Near-Wordless Document Structure Classification

Kristen Summers
Department of Computer Science
Cornell University
`summers@cs.cornell.edu`

Abstract

Automatic derivation of logical document structure from generic layout would enable a multiplicity of electronic document manipulation tools of a type that is becoming crucial to users who wish to browse the internet. This problem can be divided into segmentation (dividing the text into a hierarchy of pieces) and classification (categorizing these pieces as particular logical structures.) This paper proposes an approach to the classification of logical document structures, according to their distance from prototypes that are primarily geometric. The prototypes consider linguistic information minimally, thus relying minimally on the accuracy of OCR and decreasing language-dependence. Different classes of logical structures and the differences in the requisite information for classifying them are presented. A prototype format is proposed, existing prototypes and a distance measurement are described, and performance results are provided.

1 Introduction

The availability of large, heterogeneous repositories of electronic documents is increasing rapidly. As good internet navigation and resource discovery tools are developed, the text files accessible to a user through the internet will come to form an implicit distributed collection of documents in much the same way as other files will form an implicit database collection [19]. As the quantity of available information grows, however, so too does the probability of overwhelming a user; in order to use the internet document collection effectively, users will require sophisticated tools for manipulating its elements.

Electronic document manipulation tools can benefit greatly from exploiting the existence of various types of document structure. A document possesses at least the following types of structure: the *physical* or *layout* structure, consisting of the document's division into pages and the marks on these pages and their positions; the *bibliographic* structure, consisting of the title, author, publication information, etc.; the *logical* structure, consisting of the (visually) observably separate components that make up the document, such as paragraphs, sections, lists, etc.; and the *content* structure, consisting of content-oriented elements, such as passages in a text or references to a character in a novel. Each of these types of structure can be represented as a hierarchy.

The automatic discovery of logical document structure would enable a multiplicity of electronic document tools, including (1) automated markup, (2) structural hyperlinking, (3) hierarchical browsing, and (4) logical component-based retrieval. Markup, in SGML or another format, can proceed based on logical structure directly; hyperlinking can be automated by defining links to exist between certain kinds of structures under given sets of conditions; browsing can proceed by exploring the logical structure tree; and retrieval can be based in whole or in part on the existence of and relationships between logical structures. (This last can, of course, be combined with classical information retrieval techniques [6, 8, 16].)

Documents have an *inherent* layout structure, i.e. the character- and line-level marks on a page and their positions can be automatically described (in shape, if not meaning) precisely, and they consistently have a significance, by virtue of the fact that document layout is intended to convey meaning to human readers. In contrast, unless a logical structure is *imposed* by means of some kind of explicit markup, it must be inferred from other cues. This paper is concerned with inferring logical structure in *generic* documents, i.e., those for which the formatting methods used are unknown.

The problem of deriving the logical structure of a document from its inherent layout structure (a form of representation that can be obtained from an OCR system or a rendering of a document in a page definition language, hence a form that includes many existing electronic documents) can be divided into two stages: (1) *segmentation* of the document into a hierarchy of logical elements and (2) *classification* of the nodes of the hierarchy according to the type of logical structures they represent. Our earlier work [15] has proposed an approach to segmentation based entirely on an analysis of the contours of text (i.e., the left and right white space); this paper proposes an approach to classification that relies minimally on symbolic and linguistic cues.

By its primary use of geometric cues in this classification, the approach described here comes close to language-independence; the reliance on language is sufficiently slight and well-identified to make the adjustment to a new language a reasonably easy task. A stronger reliance on geometry than characters is, furthermore, suitable to a system designed to use the results of OCR, since OCR cannot yet provide assurance of every character.

The proposed approach is to define a set of prototypes, each of which describes a structure to be recognized by the system. A distance measurement is defined, and classification proceeds by finding the prototype to which a document segment is nearest. The segment is then assigned the category identified with the prototype. A few categories must be identified after this main step, as they rely entirely on their relationships with other document elements; this is addressed in Subsection 3.3.

1.1 Outline

The remainder of this section describes previous related work. Section 2 defines different classes of logical structures and specifies their relationships to the work presented here. Section 3 describes the prototype attributes currently allowed and their contributions to the distance metric. Section 4 describes the implementation and performance of this approach, and Section 5 provides a discussion of the work and future directions.

1.2 Previous Work

Most previous work on the automatic discovery of logical structure in electronic documents has been concerned with the special case in which the formatting rules for a document are known. More precisely, the underlying assumption is that all documents to be considered by a given system represent the logical structures of interest by the same identifiable layout phenomena. As a result of this assumption, segmentation and classification are performed in the single step of finding the requisite layout structures.

The systems in [9], [12], [14], and [18] identify structures based on a flat set of divisions of pages of one narrow format type each.[1] The system in [3] learns to identify lexically similar structures;

[1]These narrow formats include such types as envelopes and tables, which are typically quite highly and uniformly

again, these form a flat set of divisions. The system in [4, 5] is more general, separating pages into text and graphics. These approaches are appropriate when the structures of interest include no nesting; all but [4, 5] and [3] also require that the documents of interest be of a known, highly structured type.

More general systems are presented in [11], [7, 20], and [13]. The former two systems find logical document hierarchies consisting of paragraphs, sections, and subsections by parsing, based on knowledge of standard printers' formatting rules. The latter system finds hierarchies that include intermediate structures by tree manipulation, based on knowledge of the rules for the style of a particular document.[2] This is clearly the approach of choice when the necessary information is available; if the formatting rules used to represent structures of interest are known and have not been violated, then using these to parse the document is the logical way to find the corresponding structures.

Unfortunately, not all documents conform to a well-defined set of rules in their layout representation of logical structure.[3] A user who finds a document on the internet has no guarantee about the approach used by the document creator, and the greater the number and complexity of relevant structures, the less predictable are the forms in which they will appear. The approach presented here does not rely on following any precise rules; thus, it will perform reasonably on novel documents. It relies instead, as mentioned, on the fact that documents are arranged to convey information and thus will not vary too extremely in the presentation of their logical components.[4]

2 Structure Classes

Within the logical structure of a document, there exist different element types. These types require different kinds of processing in order to accurately identify them.

The main distinction is drawn between *primary* structures, which are defined by their own contents and (possibly context-dependent) presentation and *secondary* structures, which can be fully defined by their relationships to other structures. For instance, a *paragraph* is a primary structure, whereas a *paragraph group* structure is secondary. The prototypes presented here are used for the classification of primary structures; secondary structures are then identified by their relationships to primary structures.

Primary structures include both *linguistic* structures, the definitions of which rely upon the words contained in the document segment, and *non-linguistic* structures, which are definable by geometric and symbolic (non-alphabetic) attributes. For instance, *theorem* is a linguistic structure; distinguishing a theorem from a definition (or even a quotation, in some cases) involves performing an analysis of its words and content. The structure *special paragraph*, defined as a paragraph that differs in format from the surrounding norm is, however, non-linguistic; in many stylistic conventions, theorems are a subset of special paragraphs.[5] Non-linguistic structures may require symbols or they may be purely *geometric* structures, requiring only shape-based information.

This paper addresses primarily the categorization of non-linguistic structures; an extremely shallow element of word-based processing is included in order to find (without absolute certainty)

structured, in a fashion that requires no hierarchy.

[2]For example, a document may be known to have been formatted with the LaTeX `article` style; in this case, precisely those structures for which the author has used the provided macros will be found.

[3]In fact, [1] suggests that people tend to violate rules of this kind, even when they write the rules themselves.

[4]One effect of this is that the system would probably not perform well on a text like [2]; documents designed to violate human expectations fall outside the scope of this work.

[5]For instance, in the LaTeX `article` style [10], theorems and similar structures are entirely left-justified, but ordinary paragraphs have an indented first line.

Table 1: Characterizations of Current Structures

Structure	Primary	Secondary	Linguistic	Symbolic	Geometric
Paragraph	x				x
Heading	x			x	x
Section Body		x			
Section		x			
First Paragraph	x				x
List Item	x				x
List	x				x
Title Start	x				x
Title Part	x				x
Equation	x			x	x
Paragraph Part		x			
Paragraph Group		x			
Theorem	x		x		x
Definition	x		x		x
Proof	x		x		x

theorems, definitions, etc. More complete linguistic processing may be added as a final step, with whatever degree of depth and sophistication is appropriate to the desired application.[6] In many cases, the very simple linguistic analysis presented here suffices to find logical structure (as opposed to content structure). The structures found by the current system and their characterizations are given in Table 1.

3 Logical Structure Prototypes

The prototypes proposed here for logical structures include the following five attributes, any of which may remain unspecified; all segments match an unspecified attribute.

- `Geometry` The `geometry` of a document segment is a representation of its text contours in our indentation alphabet, described in Subsection 3.1.1.
- `Context` The `context` of a text block consists of a set of descriptive features of the blocks that precede and succeed it, specified in Subsection 3.1.2.
- `Height` The `height` specifies the typical length of the structure, in lines.
- `Symbols` The `symbols` attribute consists of a description of typically appearing or non-appearing symbols, as given in Subsection 3.1.3.
- `Children` The `children` attribute describes typical or atypical children of the node in the hierarchy and is defined in Subsection 3.1.4.

[6]For example, finding the author name and affiliation within the title part of a paper would require more linguistic analysis than the prototypes advocated here allow.

The distance measurement is described in Subsection 3.2. The details of the resulting algorithm are supplied in Subsection 3.3. Sample existing prototypes are given in Figure 1

3.1 Prototype Attributes

3.1.1 The Indentation Alphabet

Earlier work has presented the *indentation alphabet* data structure for capturing the geometry of document segments [15]. An indentation alphabet provides a means of describing the text contours of a document portion. In essence, it consists of a set of symbols and their definitions, together with a set of matching rules. Each symbol corresponds to a type of indentation, and a text block is represented by the symbols that describe its sub-blocks. (A fully blank line is considered a type of indentation.) The sub-blocks of a node are the divisions that make up its children in the segmentation tree.

Our segmentation algorithm relies heavily on indentation alphabets, so they form a natural characterization of document elements for use in the classification stage. Specifically, the indentation alphabet representation of a node provides a description of its internal geometry, at the level of granularity appropriate to its level in the tree. The indentation alphabet used for this characterization, a slightly less detailed version of that used for segmentation, distinguishes the following contour types (describing text only, not blank lines):

Both Justified The sub-block's left and right edges are the same as those which precede it.

Left Justified The sub-block's left edge is the same as that which precedes it.

Centered-In The sub-block is centered with respect to what precedes it, and its left edge lies to the right of the left edge of its predecessor.

Centered-Out The sub-block is centered with respect to what precedes it, and its left edge lies to the left of the left edge of its predecessor.

Centered The sub-block is centered with respect to what precedes it.

Left In The sub-block's left edge lies to the right of the left edge of its predecessor.

Left Out The sub-block's left edge lies to the left of the left edge of its predecessor.

When a sub-block meets the definition of more than one indentation alphabet symbol, it is characterized by the most specific choice; a symbol is always allowed to match a more general symbol. For example, a line that can be described as Centered-In can also be described as Centered or, alternatively, as Left-In. Either may be correct; the centering may be intentional, or it may be an accident of the length of the line. The line is characterized by Centered-In, the most specific of these; it may match either of the others in comparing this block to a prototype.

3.1.2 The Context Attribute

The `context` attribute enables a small degree of context-dependence in the classification algorithm; it provides a description of a prototypical predecessor and successor, including the following elements:

Paragraph

`Geometry:` Left In Left Out

`Context:` **Pred.:** `Blank:` yes
Succ.: `Blank:` yes

First Paragraph

`Geometry:` Left In Left Out

`Context:` **Pred.:** `Blank:` yes, `Same:` no, *`Category:` *heading*
Succ.: `Blank:` yes, `Same:` no

Heading

`Context:` **Pred.:** `Same:` no, `Blank:` yes
Succ.: `Same:` no, `Blank:` yes*

`Height:` 1

`Symbols:` **Positive:** $\langle$ {1, ... 9}, `start` $\rangle$
Negative: $\langle$ {., !, ?}, `start` $\rangle$

Theorem

`Geometry:` Left In Left Out

`Context:` **Pred.:** `Blank:` yes, `Same:` no
Succ.: `Blank:` yes, `Same:` no

`Symbols:` **Positive:** $\langle$ {theorem, lemma} `start` $\rangle$

List Item

`Geometry:` Left In Left In

`Context:` **Pred.:** `Same:` yes, `Blank:` yes, `Category:` *List Item*
Succ.: `Same:` yes, `Blank:` yes

Figure 1: Sample prototypes. Attributes with a * immediately preceding their name and specification are marked as necessary. Note that the *paragraph* prototype includes both the shape cue for indented paragraphs and the cue of surrounding white space, often used without indenting. This kind of paragraph will be assigned to the *paragraph* category, as well as an indented paragraph. *First paragraph* and *theorem* are both types of *special paragraph*, a paragraph that differs from its surrounding context.

- `Geometry` Again, this is the representation of a text block in the indentation alphabet described in Subsection 3.1.1.
- `Sameness` This indicates whether the `geometry` of the predecessor/successor is prototypically the same as that of the segment in question. For example, a *list item* is typically geometrically similar to both its predecessor and successor; a *special paragraph* is not.
- `Blank` This indicates whether a blank space (greater than the interline spacing of the page) exists between the text under consideration and its predecessor/successor.
- `Category` This applies only to the predecessor of a segment, and it describes the classification that would prototypically be assigned to it. For example, the `category` of the predecessor of a prototype *proof* is *theorem*, where *theorem* includes theorems and lemmas.

3.1.3 The Symbols Attribute

The `symbols` attribute includes both the use of non-linguistic symbols and the small amount of word-based processing included in this approach; it consists of a list of positive and/or negative expectations. (Positive expectations typically appear; negative expectations typically do not.) Each of these specifications includes a list of symbols (or words) and a position indicator, which may be one of: `start`, `end`, `any`. A positive (negative) expectation with a position of `start` indicates that one of its symbols should (should not) start the structure; `end` refers to ending the structure, and `any` indicates an expectation about whether the symbol(s) will appear anywhere in the text segment.

For example, headings typically do not end with a punctuation mark that may end a sentence;[7] hence, the *heading* prototype includes a negative expectation of ⟨ {., !, ?}, `end` ⟩.

3.1.4 The Children Attribute

The `children` attribute describes the typical or atypical categories of the hierarchical children of a given structure. It consists of a list of positive and negative expectations, each of which is a list of children. A positive expectation indicates that the structure should have a child that is a member of its category list; a negative expectation indicates that the structure should not.

For instance, a *list*'s children are typically list items or sub-lists; moreover, any other categories of children are atypical and contribute to a structure's distance from the *list* prototype.[8] Hence, the `children` attribute of the *list* prototype includes the positive expectation { *list item*, *list* } and the negative expectation Σ − { *list item*, *list* }, where Σ is the set of all possible structure categories.

3.2 The Distance Measure

The measurement of the distance from a segment to a prototype is given on a scale from 0 to 1 by an approximate average of the following attribute distances (each of which is normalized to fall within the interval $[0,1]$.) An unspecified attribute always yields distance 0. Attributes may also occasionally be marked as necessary; a non-zero distance from a necessary attribute automatically generates a distance of 1 from the prototype.

[7]Sometimes they do; this indicates the value of using prototypes, rather than strict definitions

[8]A *list* is not, however, considered a secondary structure; a structure with many children, of which one is a *paragraph* and the rest *list items* is probably a list that contains an interruption.

`Geometry` The gross geometric distance between a text block and a structure prototype is given by the Levenshtein distance [17] between the indentation alphabet representations; this is normalized by dividing by the sum of the lengths of the two representations.

`Context` The context distance is the average of its the predecessor and successor distances; each of these is the average of the sub-attribute distances.

`Geometry` This distance is precisely as described above.

`Sameness` This distance is 0 if the sameness expectation is satisfied; otherwise it is 1.

`Blank` This distance is 0 if the expectation is satisfied; otherwise it is 1.

`Category` This distance is 0 if the expectation is satisfied; otherwise it is 1.

`Height` The gross height distance is given by the difference in height between the prototype and the text segment; this is normalized by dividing by the maximum of the two heights. Thus, a difference of one or two lines, while quite significant in a structure expected to be short, will be relatively insignificant in a structure expected to be large.

`Symbols` The gross symbol distance is given by the number of unmet symbol expectations, i.e., positive expectations for which none of the symbols is found in the text block at an appropriate position and negative expectations for which one of the symbols is found at the specified position. This distance is normalized by dividing by the total number of expectations.

`Children` The children distance is the average of the distance due to positive expectations and that due to negative expectations, which are determined slightly differently. The gross positive expectation distance is given by the number of *unmet* positive expectations, i.e., the number of positive expectations for which none of the children match any of the specified categories. This distance is normalized by dividing by the number of positive expectations. The gross negative expectation distance is given by the number of *violations* of the negative expectations, i.e., the number of children that belong to a category given in a negative expectation. This distance is normalized by dividing by the total number of children; hence, a single bad child forms a small deviation in a large group but a large deviation in a small group.[9]

3.3 The Classification Algorithm

The prototypes are designed with easy application in mind; since the only surrounding categories that are considered are of the children of a node and of its predecessor on the same level, classification proceeds in a bottom-up fashion, moving through each tree level sequentially. Each node is examined and assigned to the category identified with the prototype to which it is nearest, i.e. that from which its distance is minimal. In the case of a tie, the more specific category is chosen.

The above process assigns primary categories only. When this is complete, a second phase begins; this phase assigns secondary categories. First, node repetitions are coordinated, in the following sense. If a sequence of nodes $n_1, n_2, ..., n_k$ can be found such that $k \geq 2$ and n_{i+1} is the sole descendent of n_i for all $1 \leq i < k$, this sequence forms a repetition to be coordinated. The nodes are collapsed into one (with parent the parent of n_1 and children the children of n_k); if the nodes in the repetition belonged to different categories, the replacement node is assigned the category from which the distance was closest. The newly resulting tree is searched for nodes

[9]For instance, a *paragraph* child in a group with 10 *list item* children will not by itself prevent the parent from matching the *list* prototype, but a *paragraph* child with a single *list item* sibling probably will.

that meet the descriptions of the defined secondary categories, and these nodes are assigned the corresponding categories.

4 Performance

4.1 Implementation

The algorithm is implemented in Lisp. To test its accuracy, it was run on correct segmentations of 51 pages, spanning several randomly selected computer science technical reports. The technical reports were scanned in; then, OCR was performed, using Xerox's ScanWorX software. ScanWorX yields output in a format called XDOC, which was parsed by a perl script.[10] A human-aided segmentation was performed on the result (so as not to introduce segmentation errors, since only the classification algorithm was currently being tested); the classification algorithm was run on the result of this correct segmentation. Finally, the results of the classification algorithm were displayed by a tcl script. Each page was processed using the results of the preceding page (if any) and the segmentation of the succeeding page (if any) to provide the necessary context.

4.2 Evaluation

To evaluate the accuracy of the classification, each node in the segmentation hierarchy was examined for which there exists in the system a suitable logical structure description. Those nodes that describe logical structures not yet included (such as *theorem-and-proof*) were not evaluated. Each examined node was characterized as *correct*, *overgeneralized*, or *incorrect*. A correct node has been classified as the most specific logical structure in the system that appropriately describes it. An overgeneralized node has been classified as an appropriate logical structure, but there exists in the system a more precise logical structure that is also appropriate. (For example, a theorem may be classified as a *paragraph*.) Finally, an incorrect node has been assigned an inappropriate structure type.

These characterizations yield two slightly different measures of accuracy. *Precise accuracy* is the percentage of classifications that are characterized as correct; *generalized accuracy* is the percentage of classifications that are characterized as either correct or overgeneralized, i.e., those categorizations that are not errors. That is:

$$\begin{aligned} \textit{Precise accuracy} &= \frac{\textit{Correct}}{\textit{Correct} + \textit{Overgeneralized} + \textit{Incorrect}} \\ \textit{Generalized accuracy} &= \frac{\textit{Correct} + \textit{Overgeneralized}}{\textit{Correct} + \textit{Overgeneralized} + \textit{Incorrect}} \end{aligned}$$

4.3 Results

The page-by-page results are shown in Figure 2. The overall precise accuracy is 81.4%, with an average of 83.5% precise accuracy per page. The overall generalized accuracy is 86.9%, with an average of 88% accuracy per page.

A sample page and a display of part of its hierarchy are shown in Figure 3. All primary structures in this page are found correctly, including the section heading (not shown, a child of the section node), the list and its items, and paragraphs. Unfortunately, an error is made in the second

[10] Many thanks to Jim Davis for providing the bulk of this script.

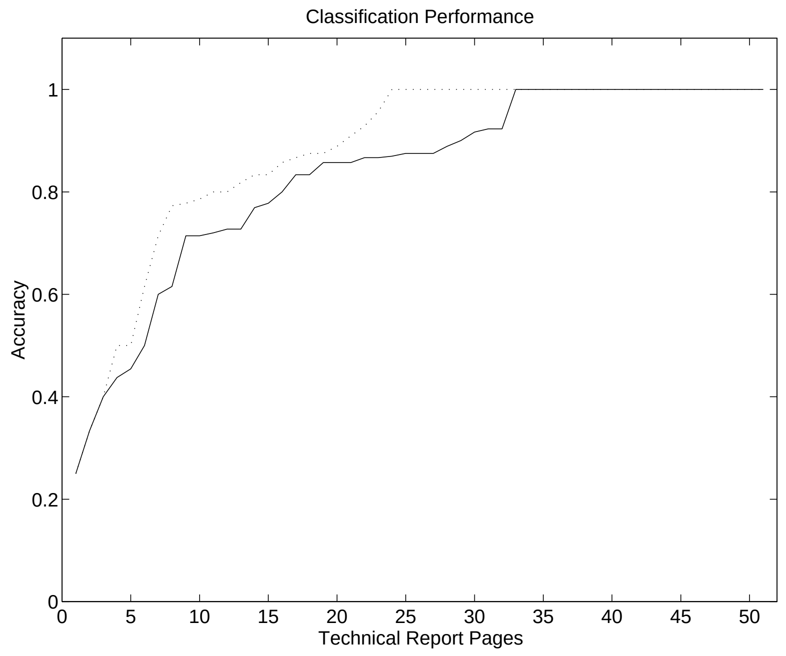
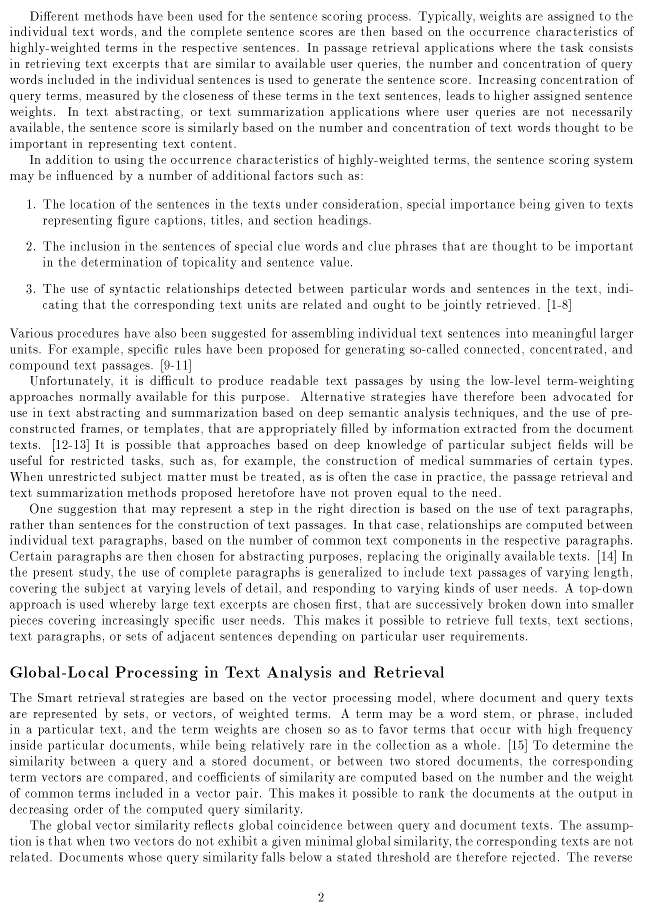


Figure 2: Performance of the classification algorithm. The solid line shows the gross accuracy on the (sorted) technical report pages. The dotted line shows the generalized accuracy.

phase of the algorithm; the paragraph parts surrounding the list are taken for paragraphs. This is a typical error, resulting from the fact that in order to find the secondary structures *paragraph part* and *paragraph group*, the algorithm must often decide which of several nodes is the true *paragraph*. That is, given a configuration like the one in the figure at the end of the first pass, the algorithm must decide whether the parent is a *paragraph* and the children *paragraph part*s or the parent is a grouping and the children are *paragraph*s. When a parent is closer to the *paragraph* prototype than its children, it is presumed to be the real paragraph; when the children are all closer than the parent, they are presumed to be the real paragraphs. (It is this process that yields the *paragraph group* represented by the word "Paragraphs" in the figure; its children are all closer to the *paragraph* prototype than it is.) In the case of a tie, no changes are made, yielding the two errors in the figure, which are the only errors made on that page.

5 Discussion and Future Work

This paper has presented a simple, prototype-based approach to classifying document elements as logical structures while relying only minimally on linguistic information. This yields two primary advantages: (1) novel documents can be handled, as the text blocks need not meet a precise definition to be characterized and (2) the emphasis on geometric, rather than word-based, attributes of text makes the system both robust with respect to most OCR errors and far less language-dependent than it would otherwise be. While the approach here does not achieve language independence, it does reduce the number of changes necessary in order to change the language in which the system



Different methods have been used for the sentence scoring process. Typically, weights are assigned to the individual text words, and the complete sentence scores are then based on the occurrence characteristics of highly-weighted terms in the respective sentences. In passage retrieval applications where the task consists in retrieving text excerpts that are similar to available user queries, the number and concentration of query words included in the individual sentences is used to generate the sentence score. Increasing concentration of query terms, measured by the closeness of these terms in the text sentences, leads to higher assigned sentence weights. In text abstracting, or text summarization applications where user queries are not necessarily available, the sentence score is similarly based on the number and concentration of text words thought to be important in representing text content.

In addition to using the occurrence characteristics of highly-weighted terms, the sentence scoring system may be influenced by a number of additional factors such as:

1. The location of the sentences in the texts under consideration, special importance being given to texts representing figure captions, titles, and section headings.
2. The inclusion in the sentences of special clue words and clue phrases that are thought to be important in the determination of topicality and sentence value.
3. The use of syntactic relationships detected between particular words and sentences in the text, indicating that the corresponding text units are related and ought to be jointly retrieved. [1-8]

Various procedures have also been suggested for assembling individual text sentences into meaningful larger units. For example, specific rules have been proposed for generating so-called connected, concentrated, and compound text passages. [9-11]

Unfortunately, it is difficult to produce readable text passages by using the low-level term-weighting approaches normally available for this purpose. Alternative strategies have therefore been advocated for use in text abstracting and summarization based on deep semantic analysis techniques, and the use of pre-constructed frames, or templates, that are appropriately filled by information extracted from the document texts. [12-13] It is possible that approaches based on deep knowledge of particular subject fields will be useful for restricted tasks, such as, for example, the construction of medical summaries of certain types. When unrestricted subject matter must be treated, as is often the case in practice, the passage retrieval and text summarization methods proposed heretofore have not proven equal to the need.

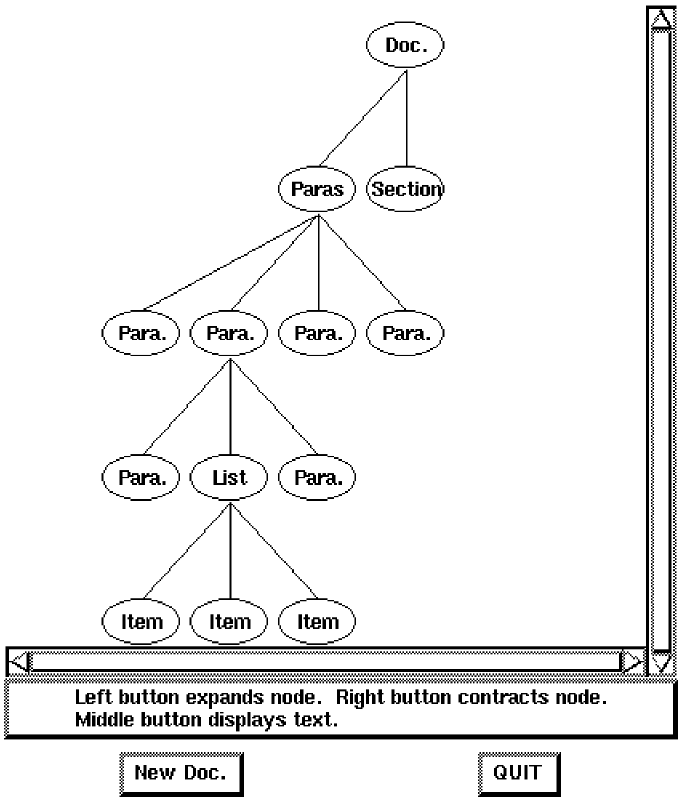
One suggestion that may represent a step in the right direction is based on the use of text paragraphs, rather than sentences for the construction of text passages. In that case, relationships are computed between individual text paragraphs, based on the number of common text components in the respective paragraphs. Certain paragraphs are then chosen for abstracting purposes, replacing the originally available texts. [14] In the present study, the use of complete paragraphs is generalized to include text passages of varying length, covering the subject at varying levels of detail, and responding to varying kinds of user needs. A top-down approach is used whereby large text excerpts are chosen first, that are successively broken down into smaller pieces covering increasingly specific user needs. This makes it possible to retrieve full texts, text sections, text paragraphs, or sets of adjacent sentences depending on particular user requirements.

Global-Local Processing in Text Analysis and Retrieval

The Smart retrieval strategies are based on the vector processing model, where document and query texts are represented by sets, or vectors, of weighted terms. A term may be a word stem, or phrase, included in a particular text, and the term weights are chosen so as to favor terms that occur with high frequency inside particular documents, while being relatively rare in the collection as a whole. [15] To determine the similarity between a query and a stored document, or between two stored documents, the corresponding term vectors are compared, and coefficients of similarity are computed based on the number and the weight of common terms included in a vector pair. This makes it possible to rank the documents at the output in decreasing order of the computed query similarity.

The global vector similarity reflects global coincidence between query and document texts. The assumption is that when two vectors do not exhibit a given minimal global similarity, the corresponding texts are not related. Documents whose query similarity falls below a stated threshold are therefore rejected. The reverse

2



Left button expands node. Right button contracts node.
Middle button displays text.

Figure 3: A sample page and its results. All primary structures and most secondary structures are correctly identified, but two secondary errors can be seen in the siblings of the *list* node.

functions or to add a language to its vocabulary.[11] (In the case of languages that share symbols, only the prototypes for linguistic structures need be changed; otherwise, both symbolic and linguistic structures must be changed.)

Further investigation is warranted in at least the following areas.

Learning It might be possible to use machine learning techniques to form prototypes and/or to choose appropriate weights for different elements of the distance measurement. This would remove the danger of encoding idiosyncrasies of the prototype designer's point of view.

Distance A good weighting scheme for distance elements, whether automatically (which is preferable) or manually derived might greatly improve performance. Similarly, better basic distance measurements may exist; for example, a distance measure between geometric representations that reflects visual similarity more closely than applying the Levenshtein metric to the strings should be found.

Relationship to Segmentation Preliminary investigations have begun into cleaning the segment tree based on the categories and distances found in the classification stage, but this remains to be rigorously explored. It also remains to investigate the possibility of interleaving by classifying the elements of a hierarchy level as soon as they are found and using these results to find the next level. This alteration may improve the performance of both algorithms.

In addition, the current set of prototypes is not quite complete; work continues on extending the system. The initial results presented here are, however, encouraging. We are coming to be able to identify a variety of logical structures with a great degree of certainty, which will provide support for automated markup and the multiplicity of (possibly distributed) browsing and searching applications that rely on it.

Acknowledgments

The author is very grateful to John Hopcroft for his guidance and support and also to Daniela Rus for hers. Special thanks to Jim Davis for many enlightening discussions. Thanks also to Annie Liu, Matthew Morgenstern, Amit Singhal, and all other members of the Information Capture and Access group at Cornell for useful comments and support.

References

[1] P. D. Bruza and T. W. C. Huibers. Detecting the erosion of hierarchic information structures. In *Proceedings of the Workshop on Principles of Document Processing*, Seeheim, 1994.

[2] Jacques Derrida. *Glas*. Collection Digraphe. Editions Galilee, Paris, 1974.

[3] Peter Fankhauser and Yi Xu. *MarkItUp!* An incremental approach to document structure recognition. *Electronic Publishing*, 6(4):447–456, 1993.

[4] Lloyd Alan Flechter and Rangachar Kasturi. Segmentation of binary images into text strings and graphics. In *Proceedings of the SPSE 40th Conference on Applications of Artificial Intelligence V*, volume 786, pages 533–540, 1987.

[11] This is the case, of course, only for languages that share most layout conventions. This holds for a great number of languages, however.

[5] Lloyd Alan Fletcher and Rangachar Kasturi. A robust algorithm for text string separation from mixed text/graphics images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(6):910–918, November 1988.

[6] E. Fox, Q. Chen, and R. France. Integrating search and retrieval with hypertext. In E. Berk and J. Devlin, editors, *Hypertext/Hypermedia Handbook*, pages 329–355. McGraw-Hill, New York, 1991.

[7] Hiromichi Fujisawa, Yasuaki Nakano, and Kiyomichi Kurino. Segmentation methods for character recognition: From segmentation to document structure analysis. In *Proceedings of the IEEE*, volume 80, pages 1079–1092, July 1992.

[8] Marti A. Hearst and Christian Plaunt. Subtopic structuring for full-length document access. In *Proceedings of SIGIR*, Pittsburgh, PA, 1993.

[9] A. Jain and S. Bhattacharjee. Address block location on envelopes using gabor filters. *Pattern Recognition*, 25(12), 1992.

[10] Leslie Lamport. LaTeX*: A Document Preparation System.* Addison-Wesley, Reading, 1986.

[11] Masaaki Mizuno, Yoshitake Tsuji, Toshiyuki Tanaka, Haruhiko Tanaka, Masao Iwashita, and Tsutomu Temma. Document recognition system with layout structure generator. *NEC Research and Development*, 32(2):430–437, July 1991.

[12] G. Nagy, S. Seth, and M. Vishwanathan. A prototype document image analysis system for technical journals. *Computer*, 25(7), 1992.

[13] Gilbert B. Porter and Emil V. Rainero. Document reconstruction: A system for recovering document structure from layout. In *Proceedings of the Conference on Electronic Publishing*, pages 127–141, 1992.

[14] M. Armon Rahgozar, Zhigang Fan, and Emil V. Rainero. Tabular document recognition. In *SPIE Proceedings*, San Jose, February 1994.

[15] Daniela Rus and Kristen Summers. Using white space for automated document structuring. In *Proceedings of the Workshop on Principles of Document Processing*, Seeheim, 1994.

[16] Gerard Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer.* Addison-Wesley Publishing Company, Reading, 1989.

[17] D. Sankoff and J. Kruskal. *Time warps, string edits, and macromolecules: the theory and practice of sequence comparison.* Addison-Wesley, 1983.

[18] Dacheng Wang and Sargur N. Srihari. Classification of newspaper image blocks using texture analysis. *Computer Vision, Graphics, and Image Processing*, 47:327–352, 1989.

[19] Gio Wiederhold. Mediators in the architecture of future information systems. *Computer*, pages 38–49, March 1992.

[20] Hiroshi Yashiro, Tatsuya Murakami, Yoshihiro Shima, Yasuaki Nakano, and Hiromichi Fujisawa. A new method of document structure extraction using generic layout knowledge. In *International Workshop on Industrial Applications of Machine Intelligence and Vision*, Tokyo, April 1989.