document Analysis Systems*, 1994.

66. F. Esposito, D. Malerba, and G. Semeraro, E. Annese and G. Scafuro, "An experimental page layout recognition system for office document automatic classification: an integrated approach for inductive generalization," *Proceedings. 10th International Conference on Pattern Recognition*, pp557-562, Los Alamitos: IEEE Computer Society, 1990.

67. T. Bayer, U. Bohnacher, H. Mogg-Schneider, InfoPortLab "An Experimental Document Analysis System," *Proceedings of the IAPR-Workshop on Document Analysis Systems*, Kaiserslautern, Germany, 1994.

68. C. Wenzel, S. Baumann, T. Jager, "Document Classification by Voting of Competitive Approaches," *Proceedings of International Workshop on Document Analysis Systems (DAS' 96)*, pp352-374, Philadelphia, 1996.

69. S. Chen, *Document Layout Analysis Using Recursive Morphological Transforms*, Ph.D. thesis, Univ. of Washington, Seattle, Washington, 1995.

70. R. M.Haralick, "Document Image Understanding: Geometric and Logical Layout," *Proceedings of CVPR (Computer Vision and Pattern Recognition)'94*, pp385-90, 21-23 June 1994.

71. J. Liang, J. Ha, R. Rogers, B. Chanda, I.T. Philips and R. M.Haralick, "The Prototype of A Complete Document Understanding System," *Proceedings of IAPR Workshop on Document Analysis Systems*, pp130-154, 1996.

72. J. Liang, I.T. Philips, J. Ha, R.M. Haralick, "Document Zone Classification Using the Sizes of Connected Components," *Proceedings of the SPIE*, vol 2660, Document Recognition III, pp 150-157, San Jose, California, 1996.

73. M. Yamada and T. Miyasato, "Conversion Method from Document Image to Logically Structured Document Based on ODA", *Trans. IEICE*, 19, 5, pp. 286-295(1990-10).

74. K. Kise, "Layout Model Based Analysis of Document Structure" C Trans. *The Institute of Electronics, Information and Communication Engineers* (IEICE)(D-II), J72-D, C 8, C pp1029-1039, October, 1993.

75. A. Dengel, R. Bleisinger, R. Hoch, F. Hones, M. Malburg, F. Fein, "Office-MAID- A System for Automatic Mail Analysis," Interpretation and Delivery, in:L. Spitz and A. Dengel (eds.) *Document Analysis System*, World Scientific Co. Inc., Singapore, pp52-75, 1995.

76. G. Nagy, S. Seth, M. Viswanathan, "A Prototypical Document Image Analysis System for Technical Journals," *IEEE Computer*, Vol 25, No 7, pp10-24, 1992.

77. Richard Forsyth and Roy Rada, *Machine learning, applications in expert systems and information retrieval*, Ellis Horwood Limited, *Devon, United Kingdom,* 1986

78. Rosenblatt, Frank *The Perceptron: A Probabilistic model for Information Storage and Organization in the Brain*, Psychological Review, 1958.

79. Rosenblatt, Frank *Principles of Neurodynamics*: Spartan Books, New York, 1962.

80. Robert M. Haralic, "Document Image Understanding: Geometric and Logical Layout" *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Seattle, Washington, 1994

81. W. Horak, "Office Document Architecture and office Document Interchange Formats - A Current Status of International Standardization," *IEEE Computer*, 18(10): 50-60, October 1985.

82. W. B. Croft, "User-Specified Domain Knowledge for Document Retrieval," *Proceedings of ACM SIGMOD International Conference on Management of Data*, pp504-509, San Francisco, California, 1987.

83. E. Bertino, F. Rabitti, and S. Gibbs, "Query Processing in a Multimedia Document System," *ACM Transactions on Office Information Systems*, 6(1) 1-41, January 1988.

84. F. Rabitti, *A Model for Multimedia Documents*. Office Automation, Edited by D. Tsichritzis, Berlin, Germany, 1985.

85. J.M. Smith and D.C.P. Smith. "Database Abstractions: Aggregation and Generalization," *ACM Transactions on Database Systems*, 2(2) 105-133, 1977.

86. E. Codd, "A Relational Model for large Data Banks," *Communications of ACM*, 13(6) 377-387, 1970.

87. C. Date, *An Introduction to Database Systems*, Addision-Wesley, Reading, Massachusetts, 1986.

88. J.D. Ullman, *Principles of Database System*, Computer Science Press, New York, 2nd Edition, 1982.

89. L.A. Rowe and M.R. Stonebraker, "The POSTGRES Data Model," In *Proceedings of the 13th International Conference on Very Large Data Bases*, pp83-96, Brighton, September 1987.

90. P. Hoepner. "Synchronizing the Presentation of Multimedia Objects–ODA Extensions," *ACM SIGOIS Bulletin*, 12(1) 19-32, July 1991.

91. D. Woelk, W. Kim, and w. Luther, "An Object-Oriented Approach to Multimedia Databases," In *Proceedings of the 8th International Conference on Very Large Data Bases*, pp1-10, Mexico, September 1982.

92. S. Christodoulakis, M. Theodoridou, F. Ho, M. Papa, and A. Pathria, "Multimedia Document Presentation, Information Extraction, and Document Formation in MINOS: A Model and System," *ACM Transactions on Office Information Systems*, 4(4) 345-383, October 1986.

93. J. Wang and P.A. Ng, "TEXPROS: An Intelligent Document Processing System", *International Journal of Software Engineering and knowledge Engineering*, Vol.15, No. 4, pp171-196, April 1992.

94. J. Wang, F. Mhlanga, Q. Liu, W. Shang and P. Ng, "An Intelligent Documentation Support Environment", *Proceedings of the Fifth International Conference on Software Engineering and Knowledge Engineering*, San Francisco, California, pp429-436, June 1993.

95. M. Snoeck and G. Dedene, "Generalization/Specification and Role in Object Oriented Conceptual Modeling," *Data and Knowledge Engineering*, Vol. 19, No. 2, pp171-195, June 1996.

96. Ide, E. and G. Salton, "Interactive Search Strategies and Dynamic File Organization in Information Retrieval," *The Smart Retrieval System-Experiments in Automatic Document Processing*, eds. G. Salton, Prentice-Hall Inc., pp373-393, Upper Saddle River, New Jersey, 1971.

97. Rocchio, Jr.,. J. J., "Relevance Feedback in Information Retrieval," *The Smart Retrieval System-Experiments in Automatic Document Processing*, eds. G. Salton, Prentice-Hall Inc., pp313-323, Upper Saddle River, New Jersey, 1971.

98. Anil K. Jain and Bin Yu, "Document Representation and Its Application to Page Decomposition," on *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol.20, No.3, 1998.

99. O. Iwaki, H. KIDA and H.ARAKAWA, "A segmentation method Based on Office Document Hierarchical Structures," *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, pp759-763, Alexandria, Virginia, 1987.

100. Y.Y. Tang, H. Ma, X. Mao, D. Liu and C. Suen, "A New Approach to Document Analysis Based on Modified Fractal Signature," *Proceedings of the Third International Conference on Document Analysis and Recognition*, pp567-570, Montreal, 1995.

101. Z. Galil and R. Giancarlo "Data Structures and algorithms for approximate string matching, " *Journal of Complexity*, 20, pp33-72, 1988.

102. E. Ukonnen "On approximate string match," *Proceedings of the International Conference on Foundation of Computation Theory*, Borgholm, Sweden, August 1983.

103. Weiming Yan and Wenming Wu, *Data Structure*, QingHua University Press, Beijing, P. R. China, 1987.