ELECTRONIC PUBLISHING, VOL. 6(4), 457–468 (DECEMBER 1993)

A mixed approach toward an efficient logical structure recognition from document images

TAO HU AND ROLF INGOLD

Institute of Computer Science, University of Fribourg
Chemin du Musée 3
CH-1700 Fribourg, Switzerland

email: hu_tao@cfruni52.bitnet

SUMMARY

This paper presents our efforts to improve the efficiency of a document structure analysis system, which intends to analyse the complete logical structure of a document. The usage of fuzzy logic improves the system robustness; however, the problem of system efficiency was revealed to be critical.

Different techniques have been studied to overcome this problem. Dynamic programming, heuristics, and dynamic threshold are used for parsing, which achieves a linear complexity. A new concept of key step, based on the principle of sub-goals, is incorporated with a multi-pass and mixed top-down analysis strategy, which avoids the combinatorial explosion of the number of search paths. Finally, the paper shows that the error-tolerating parser based on an analysis graph seems more realistic and efficient than an error-correcting parser.

KEY WORDS Document structure analysis Logical structure recognition System architecture Fuzzy logic Top-down analysis Analysis strategy Dynamic programming Heuristics Error-tolerating parser

1 INTRODUCTION

1.1 The goals of our prototype system

Document structure analysis (DSA) is often defined as a process, which transforms the page images of a document into its structural representations: *layout structure* and *logical structure* [1,2]. The logical structure of a document can be further split up into two levels: a *macro structure* and a *micro structure*. The former describes the high-level structure of a document down to fragments including, for example, chapters and paragraphs, (see Figure 1); and the latter refers to the internal structure of a fragment down to characters, for instance, words, numbers, and letters, (see Figure 2).

Most of available systems [3–5] recognise only the layout structure or the partial (macro) logical structure of a document. The goal of our prototype is to recognise a complete logical structure, as done by some recent systems [6–8], but with a more unified approach. Furthermore, our prototype intends to overcome the main obstacles toward a practical DSA system; therefore, the problems of *robustness* and *efficiency* should be considered in the first place. An overview of this prototype has been given in [9], where *fuzzy logic* [10]

CCC 0894–3982/93/040457–12
Received 15 August 1993
Revised 1 December 1993

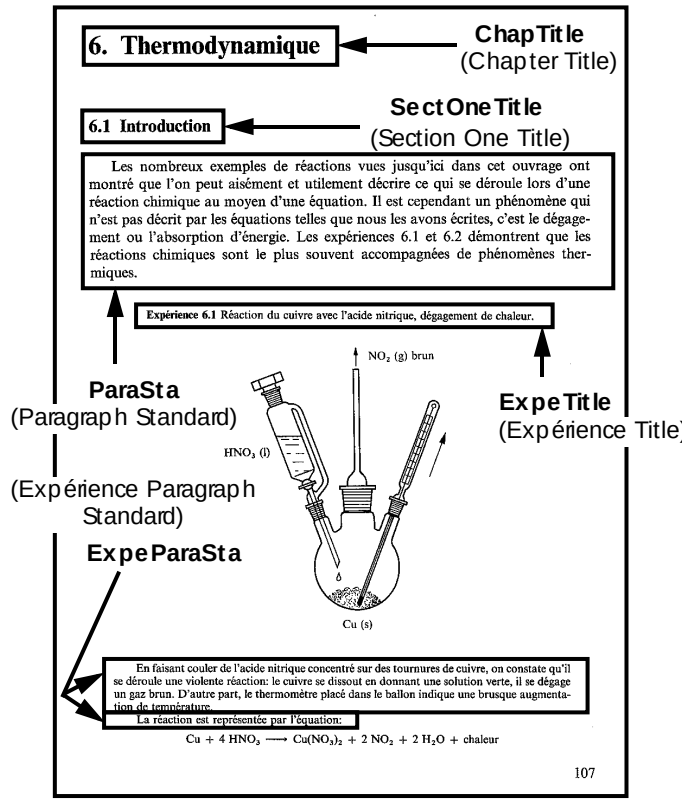

Figure 1. Macro-structure of the first page of the chapter 6 of a scientific book

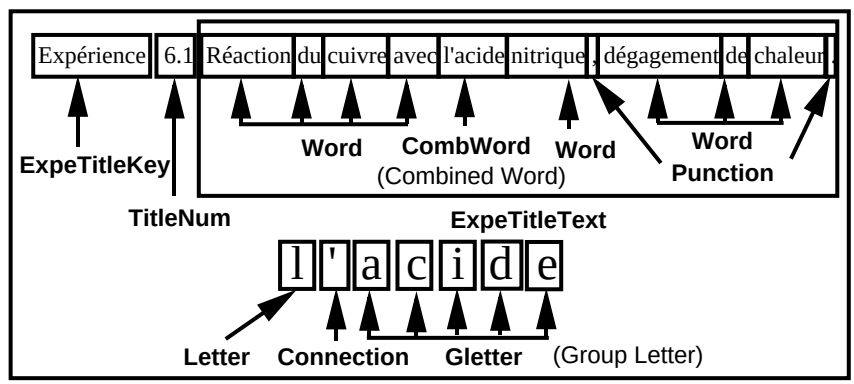

Figure 2. Micro-structure of a subtitle of the first page

is used in primitive matching to achieve the robustness and to tolerate small local errors in the analysis process. This paper will present the main efforts of improving the analysis efficiency.

1.2 Needs for an efficient analysis algorithm

Available DSA systems are mostly at their prototype stage; the problem of analysis efficiency has not yet been solved well. In syntactic pattern recognition [11], parsing often refers to the process of searching a path through a *directed graph* for an input string; and an *error-correcting parsing* is a common method for evaluating string distances. The most popular searching algorithm in some famous DSA systems [12] is the *best-first searching* [13]; but it is an exponential algorithm in the worst case, and it can be used only for some small applications.

Our prototype intends to analyse different document classes, including books and manuals. The important logical entities of a book should be recognized correctly; however, in its other parts, some small local errors are tolerable. Therefore, some input noise from page images and some small errors from basic recognition processes, such as, segmentation and OCR, or from *document description* [14] should be tolerated. Figure 3 shows an example of document description. For analysing a book, it implies the number of lines of the book, or the number of signs of a fragment may be big. The complex logical structures of both levels suggest that the searching spaces are large. Fuzzy logic has been applied [9], which reveals to be very helpful to deal with uncertainty and to check the similarity between an

```
ScieBook: DOC => {Chapter};
Chapter: PRT => ChapTitle {SectOne};
   ChapTitle.zone = main;
   ChapTitle.alignment = (Allowed, Leftadjusted, [-3 pt, 3 pt],
                          [-3 pt, 3 pt], [-3 pt, 3 pt]);
   ChapTitle.lineHeight = [18pt, 18pt];
   ChapTitle.spaceBefore = (Obligatory, [0 pt, 100pt]);
   ChapTitle.interSpace = (Forbidden, [5 pt, 25 pt]);
   ChapTitle.font = (Times, 18pt, Bold, Roman);
   ...
ChapTitle: FRG => LevelOneNum {Word}; ...
Word: STR => {Letter};
   Letter.cand = {"A"|"B"|...};
   Letter.font = (@, @, @, @);
   Letter.separBefore = <FST: (Allowed, [@, @]),
                         NXT: (Forbidden, [0 pt, 2 pt])>;
SectOne: PRT => SectOneTitle SectOneCont; ...
SectOneCont: PRT => MainText | {SectTwo};
MainText: PRT => {(ParaSta [{(Experience | NonText| Table
                  | ItemFST | ItemFol) [ParaSpe]}])
                  | (NonText ItemFol) | TopicTitle};
ParaSta: FRG => ParaData ;
   ParaSta.separBefore = (Allowed, [3pt, 20pt]);
ParaData: STR => {Word | CombWord | Number | CombNumber
                  | LevelOneNum | LevelTwoNum
                  | LevelThreeNum | Punction | KeyExpr
                  | KeySent | MathOpe | MixWord | Phrase}; ...
```

Figure 3. A partial document description of the Chapter 6 of a scientific book

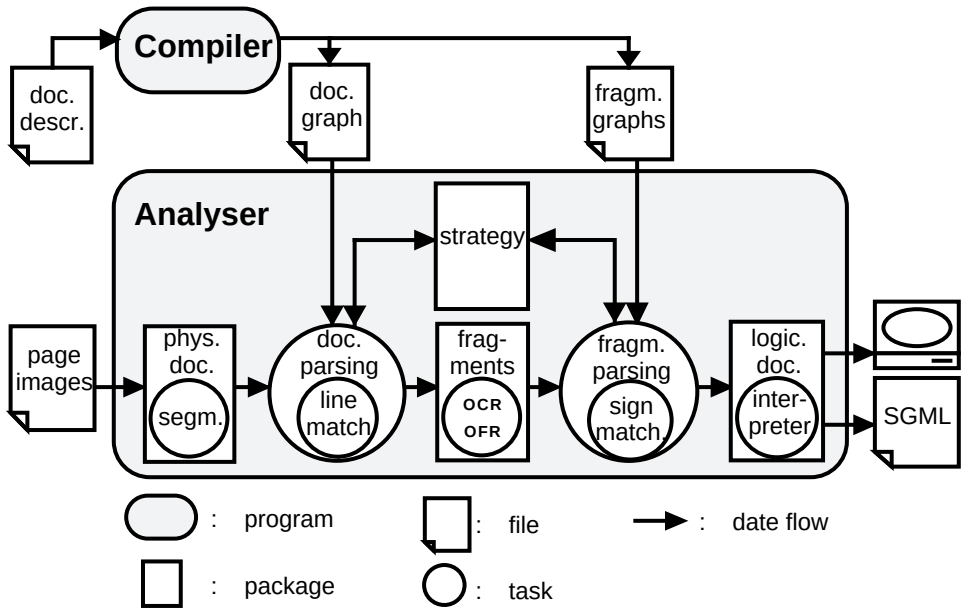

Figure 4. Simplified system architecture of the prototype

input element and a candidate pattern in a very fine way; however, it also increases the number of kept candidates at each searching stage and makes the problem of the analysis efficiency critical.

In Section 2, a brief description of the prototype is presented, and some terms used in this article are given. In Section 3 and 4, the main efforts of improving analysis efficiency are discussed in detail. The analysis process includes two main parts: *parsing* and *analysis strategy*. The former finds a path within an analysis graph; and the latter controls the whole analysis process. In Section 5, some experimental results based on a real example are introduced. In the final Section, the special techniques used in the prototype are summarised, and the propositions of future developments are presented.

2 SYSTEM OVERVIEW

2.1 Basic system architecture

A simplified architecture of our prototype is represented in Figure 4. The input of the system includes *a set of page images* for a document and its *document description*. The output is the logical structure of the document, which can be displayed or saved in a file with *SGML* format [1].

The system consists of two main programs: the *compiler* [15], which compiles a document description into one document graph and several fragment graphs; and the *analyser*, which turns the page images of a document into their structural representations based on both kinds of graphs presented above.

2.2 Top-down analysis with two parsing levels

Based on a document description, both *top-down* and *bottom-up* analysis can be performed. However, a *top-down analysis* is often more efficient since the searching is guided by an analysis graph [16]. Our top-down analysis has further been split up into two levels: *document parsing* and *fragment parsing*. The former intends to generate the macro structure of a document, and the latter generates its micro structure.

In order to support both parsing levels, the compiler turns a document description into one *document graph* and a set of *fragment graphs*. Because of the structural similarity of both kinds of graphs, we use the generic term of *analysis graph*.

2.3 Some basic concepts

Matching checks the similarity between the extracted typographic attributes of a physical element and those of a node in an analysis graph. Matching operates on both levels: between *line* and *line node* for a document graph as well as between *sign* and *character node* for a fragment graph.

Parsing is achieved by finding a *path* throughout an analysis graph for an input list; therefore, each *path step* is in relation with a node in the graph and an element in the input list. For each path step, the predefined attributes of a node should match the extracted attributes of its corresponding physical entity.

Based on fuzzy logic, the result of matching takes the form of a *matching degree* (MD), from 0% to 100%; and its complement, 1 - MD, is currently used as *node cost*. The value of the *path cost* is calculated by cumulating all the node costs on a path.

2.4 The main parts of the analyser program

The *analyser program* can be further split up into several packages and tasks:

- the high level of a *physical document tree* is built by *segmenting* page images into regions and lines [17];
- based on a *document graph*, the *document parsing* intends to find all possible paths according to the results of *line matching*; then, these paths will be further checked on fragment level; according to one of these paths, one way of logical labelling for each line can be produced;
- *fragments* are constructed by grouping some connected lines, which share the same logical label; and the information about each sign of the fragment is obtained by the tasks of *optical character recognition* (OCR) and *optical font recognition* (OFR) [18];
- based on *fragment graphs*, the *fragment parsing* tries to assign a logical label for each sign according to the results of *sign matching*; if the *optimal path* is found based on the results of both parsing tasks, a *logical structure tree* will be built, otherwise, the *document parsing* will be activated again to try another possible path;
- the two parsing tasks are controlled by the *strategy* package, which provides several strategies for analysis;
- finally, the *logical structure tree* is displayed or interpreted into its *SGML* representation by the *interpreter* task.

3 PARSING AND DYNAMIC THRESHOLDS

The goal of parsing is to find the minimum cost path through an analysis graph for a list of physical elements. Based on a *directed graph*, many searching algorithms could be used [13]. However, our analysis graph has its special features; therefore, the main problems are: how to use these algorithms, which one is more efficient than others, and how to combine or modify them to serve our purpose.

The overview of some major searching algorithms is given in Subsection 3.1; the special features of an analysis graph are discussed in Subsection 3.2; in Subsection 3.3 and 3.4, we present two parsing algorithms developed for the prototype: one based on depth-first searching and the other built on dynamic programming searching. We prefer the latter; however, the former can be used for comparison. In the final Subsection, we illustrate the effects of error-tolerating parsers on system efficiency through a comparison with error-correcting parsers.

3.1 Major searching (parsing) algorithms

The search space of most searching algorithms can be represented by an acyclic directed graph [13,19], which can also be modelled by a tree, at the cost of introducing some duplicate nodes.

Many conventional searching algorithms are based on a tree structure, such as, *depth-first search* and *breadth-first search*. Without any specific knowledge, the two exhaustive searching algorithms are often used; but the time or space requirement increases exponentially.

Heuristic algorithms, for instance, *best-first search*, tend to use some heuristic knowledge, which is often in the form of a heuristic function, to prune tree branches and to control the searching order among alternative candidates at each stage [13]. These algorithms are suitable when the measured values of some candidates are distinguishable. However, in the worst case, these algorithms have still exponential expenses.

Dynamic programming [20] overcomes the above shortcomings by using *structural dominance* to prune branches; in other words, among the several sub-paths between two path steps only the minimum cost sub-path is kept. This algorithm is based on a *layer network structure*, where the layer number is equal to the size of the input list and the number of searching stages; and there are no duplicate nodes at each layer. This algorithm will be explained in more detail in Subsection 3.4, and it has been used successfully in some fields like signal processing and string matching [11].

Using a method based on the principle of *sub-goals* [13], a large problem can be separated into several independent sub-problems. Therefore, both the search space and the length of the input list can be greatly reduced.

3.2 The special features of analysis graphs

Figure 5 shows an example of an analysis graph, where each node corresponds to a physical element (line or sign). The horizontal arrow of a node points to its *successor* and the vertical arrow points to its *alternative*. The successor will be visited when the node is matched with the input element; otherwise, the alternative will be visited. There are only one *starting node* and one *ending node* in an analysis graph. The according *typographic and semantic*

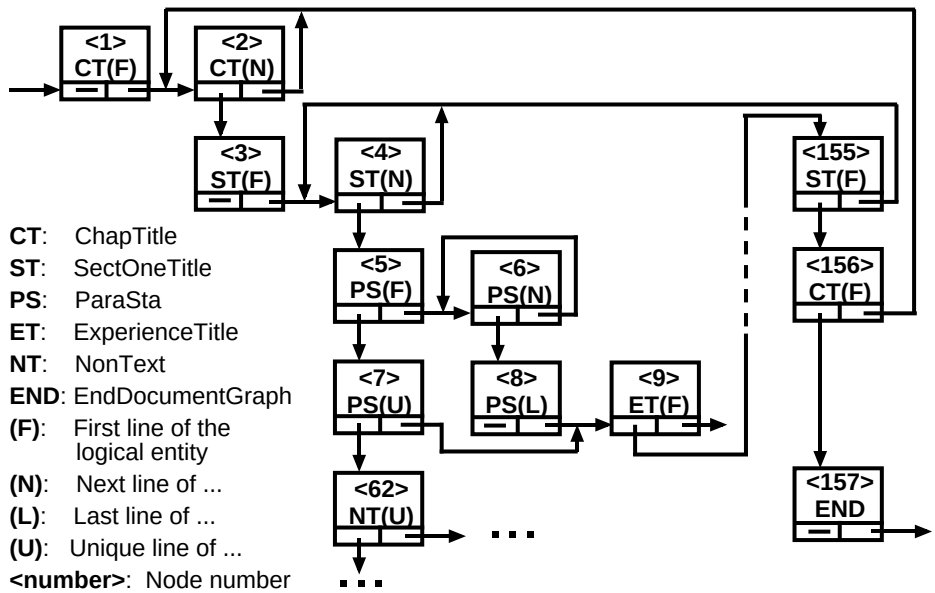

Figure 5. A partial document graph based on the above document description

attributes are attached to each node except for the ending node. *Cycle* is allowed in a graph, such as, node 2 and node 4; therefore, many searching algorithms cannot be used directly.

3.3 An improved depth-first parsing algorithm

Based on an analysis graph, a *depth-first parsing* can be implemented easily. A parser begins from the starting node of the graph and tries to match the first element of an input list. If the match is successful, the current pointer of the parser points to the successor of the current node, otherwise, to the alternative of the current node, and so on recursively.

Several techniques have been used to improve the algorithm. Heuristic thresholds, such as, node cost and path cost, are used to cut less possible sub-paths resulting in the improvement of the parsing efficiency. Multiple pass parsing together with a regular adjustment on these thresholds at each pass is utilised; and the problem of finding no path or many paths can be reduced. However, this algorithm is not efficient when many paths have similar path cost; and it is also hard to deal with small local errors.

Furthermore, in order to prevent an endless searching, some heuristic termination conditions are applied, such as, when a low cost path is found or when more than a fixed number of possible paths are saved; however, there exists the risk of finding only a sub-optimal path.

In order to deal with these problems, a more robust and efficient searching algorithm is needed; and the algorithm of *dynamic programming* is one of the possible solutions.

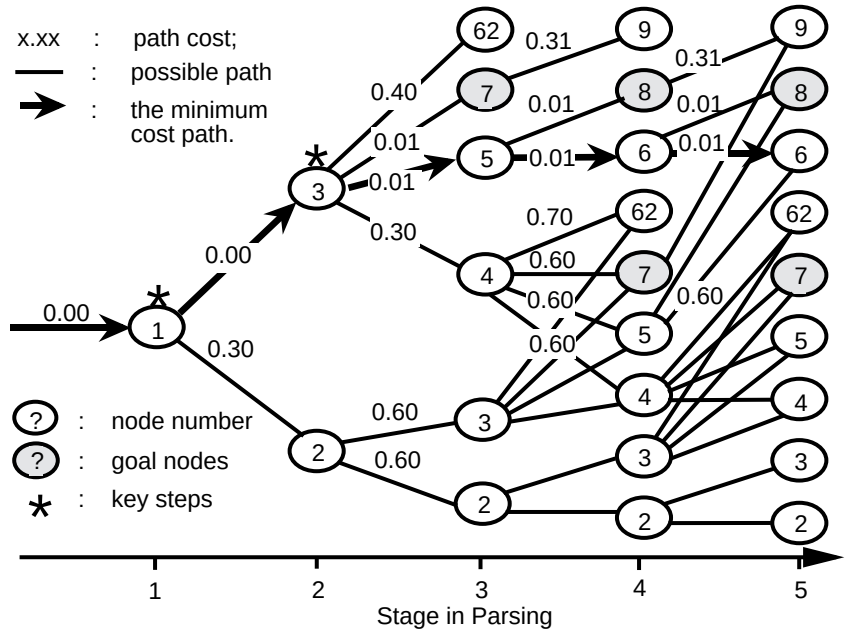

Figure 6. A layer network view of first five stages of parsing the above document graph

3.4 A forward dynamic programming parsing algorithm with heuristics and dynamic thresholds

Based on an analysis graph, a layer network can be built dynamically. Figure 6 is a layer network based on the analysis graph of Figure 5. The starting node of the graph and its alternatives (the transition closure of its alternative) form the nodes at stage one; then the successors of these nodes at stage one, including the transitive closure of their alternatives constitute the nodes at stage two, this procedure continues until the stage number equals the length of an input list.

The principle of optimality of a forward dynamic programming algorithm is stated in [20]. Using this algorithm, a forward computation moves from starting node A; this procedure yields the optimal path from A to every nodes until a goal node is reached. In an analysis graph, a node is called a *goal node* if its successor or one of the alternatives of the successor is the ending node of the graph.

The heuristic thresholds used for the *improved depth-first parsing algorithm* are also applied in this mixed parsing algorithm, but in a dynamic way. *Dynamic programming* is, in fact, a special case of *breadth-first searching*, where all candidate nodes at the same stage are compared with the current input element; therefore, a decision about the kept candidates can be made according to some thresholds, which can be adjusted dynamically according to *the minimum node cost* or *minimum path cost* at each stage. Using the technique of *dynamic threshold*, some local errors can be tolerated easily; and the heuristic pruning can be done in a more suitable way.

Combining all these techniques, the mixed parsing algorithm is much more efficient and robust than the depth-first one. It is, in fact, a linear function of the length of the input

list, with a constant factor, which is, at worst case, the square of the size of the analysis graph.

3.5 Error-correcting parsing vs error-tolerating parsing

Error-correcting parsing is often related to string matching [11]. This parsing intends to find the closest path between a given string model and an input string by calculating the cost of applied edition operators, namely, insertion, deletion, and substitution.

In an analysis graph, there are often too many possible models for a given input string; therefore, it is impractical to apply the error-correcting technique to check each model. However, an *error-tolerating parsing* can play a similar function in a more efficient way.

Our parsing algorithms are based on fuzzy logic and thresholds, which make an error-tolerating parsing possible. This parsing intends to tolerate some local errors between the current model and the input string by using thresholds, such as, node cost and path cost. If we keep in mind that these models are complementary, one model can be thought as the result of applying some editing operations on another model, and a node cost is similar to the cost of an editing operation; then, the error-tolerating parsing can simulate the error-correcting parsing well, but in a more efficient way, because a lot of duplicated calculations are avoided.

4 ANALYSIS STRATEGIES AND KEY STEPS

4.1 Analysis strategies based on two levels of parsing

Based on two levels of parsing, the general analysis order should be top-down. The determination of a fragment bound and the choice of a fragment graph for fragment parsing depend on the results of document parsing, so document parsing should be made before fragment parsing. However, a fine controlled *analysis strategy* has also a great influence on the system efficiency. At least, two kinds of strategies are possible, namely: *pure top-down* and *mixed top-down*.

With a *pure top-down strategy*, the *document parsing* keeps all possible paths for a given input line list, then these paths are checked down to character level one by one. The problem with this strategy is that many paths can have similar path costs at *document parsing* level; therefore, the price for fragment parsing will be very high. Using strict threshold may reduce the number of possible paths; however, it increases also the risk of missing *the final optimal path*, which should be based on the results of both parsing levels.

Utilising the *mixed top-down strategy*, a partial parsing at document level is done first; then the analysis goes down to the fragment level when a local ambiguity occurs. The fragment parsing may reduce the local ambiguity and the analysis goes back to document level again. The change between these two levels may be repeated several times until the parsing at both levels is finished.

4.2 Key steps

In *document parsing*, often more than one possible path exists; however, these ambiguities are often local. Therefore, we can find those common path steps and use them to separate a whole path into several independent sub-paths; then both the size of a document graph and

the length of an input list can be reduced considerably, and the combinatorial explosion on path number caused by local ambiguity can be avoided.

Based on the above consideration, we have defined the concept of *key step*. A *path step* is related to a searching stage of the *forward dynamic programming parsing*. If at one stage, there exists a graph node, such that the minimum cost sub-path ending with this node has much lower path cost than other possible sub-paths at the same stage, then the path step is called a key step. A key step is always related to a definite node at an analysis graph, an element in an input list and a searching stage of the forward dynamic programming parsing. Normally, the final optimal path should go through those graph nodes, which are relevant to all key steps of the parsing; therefore, these key steps can be used to split up a long path, a big analysis graph and a long input list. To separate an analysis graph based on two key steps, the two relevant graph nodes should be used, which will be the starting node and the only goal node of the new defined sub-graph. Still the fragment parsing on these key steps needs to be done only once so that the computing complexity of the whole analysis can be reduced greatly.

By using the concept of key steps, some errors or ambiguities can be easily localised; therefore, it is also very helpful for error report executed by the system or for an interactive change accomplished by users.

4.3 Our proposed strategy

The task of the analyser program is to find *the final optimal path* based on both parsing levels. A minimum cost path at document level does not imply that this path is still optimal after fragment parsing; therefore, the document parsing should provide more possible paths for fragment parsing.

The *mixed dynamic programming parsing* is guaranteed to find the minimum cost path(s) based on an analysis graph; however, at *document parsing*, the strict dominating operations may eliminate some sub-paths, which are parts of the final optimal path after *fragment parsing*. Therefore, we propose the following multiple pass strategy to deal with some complex cases:

- in the first pass, the *mixed dynamic programming parsing* is used to find all key steps at document level, then to separate the document graph and the input list;
- using the mixed parsing again to analyse these sub-lists based on new sub-graphs, a *mixed top-down analysis strategy* is applied so that the fragment level check can be done before a dominating operation at document level intends to eliminate a sub-path, which is only slightly different in path cost compared to the one kept;
- finally, these separated optimal sub-paths are obtained, and they are connected to form the final optimal path.

5 EXPERIMENT AND RESULTS

The prototype system has been implemented in Ada and C on Sun Sparc stations. Due to object-oriented programming techniques, the several programs or modules of the system are independent; still, each module consists of several independent packages and tasks. Therefore, expansions and modifications of the system can be introduced easily.

A complex example, the chapter 6 of a scientific book, has been used to test the *robustness* and *efficiency* of the system. This chapter includes 22 pages, Figure 1 is its first page. The logical structure of the chapter is very complex: more than 50 different logical entity types have been defined; but the typographic attributes of some of them are similar. Figure 3 presents a partial document description of the chapter. Even with the simplified document description the generated document graph contains 157 nodes, and the generated fragment graph for the fragment called *ExpeParaSta* has 89 nodes. Different non-text elements (figures, formulas, and tables) are mixed in these pages, which increases the ambiguity. Still the basic recognition processes (segmentation, OCR, and font identification) are affected by skewed image, input noise, etc; and the document description does not take into account the existence of some exceptions, such as some undefined chemical symbols.

Different parsing algorithms and analysis strategies have been applied to this example, and the experiment has shown that the above proposed methods are efficient. For instance, the first paragraph of the experience has 247 signs. The fragment parsing needs the graph of *ExpeParaSta*. Using the *improved depth-first parsing*, the parsing stopped by storage overflow after having searched more than 90,000 nodes. However, with the *mixed dynamic programming parsing*, the minimum cost path was found after searching 19,355 nodes. Without heuristics, the parsing would have needed more than 400,000 nodes. Figure 6 shows the layer network view of the first five stages of parsing on the document graph of our example. These fine lines are the possible paths at each stage, and the thick arrows show the *minimum cost path*. The path steps at stage one and stage two are obvious *key steps*: at stage one, there is only one path; and at stage two, the path through the node 3 has a much lower path cost than the path through the node 2. The obtained logical structure tree of the first page is shown in [9].

6 CONCLUSIONS

In this article, special efforts toward an efficient document structure analysis have been presented. Thanks to the use of fuzzy logic in matching, error-tolerance on parsing can be performed. All parsing algorithms and analysis strategies are based on two levels of analysis graphs; however, our mixed approach has integrated both levels. The mixed forward dynamic programming parsing, based on a dynamically built layer network, together with the heuristics and dynamic threshold techniques reduce the computing complexity into linearity. Introducing the concept of key step, an application of the principle of sub-goals, a multi-pass and mixed top-down analysis strategy, has been proposed, which avoids the combinatorial explosion of path number and reduces computing expenses. Based on an analysis graph, the conventional error-correcting parsing becomes impractical; however, error-tolerating parsing can be implemented easily and it can simulate the error-correcting functions in a more efficient way.

The task of improving the robustness and efficiency on an AI system has been one of central problems for decades [11]. However, its application-dependent features make the problem still critical for many practical systems. Basic work towards these goals have been done on the prototype; however, some refinements should be done, for instance, through testing more document models to adjust the parameters used in the system. Furthermore, a more flexible system control mechanism should be considered.

REFERENCES

1. *International Standard ISO/IEC 8879: Information Processing — Text and Office Systems — Standard Generalized Markup Language (SGML)*, ed., International Organization for Standardization, Geneva/New York, first edition, 1986.
2. *International Standard ISO/IEC 8613: Information Processing — Text and Office Systems — Office Document Architecture (ODA) and Interchange Format*, ed., International Organization for Standardization, Geneva/New York, 1989.
3. L.D. Wilcox and A.L. Spitz, 'Automatic recognition and representation of documents', in *Document Manipulation and Typography: Proceedings of the International Conference on Electronic Publishing, Document Manipulation and Typography (EP88)*, ed., J.C. van Vliet, 47–57, Cambridge University Press, Cambridge, UK, (1988).
4. S. Tsujimoto and H. Asada, 'Understanding multi-articled documents', in *Proceedings 10th International Conference on Pattern Recognition*, 551–556, IEEE, New York, NY, (1990).
5. G. Nagy, S. Seth, and M. Viswanathan, 'A prototype document image analysis system for technical journals', *IEEE Computer Magazine*, 10–22, (1992).
6. Y. Chenevoy, *Reconnaissance structurelle de documents imprimés: études et réalisations*, Ph.D. dissertation, CRIN-University of Nancy, France, December 1992. (Thèse de Doctorat de l'INPL.)
7. T.A. Bayer, 'Understanding structured text documents by a model-based document analysis system', in *Proceedings of the Second International Conference on Document Analysis and Recognition*, 448–453, IEEE Computer Society Press, Los Alamitos, CA, (1993).
8. J. Lii, P.W. Palumbo, and S.N. Srihari, 'Address block location using character recognition and address syntax', in *Proceedings of the Second International Conference on Document Analysis and Recognition*, 330–335, IEEE Computer Society Press, Los Alamitos, CA, (1993).
9. T. Hu and R. Ingold, 'A prototype for document structure analysis using fuzzy logic', in *Proceedings of Seventh International Conference on Image Analysis and Processing*, (1993).
10. L.A. Zadeh, 'Fuzzy sets', *Information and Control*, **8**, 338–353, (1965).
11. H. Bunke and A. Sanfeliu, *Syntactic and Structural Pattern Recognition Theory and Applications*, World Scientific, 1990.
12. A. Dengel, 'A step towards understanding paper documents', Technical report, The German Research Centre for Artificial Intelligence, (1990).
13. R.E. Korf, 'Optimal path-finding algorithms', in *Search in Artificial Intelligence*, eds., L. Kanal and V. Kumar, 223–267, Springer, Heidelberg/Berlin/New York, (1988).
14. R. Ingold, 'A document description language to drive document analysis', in *Proceedings of the First International Conference on Document Analysis and Recognition*, 294–301, (1991).
15. P. Joye, 'Compilation d'un langage de description de documents destinés a l'analyse structurelle', Technical report, LIUF, University of Fribourg, Switzerland, (1993).
16. R. Ingold and D. Armangil, 'A top-down document analysis method for logical structure recognition', in *Proceedings of the First International Conference on Document Analysis and Recognition*, 41–49, (1991).
17. A. Azokly, A. Zramdini, and R. Ingold, 'Reconnaissance de la structure physique de documents composites', *BIGRE 80: Actes de CNED'92: Traitement de l'écriture et des documents*, 30–39, (1992).
18. A. Zramdini, A. Azokly, and R. Ingold, 'Importance de l'identification de la fonte dans la reconnaissance structurelle de documents', *BIGRE 80: Actes de CNED'92: Traitement de l'écriture et des documents*, 233–241, (1992).
19. R.E. Korf, 'A survey of recent results', in *Exploring Artificial Intelligence: Survey Talks from the National Conferences on Artificial Intelligence*, 197–237, Morgan Kaufmann, Los Altos, CA, (1988).
20. S.E. Greyfus and A.M. Law, *The Art and Theory of Dynamic Programming*, Academic Press, Orlando, FL, 1977.