Using White Space for Automated Document Structuring

Daniela Rus and Kristen Summers
Department of Computer Science
Cornell University

July 30, 1994

Abstract

We present and analyze efficient algorithms for the automated recognition and interpretation of layout structures in electronic documents. The key idea is to use the patterns in the distribution of white space in a document to recognize and interpret its components. The recognition algorithm divides the document into a hierarchy of logical elements; the interpretation algorithms classify these divisions as base-text, tables, indented lists, polygonal drawings, and graphs. We present experimental data and discuss an information access application. Our methodology allows the automatic markup of documents[1] and the creation of multi-level indices and browsing tools for electronic libraries.

1 Introduction

The proliferation of information in electronic form, the development of high-speed networking and the evolution of digitized audio and video, make the flexible browsing and retrieval of electronic documents one of the grand challenges of computer science. Many word-based tools have been proposed in response to this challenge [28, 13, 8, 15]. Our methods use non-textual cues to complement the linguistic approach to information retrieval, as well as to provide more versatility for indexing and browsing. By *non-textual cues* we mean layout-oriented document components like figures, fonts, marks (such as bullet points), indentation, *etc.* For example, we can retrieve the paper in Figure 1 by posing a query that is visual in nature: "find the paper with a table at the top of the fourth page". Flexible and sophisticated document manipulation tools greatly increase the practical usefulness of electronic libraries. Unless users find electronic documents to be as easily manipulable as paper, they will resist the transition.

A typical browsing application in an electronic library is illustrated by the following example. Mike is interested in surveying papers on mechanical walkers. A keyword search over the Ei-Compendex database produces 936 records. Mike wants the flexibility to browse over the titles and authors of the papers. When a particular title appears interesting he wants a more detailed view, consisting of that paper's section titles. If this view continues to sustain his interest he wants to access more detailed pieces of the document, such as the full text of the abstract, all the figures that go with a particular section, or the performance tables for the mechanical walker. Current systems do not provide this flexibility; the work in this paper leads to systems that do.

[1]For instance in the SGML format

We believe that the non-textual content of documents complements the textual content and should play an equal role in information retrieval. An illustrative example from the realm of speech (where non-verbal cues include facial expression, gestures, postures, tone of voice, *etc.*) is found in [27]'s description of the hilarity generated on an aphasia ward by a presidential address. The patients, with no understanding of words as such but with a heightened sense of the visual and tonal aspects of communication, found the president's dishonesty so obvious as to be laughable. Their close attention to the details of the non-linguistic components of speech enabled them to see clearly the Actor's dissimulation, obscured for many normal viewers. This parallel is not just anecdotal. In our previous work [26, 25], we discuss classes of retrieval tasks that are supported by non-textual cues but not by word-based systems.

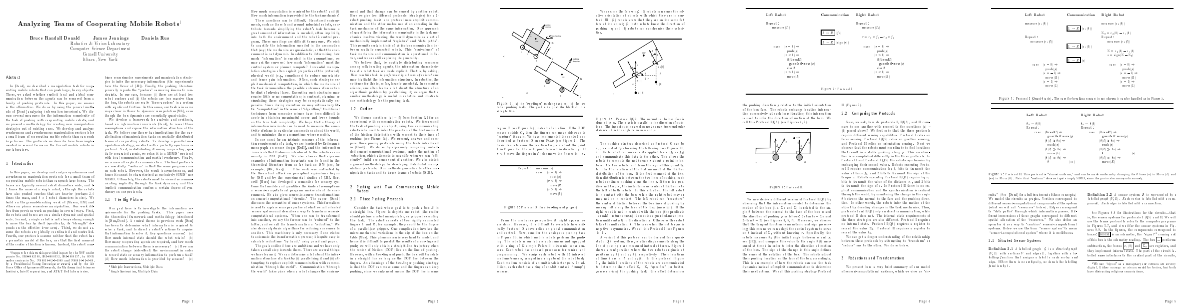


Figure 1: A zoomed-out view of an article on coordinated pushing with mobile robots. The first page has the title/author information represented as centered text. Page three has an itemized list. Pages three and four contain polygonal drawings of boxes. Pages four and five contain robot protocols presented in tabular form.

Structural, non-textual information can be used in the development of a variety of tools for document manipulation. Consideration of the logical locations of word occurrence (for instance, in the abstract versus a later section) can be used for improved weighting schemes in information retrieval systems. Logical structure may be directly mapped to automated markup in a format such as SGML [12]. Documents may be indexed according to the presence of (and relationships between) various logical structures. Such an index can enable flexible hierarchical browsing, based on expanding components to their subcomponents on request; it can also enable component-based queries such as "sections containing tables" or "figures from relevant sections," where relevance is determined by information retrieval techniques. Automated hyperlinking may proceed based on structural information, *e.g.*, a match between an element of the reference section in one document of a collection and the title of another might yield a reference hyperlink.

Our approach is to automate the *layout* analysis of electronic documents, in order to provide support for automated markup, intelligent document navigation, and conceptualized retrieval. Our methodology consists of developing algorithms to automatically detect and encode views of electronic documents that users might wish to request. Rather than explicitly and manually constructing all the possible models of a document, we provide flexibility by automatically building a hierarchy of partial models of the document. The layers in the hierarchy highlight different levels of detail. For example, a useful *zoomed-out* view (see Figure 1) reveals the presence of coarse-grained structures like figures, paragraphs, sections, *etc.*[2], but hides the actual text.

Our algorithms are efficient, geometric in nature, and apply to *generic* documents, *i.e.*, the documents are not marked up, and nothing is known or assumed about the formatting tools that created them; we do not require even a very general Document Type Definition or grammar to define the document's representation of structure. Instead, we rely on the analysis of white space, a simple concept that we believe to be universally significant in text. The basic information consists of the quantity and location of white space in a block of text. We provide support for customizing browsing and searching by algorithms for (1) segmentation, that is, discovering the topology of the layout hierarchy, and (2) classification, that is, labeling the nodes in the hierarchy with the structures they represent: paragraphs, itemized lists, sections, tables, and line drawings. The segmentation algorithm divides the document into a hierarchy of layout components by examining the text contours only.[3] The interpretation algorithms classify document portions as base-text, tables, polygonal drawings and graphs, and indented lists. We have implemented a prototype system based on these algorithms. Our experimental results are obtained by using the the Cornell Computer Science Technical Report electronic library and our data is presented in Section 4.

1.1 Outline

In the remainder of this section, we discuss previous related work.

In Section 2, we present the segmentation algorithm, which divides the document into a hierarchy of layout components by examining the text contours only (*i.e.*, the left and right white space). The algorithm itself is described in Subsection 2.1. In Subsection 2.2, we discuss the indentation language used by this algorithm to capture basic zoomed-out document views.

In Section 3, we describe and analyze interpretation algorithms for classifying document portions as tables (in Subsection 3.1), polygonal drawings and graphs (in Subsection 3.2), indented lists (in Subsection 3.3), and base-text.

In Section 4 we describe the implementation of our system. We present data collected from the segmentation system and from the table detection system and we describe an experiment that uses our algorithms for (1) automatically marking up documents and for (2) non-textual retrieval tasks.

Finally, in Section 5, we talk about the implications of our results and future directions of work.

1.2 Previous Work

We are inspired by research in three areas: information retrieval, automated document structuring, and document processing as described in [3].

[2]The term zoomed-out refers to a document view in which only the shapes of its components are distinguishable; this corresponds roughly to the human eye perception of a paper document from a great distance. For example, the representation of a computer science technical paper includes the title and author information at the first level of the tree, the section structure at the second level, and the individual blocks of each section (text paragraphs, figures, tables, code) below.

[3]Our notion of layout components requires readily identifiable reasons to group together the included document elements separately from their surrounding context. Subsections, itemized lists, and figures are examples of layout components.

- *Information Retrieval.* In classical information retrieval [2, 5, 7, 11, 13, 28, 29], unstructured text is indexed based on its keywords. Each keyword that occurs in a collection is viewed as an axis in a large space. A document is viewed as a point in this vector space; its dimension along each axis is given by the frequency of the defining word in the document. The approach of representing documents yields an index that encodes information about document topics; we seek to supplement this with the content cues found in coarser patterns: tables, graphs, pictures, *etc.*

- *Automated document structuring.* The goals of the document structuring community are to identify the key constituents of a document image (sections, paragraphs, pictures, *etc.*) from its layout and represent the logical relationships between these constituents. This is a first step towards analyzing the contents of documents.

  Most work on document structuring applies only to documents whose format is known in advance. The systems in [14, 22, 24, 33], segment pages of one format type each. The systems in [21, 32] find a logical document hierarchy composed of paragraphs, subsections, and sections, based on knowledge of standard printers' formatting rules. In [23], hierarchies are built that also include intermediate structures; knowledge of the formatting rules for a particular document's style is required.[4]

  Our approach finds document divisions at multiple levels of granularity. The algorithm is driven by the structure of the document, rather than prior knowledge of document layout rules. Thus, the height of the extracted hierarchy reflects the degree of nested structure within the document, and the algorithm performs reasonably when confronted with novel documents. Information about printers' rules can be incorporated naturally, in order to aid in characterization of the revealed structures.

- *Document Processing.* Work in document processing addresses the issue of document layout generation. In [3, 4, 10], a formal separation of document structure from its layout effects is proposed. The suggestion is that a document should be defined by its content and structure, according to a *source grammar*. Then an instantiation of the document can be formed by applying the layout rules of a *result grammar* to this defined structure. Different instantiations of the same document will be formed by applying different result grammars to the same document definition.

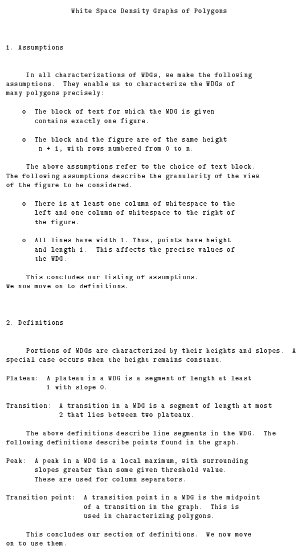
  We support this effort; if the approach suggested in these works, together with a small set of document source grammars, becomes standard, then access to the definition of a document will make the task of finding that document's structure trivial. We do not, however, believe that the problem of finding document structure in general will disappear; paper shows no sign of becoming obsolete, and this approach will not help us to find structure in a scanned-in document. Nor is it likely ever to be the case that *all* electronically available documents are represented as definitions in a known source language.

2 The Algorithm for Segmentation

In this section we present an algorithm that takes an electronic document and generates a hierarchy of partial models (or logical divisions.) The observation of repeated indentation structures

[4] For example, a document may be known to have been formatted with the LaTeX `article` style.

is used to partition the document into sections, subsections, and other finer-grained divisions, including arbitrary degrees of nesting. The resulting tree is suitable for indexing, browsing, and searching: its divisions may be indexed according to information retrieval techniques, a document may be browsed by navigating through the tree, and a set of trees may be searched for documents that include components with required properties.



```
                 White Space Density Graphs of Polygons


1. Assumptions


     In all characterizations of WDGs, we make the following
assumptions.  They enable us to characterize the WDGs of
many polygons precisely:

    o  The block of text for which the WDG is given
       contains exactly one figure.

    o  The block and the figure are of the same height
        n + 1, with rows numbered from 0 to n.

     The above assumptions refer to the choice of text block.
The following assumptions describe the granularity of the view
of the figure to be considered.

    o  There is at least one column of whitespace to the
       left and one column of whitespace to the right of
       the figure.

    o  All lines have width 1. Thus, points have height
       and length 1.  This affects the precise values of
       the WDG.

     This concludes our listing of assumptions.
We now move on to definitions.


2. Definitions


     Portions of WDGs are characterized by their heights and slopes.  A
special case occurs when the height remains constant.

Plateau:  A plateau in a WDG is a segment of length at least
          1 with slope 0.

Transition:  A transition in a WDG is a segment of length at most
             2 that lies between two plateaux.

     The above definitions describe line segments in the WDG.  The
following definitions describe points found in the graph.

Peak:  A peak in a WDG is a local maximum, with surrounding
       slopes greater than some given threshold value.
       These are used for column separators.

Transition point:  A transition point in a WDG is the midpoint
                   of a transition in the graph.  This is
                   used in characterizing polygons.

     This concludes our section of definitions.  We now move
on to use them.
```

Figure 2: A miniature document

This layout segmentation captures the divisions of the document's logical structure. In order to generate a tree that describes the logical structure all that remains is to classify its nodes. For example, Figure 2 contains a miniature document with a centered title and two sections, at the coarsest level. Each section contains a structure in which the pattern *paragraph, list* is repeated; lists lie nested within these structures.[5] The document division algorithm generates the tree in Figure 3. This tree provides a useful document view in its own right; it also constitutes the topology of the tree in Figure 4. Elements of the node classification tasks needed for the construction of Figure 4 are addressed in Section 3.

2.1 The Algorithm for Generating the Layout Hierarchy

The algorithm works in a bottom up manner, building a tree of indentation structures from a given set of leaves. At each level, the nodes are characterized according to an *indentation*

[5] The two sections contain different types of lists.

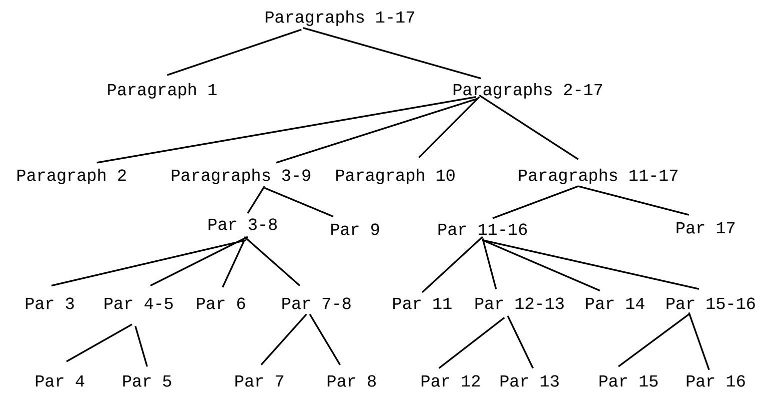


Figure 3: Indentation tree for the miniature document

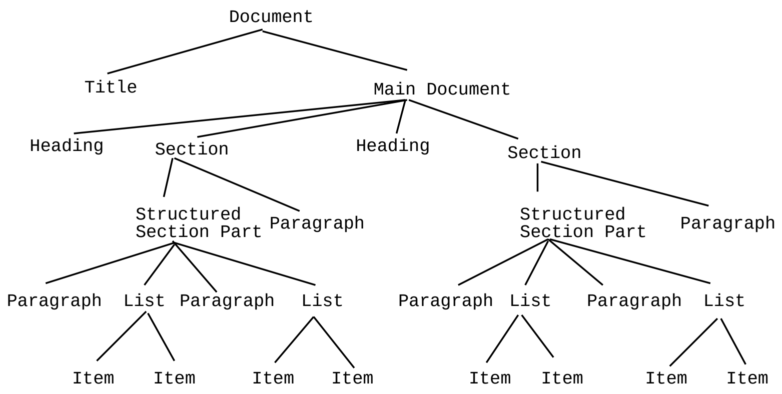


Figure 4: Goal tree for the miniature document

alphabet.[6] The level is then searched for repeated indentation patterns; sets of nodes that participate in such patterns are combined into new nodes at the tree's next level. Each of the old nodes becomes a sub-block of the block described by the new node. A node that does not participate in a repeated pattern is considered to form a pattern of length one. Thus, at the next level it remains a node, and the block it describes has a single sub-block: itself. Then the indentations of the new level's nodes are characterized in the indentation alphabet, and the process repeats, until no new patterns can be found. The algorithm appears in Figure 5.

Our current implementation for step 1 divides the document into text blocks and blank blocks. Each document line in such a block constitutes a sub-block. Other segmentation algorithms are possible, such as the one proposed in [26].

The hierarchical document divisions allow an arbitrary degree of nesting in the structure

[6]Indentation alphabets provide symbols to express the indentation of blocks of text and rules for matching sequences for these symbols. They are defined formally in Subsection 2.2.

1. Divide the document into blocks at the lowest level by an external algorithm.
2. Repeat until no changes are generated
   (a) Represent each sub-block by an appropriate character of the indentation alphabet.
   (b) Find sets of consecutive blocks that form repeating patterns in the indentation alphabet.
   (c) Combine these sets of blocks into blocks at the next higher level; the blocks that are being combined become sub-blocks.

Figure 5: The segmentation algorithm for logical tree generation

of the document. For example, in the algorithm in Figure 5, the enumerated list in step two forms an identifiable indentation structure, which is a part of the observable (higher-level) indentation structure consisting of steps one and two. With a carefully defined indentation alphabet (described in Subsection 2.2.3), the structure formed by the subsections and sections is also identifiable. The degree of such nesting in a given paper is arbitrary. Correspondingly, our bottom-up algorithm builds trees of arbitrary depth.

2.2 Indentation Alphabets

We now provide a formal specification of indentation alphabets; then we discuss the particular alphabet that our version of the algorithm uses. Finally, we offer a formal specification and analysis of the language defined by patterns in any indentation alphabet that shares the basic properties of ours.

2.2.1 Formal Definition of an Indentation Alphabet

Definition 2.1 *An* indentation alphabet *is a triple $\langle \Sigma, D, R \rangle$, where Σ is a set of symbols; D is a set of definitions for the symbols in Σ, such that, given a block of text made up of one or more sub-blocks, exactly one sequence of definitions describes the sub-blocks that make up the block; and R is a subsumption relation among the symbols in Σ, in the sense of Definition 2.2.*

Definition 2.2 *A* subsumption relation *R in an indentation alphabet $\langle \Sigma, D, R \rangle$ is a lattice over $\Sigma^* \cup \top$, where $\top$ is a special "top" symbol such that $\alpha \in \Sigma^* \Rightarrow \top \succeq_R \alpha$.*

If $\alpha \succeq_R \beta$, this is interpreted to mean that α represents an indentation form that subsumes the form represented by β. Thus, in the process of comparing sequences of indentation alphabet symbols, an instance of β may be matched against an instance of α, yielding a pattern of α. Moreover, if $\alpha \succeq_R \gamma$ as well, then both β and γ may be matched as instances of the pattern α (without a need for the explicit appearance of α). More precisely, the interpretation of the subsumption relation is: if the least upper bound of two sequences α and β is not $\top$, then the two sequences match, yielding a pattern of their least upper bound.

For example, *centered* $\succeq_R$ *both-justified* indicates that the form *centered* subsumes the form *both-justified*; a piece of text that is justified on both the left and the right may be a long piece of centered text. Hence, in comparing two sequences α and β, if an instance of *centered* is encountered in position i of α and an instance of *both-justified* is encountered in position i of β, with all previous symbols in the sequences matching, then these symbols are considered to match; the examined prefixes of α and β are instances of a pattern γ with *centered* in position i.

To see the effects of the least upper bound, consider an indentation alphabet containing a symbol *paragraph-start*, which represents an indentation followed by a move to the left margin. Suppose this alphabet also contains the symbols *indent*, *left-margin*, *centered*, *both-margins* and *right-margin*, each of which represents a single sub-block. Thus, the symbol *paragraph-start* appears instead of the consecutive symbols *indent*, *left-margin*. Centering that precedes a move to the left margin may also indicate a paragraph beginning, however; the appearance of centering may be an accident of line length. For example, consider Figure 6. The first line of the first paragraph is centered with respect to the left and right margins of the text. Similarly, right-justification may be an accident of line length, so a line that meets the right margin (but not both margins) followed by a move to the left margin may indicate a paragraph beginning. This can be seen in the second paragraph of Figure 6. To make the subsumption relation reflect these possibilities, it includes *paragraph-start* $\succeq_R$ *centered left-margin* and *paragraph-start* $\succeq_R$ *right-margin left-margin*. The representations of the paragraphs in Figure 6 are given in Figure 7. They do not match directly, but the subsumption relation matches them, as shown in Figure 8.

```
     The retrieval of information from conceptually-specified
queries has become one of the grand challenges of computer science
due to the proliferation of information in electronic form, the
development of high-speed networking, and the evolution of digit-
ized audio and video.
     Users will require tools for filtering information from large
distributed repositories.  The challenge as a whole is posed as
the Multimedia Information Capture and Access problem:  given an
electronic environment, capture it by acquiring partial models and
access the data according to users' requests.
```

Figure 6: Two paragraphs

Paragraph 1	**Paragraph 2**
Centered	*Right-Margin*
Left-Margin	*Left-Margin*
Left-Margin	*Left-Margin*
Left-Margin	*Left-Margin*
Left-Margin	*Left-Margin*

Figure 7: Representations of the paragraphs

We consider indentation alphabets that include the special symbols *start-block* and *end-block*. These symbols begin and end each block description, and they simplify the charac-

Paragraph 1	**Paragraph 2**	**Subsuming Pattern**	
Centered	*Right-Margin*		
Left-Margin	*Left-Margin*		
Left-Margin	*Left-Margin*		$\Longrightarrow$
Left-Margin	*Left-Margin*		
Left-Margin	*Left-Margin*		

Paragraph 1	**Paragraph 2**	**Subsuming Pattern**	
		Paragraph-Start	
Left-Margin	*Left-Margin*		$\Longrightarrow \ldots$
Left-Margin	*Left-Margin*		
Left-Margin	*Left-Margin*		

Paragraph 1	**Paragraph 2**	**Subsuming Pattern**
		Paragraph-Start
		Left-Margin
		Left-Margin
		Left-Margin

Figure 8: Matching the paragraphs

terization of the type of patterns we seek (*i.e.*, those which include only complete blocks). Different choices of indentation alphabets yield different results; the alphabet may be adjusted for different goals without affecting the algorithm. Similarly, the method of deriving blocks and sub-blocks may be altered without affecting the indentation alphabet. This modularity increases the power and ease of use of both elements of the document divider.

2.2.2 Useful Subsumption Relation Types

We consider indentation alphabets whose subsumption relations have special properties that aid their computational use. We define a set of such properties that will enable us to find the least upper bound of two strings by comparing one symbol at a time, without backtracking.

First, we require that the least upper bound of two strings be derivable from the least upper bounds of their substrings.

Definition 2.3 *The subsumption relation of an indentation alphabet* $\langle \Sigma, D, R \rangle$, *is* normal *if* $(\alpha \succeq_R \beta \wedge \gamma \succeq_R \delta) \Longrightarrow (\alpha\gamma \succeq_R \beta\delta)$.

That is, in a normal subsumption relation, concatenation does not invalidate subsumption. In order for substring subsumption to fully control string subsumption, though, we need the following stronger condition.

Definition 2.4 *A normal subsumption relation* R *of an indentation alphabet* $\langle \Sigma, D, R \rangle$*is* divisible *if*
$(\mathrm{lub}(\alpha, \beta) = \gamma) \Longrightarrow (\forall \delta, \eta \in \Sigma^* : \mathrm{lub}(\alpha\delta, \beta\eta) = \gamma \mathrm{lub}(\delta, \eta))$.

This indicates that a pattern found by looking at the beginning of a sequence will not be invalidated by symbols later in the sequence. In pattern matching based on a divisible subsumption relation, substrings may be considered from left to right, without requiring backtracking. We define a new lattice, made up of the substrings that will need to be considered.

Definition 2.5 *The* minimal subsumption relation $R_{\min}$ *of an indentation alphabet* $\langle \Sigma, D, R\rangle$ *with a divisible* R *is the lattice over a subset of* $\Sigma^* \cup \top$ *such that*

$$(\alpha \succeq_{R_{\min}} \beta) \Longleftrightarrow$$
$$((\alpha \succeq_R \beta) \wedge \quad (\forall \alpha', \beta' \in \Sigma^* :$$
$$(\exists \gamma, \eta \in \Sigma^* : (\gamma \neq \varepsilon \vee \delta \neq \varepsilon) \wedge (\alpha = \alpha'\gamma \wedge \beta = \beta'\eta)) \Rightarrow \neg(\alpha' \succeq_R \beta'))).$$

Thus, α subsumes β in $R_{\min}$ if in R, α subsumes β, but no proper prefixes of α and β bear this relationship. The rules of $R_{\min}$ may be used for pattern matching according to a divisible R. For matching to proceed based on individual symbols, we need to constrain the form of $R_{\min}$.

Definition 2.6 *A divisible subsumption relation* R *of an indentation alphabet* $\langle \Sigma, D, R\rangle$ *is* symbol-oriented *if*

$$\forall \alpha, \beta \in \Sigma^* : ((\alpha \succeq_{R_{\min}} \beta) \Longrightarrow |\beta| = 1).$$

With a symbol-oriented subsumption relation, we can perform matches on one character at a time, in a manner based on the following definitions.

Definition 2.7 *The* symbol lattice*,* R'*, of an indentation alphabet* $\langle \Sigma, D, R\rangle$ *with a symbol-oriented* R *is given by:*

$$\forall x, y \in \Sigma : ((x \succeq_{R'} y) \Longleftrightarrow$$
$$(\exists x_1, \ldots x_n \in \Sigma, \alpha_0, \ldots, \alpha_n \in \Sigma^* :$$
$$(x\alpha_0 \succeq_{R_{\min}} x_1) \wedge (x_1\alpha_1 \succeq_{R_{\min}} x_2) \wedge (x_2\alpha_2 \succeq_{R_{\min}} x_3) \wedge \ldots \wedge (x_n\alpha_n \succeq_{R\min} y)),$$

and $\forall x \in \Sigma : \top \succeq_{R'} x$*.*

The sequence $\alpha_0\alpha_1 \ldots \alpha_n$ *is called the* y-x addition*.*

Then, a pair of sequences to be matched according to a symbol-oriented subsumption relation can be thought of as a pair of stacks of symbols. At each step in the match, a symbol is popped off of each stack. Suppose these symbols are x and y. If the least upper bound z of x and y in the symbol lattice is not $\top$, then the match (so far) succeeds; z is appended to the subsuming pattern formed so far, the x-z addition is pushed onto x's stack, and the y-z addition is pushed onto y's stack.

The least upper bounds and additions can be calculated in advance, leading to an alternative representation of these subsumption relations as pairs of the form $\langle (x, y, z), (\alpha, \beta)\rangle$, where α is the x-z addition, and β is the y-z addition.

2.2.3 Our Indentation Alphabet

We use the following indentation alphabet. When characterizing the first sub-block in a block, the left and right margins of the document serve as the left and right edges of the "preceding sub-block."

- $\Sigma = \{$ *Start-Block, End-Block, Both-Justified, Left-Justified,* $\langle$*Centered-In,* $m\rangle$, $\langle$*Centered-Out,* $n\rangle$, *Centered, Left-In, Left-Out,* $\langle$*Blank,* $h\rangle\}$, where m, n and h range over the positive integers.

- $D =$

  Start-Block Represents the beginning of a new block; does not correspond to a sub-block.

  End-Block Represents the end of the current block; does not correspond to a sub-block.

  Both-Justified A set of one or more consecutive sub-blocks, not all of which are blank, in which the text sub-blocks are both left-justified and right-justified with respect to the preceding sub-block.

  Left-Justified A set of one or more consecutive sub-blocks, not all of which are blank, in which the text sub-blocks are left-justified but not right-justified with respect to the preceding sub-block.

  $\langle$ **Centered-In, m** $\rangle$ A set of sub-blocks containing m text sub-blocks, which are centered with respect to the preceding sub-block. The left edge of each of the m sub-blocks lies to the right of the left edge of its predecessor.

  $\langle$ **Centered-Out, n** $\rangle$ A set of sub-blocks containing n text sub-blocks, which are centered with respect to the preceding text sub-block. The left edge of each of the n sub-blocks lies to the left of the left edge of its predecessor.

  Centered A set of one or more consecutive sub-blocks, not all of which are blank, in which the text sub-blocks are centered with respect to the preceding sub-block and cannot be represented as a sequence of the symbols $\langle$ Centered-In, m $\rangle$ and $\langle$ Centered-Out, n $\rangle$.

  Left In A non-centered text sub-block whose left edge lies to the right of the left edge of the preceding sub-block.

  Left Out A non-centered text sub-block whose left edge lies to the left of the left edge of the preceding sub-block.

  $\langle$ **Blank, h** $\rangle$ A set of one or more completely blank sub-blocks, of total height h.

- $R_{\min}$ is given by:
  Left-Justified $\succeq_{R_{\min}}$ *Both-Justified*,
  Centered $\succeq_{R_{\min}}$ *Both-Justified*,
  Centered $\succeq_{R_{\min}} \langle$*Centered-In,* $m\rangle$,
  Centered $\succeq_{R_{\min}} \langle$*Centered-Out,* $n\rangle$,
  Left-In $\succeq_{R_{\min}} \langle$*Centered-In,* $1\rangle$,
  Left-Out $\succeq_{R_{\min}} \langle$*Centered-Out,* $1\rangle$,
  $\langle$*Centered-In,* $1\rangle\langle$*Centered-In,* $m\rangle \succeq_{R_{\min}} \langle$*Centered-In,* $m+1\rangle$
  $\langle$*Centered-Out,* $1\rangle\langle$*Centered-Out,* $m\rangle \succeq_{R_{\min}} \langle$*Centered-Out,* $m+1\rangle$

R may also be expressed in our alternative form for symbol-oriented divisible subsumption relations as:

$\{\langle(\textit{Left-Justified, Both-Justified, Left-Justified}), (\varepsilon, \varepsilon)\rangle,$
$\langle(\textit{Centered, Both-Justified, Centered}), (\varepsilon, \varepsilon)\rangle,$
$\langle(\langle\textit{Centered-In}, m\rangle, \langle\textit{Centered-Out}, n\rangle, \textit{Centered}), (\varepsilon, \varepsilon)\rangle,$
$\langle(\textit{Centered}, \langle\ \textit{Centered-In}, m\rangle, \textit{Centered}), (\varepsilon, \varepsilon)\rangle,$
$\langle(\textit{Centered}, \langle\textit{Centered-Out}, n\rangle, \textit{Centered})\ (\varepsilon, \varepsilon)\rangle,$
$\langle(\langle\textit{Centered-In}, 1\rangle, \textit{Left-In, Left-In}), (\varepsilon, \varepsilon)\rangle,$
$\langle(\langle\textit{Centered-Out}, 1\rangle, \textit{Left-Out, Left-Out}), (\varepsilon, \varepsilon)\rangle,$
$\langle(\langle\textit{Centered-In}, m > 1\rangle, \textit{Left-In, Left-In}), (\langle\textit{Centered-In}, \ m-1\rangle, \varepsilon)\rangle,$
$\langle(\langle\textit{Centered-Out}, n > 1\rangle, \textit{Left-Out, Left-Out})\ (\langle\textit{Centered-Out}, \ n-1\rangle, \varepsilon)\rangle\}$

In this alphabet, no sub-block is ever initially identified as *Centered*; this symbol exists for its role in the subsumption relation. Sub-blocks may be determined to follow a pattern that includes the symbol *Centered*, at which point, an instance of $\langle$*Centered-In*, $m\rangle$ or $\langle$ *Centered-Out*, $m\rangle$, or *Both-Justified* becomes subsumed by the symbol *Centered*.

The height of blank sub-blocks provides a distinction between sections and subsections, as well as other nested structures in which the divisions are indicated in part by vertical white space. Since section headings and subsection headings are typically represented in different font sizes, the space surrounding these types of headings differs. Thus, a section heading and its surrounding blank lines will not match the pattern formed by a set of subsections; the subsections will be recognized as a structure at one level, and the section containing them will form part of a section structure at a higher level. Without considering font information directly, we take advantage of its effects to enable our layout structure trees to include the useful structures of sections and subsections.

2.3 The Indentation Pattern Language

We now provide a formal specification for the language of the indentation patterns found by our algorithm, called the *block indentation patterns*, using an indentation alphabet that includes the special symbols *Start-Block* and *End-Block*. (This indentation alphabet is presumed to be symbol-oriented.) In preparation for this definition and its analysis, we first define a notion of *fully distinguishable* symbols in an indentation alphabet and a useful relations E_R over the symbols of an indentation alphabet.

Definition 2.8 *Given an indentation alphabet $\langle\Sigma, D, R\rangle$, with R symbol oriented, and $x, y \in \Sigma$, x and y are said to be* fully distinguishable *if, in the symbol lattice* $\mathrm{lub}(x, y) = \top$.

Intuitively, x and y are fully distinguishable if they can never be considered a match, regardless of what follows them.

Definition 2.9 *Given an indentation alphabet $\langle\Sigma, D, R\rangle$, with R symbol-oriented, and $\alpha \in \Sigma^*$, the* R-Extension of α, *written $E_R(\alpha)$, is given by $\{\beta | \alpha \succeq_R \beta\}$.*

The R-Extension of α thus consists of everything α subsumes, *i.e.*, the set of sequences that may match a pattern of α.

Definition 2.10 *An indentation alphabet $\langle\Sigma, D, R,\rangle$ is said to be* block-oriented *if the following are true:*

- *R is symbol-oriented.*

- *Σ contains the symbols Start-Block and End-Block, which are fully distinguishable from all other symbols in Σ.*

- *$\Sigma - \{$Start-Block, End-Block$\}$ contains at least one pair of fully distinguishable symbols.*

This describes the kind of indentation alphabets we use. We seek patterns that include only whole blocks, by requiring that they begin with *Start-Block* and end with *End-Block*. This leads to the following definition of the patterns to be found.

Definition 2.11 *Given a block-oriented indentation alphabet $I = \langle \Sigma, D, R \rangle$ the language of its* block indentation patterns *P_I is given by:*

$$P_I = \{\alpha_1\alpha_2\ldots\alpha_n \mid (n > 1) \wedge (\exists \beta \in \textit{Start-Block}\ \ (\Sigma - \{\textit{Start-Block, End-Block}\})^*\ \textit{End-Block}: (\forall i, 1 \leq i \leq n:\ (\alpha_i \in E_R(\beta)))\}$$

At first glance, parsing might seem like a natural method of finding patterns in this language. Unfortunately, the following result leads us to reject this approach.

Theorem 2.1 *For any block-oriented indentation alphabet $I = \langle \Sigma, D, R \rangle$, P_I is not a context-free language.*

Proof: Application of the pumping lemma for context-free languages to the string *Start-Block* $(xy)^n$ *End-Block Start-Block* $(xy)^n$ *End-Block Start-Block* $(xy)^n$ *End-Block* in which x, y are fully distinguishable elements of $\Sigma - \{$*Start-Block, End-Block*$\}$. Consider pumping down once. □

This language is, however, context-sensitive. This result has computational implications for the recognition of the indentation structures described here. Since context-sensitive parsing is not sufficiently efficient for use in the domain of large documents, we do not parse, but instead use a quadratic algorithm guaranteed to find repeated patterns if any exist.

2.4 A Layout Hierarchy Example

We now turn to an example application of the layout hierarchy generation algorithm. We apply this algorithm to the first section of the miniature document in Figure 2. Its initial representation is given in Figure 9.

On the first pass, two repeated patterns are found: a block of $\langle$*Blank*, 1$\rangle$ followed by a block of *Left-In, Left-In* and a block of $\langle$*Blank*, 1$\rangle$ followed by a block of *Left-In, Left-In, Left-Justified*. (These correspond to the two indented lists in the section.) Combining the elements of these patterns yields the set of divisions found on the second level (from the bottom) of the tree in Figure 3.[7]

Characterizing nodes at this level yields the representation in Figure 10. Now, a new repeated pattern appears: a block of *Left-Justified* followed by a block of $\langle$*Blank*, 1$\rangle$, *Left-In,*

[7] For readability, in the presentation of this tree, each node that remains unchanged at various levels of the tree has been represented only once, at the highest level at which it appears. The document divisions we find now include such nodes at all their levels. Similarly, blank blocks have been removed in Figure 3.

$\langle$*Blank*, 1$\rangle$, *Left-Justified* followed by a block of $\langle$*Blank*, 1$\rangle$. (This corresponds to the section's observable repetition of a paragraph followed by an indented list.) The nodes in this pattern merge, and the divisions of the third level of Figure 3 are found.

A similar process takes place simultaneously within Section 2, and as these iterate, the representation tree in Figure 3 is built.

Start-Block *Left-Justified* *End-Block*
Start-Block $\langle$ *Blank*, 2$\rangle$ *End-Block*
Start-Block *Left-In* *Left-Out* *Left-Justified* *End-Block*
Start-Block $\langle$ *Blank*, 1$\rangle$ *End-Block*
Start-Block *Left-In* *Left-In* *End-Block*
Start-Block $\langle$ *Blank*, 1$\rangle$ *End-Block*
Start-Block *Left-In* *Left-In* *End-Block*
Start-Block $\langle$ *Blank*, 1$\rangle$ *End-Block*
Start-Block *Left-In* *Left-Out* *Left-Justified* *End-Block*
Start-Block $\langle$ *Blank*, 1$\rangle$ *End-Block*
Start-Block *Left-In* *Left-In* *Left-Justified* *End-Block*
Start-Block $\langle$ *Blank*, 1$\rangle$ *End-Block*
Start-Block *Left-In* *Left-In* *Left-Justified* *End-Block*
Start-Block $\langle$ *Blank*, 1$\rangle$ *End-Block*
Start-Block *Left-In* *Left-Out* *End-Block*
Start-Block $\langle$ *Blank*, 3$\rangle$

Figure 9: Representation of miniature document's Section 1

Start-Block *Left-Justified* *End-Block*
Start-Block $\langle$ *Blank*, 2$\rangle$ *End-Block*
Start-Block *Left-Justified* *End-Block*
Start-Block $\langle$ *Blank*, 1$\rangle$ *Left-In* $\langle$ *Blank*, 1$\rangle$ *Left-Justified* *End-Block*
Start-Block $\langle$ *Blank*, 1$\rangle$ *End-Block*
Start-Block *Left-Justified* *End-Block*
Start-Block $\langle$ *Blank*, 1$\rangle$ *Left-In* $\langle$ *Blank*, 1$\rangle$ *Left-Justified* *End-Block*
Start-Block $\langle$ *Blank*, 1$\rangle$ *End-Block*
Start-Block *Left-Justified* *End-Block*
Start-Block $\langle$ *Blank*, 3$\rangle$

Figure 10: Second level representation of miniature document's Section 1

2.5 Future Extensions

The following areas are promising directions for further investigations.

- The document dividing algorithm occasionally creates trees with "extra" levels; for example, an itemized list may first be recognized as two or three smaller patterns, and only recognized as one cohesive pattern at a higher level in the tree. With a formalization of node descriptions, it should be possible to identify many, if not all, such extra levels and remove them from the tree.

- Document layout components sometimes consist of patterns that are interrupted. For example, a section may consist of a series of paragraphs, interrupted by a single indented list (all of which is set in from the left margin). The current version of the algorithm will not successfully recognize such a component; we are investigating a solution to this problem based on incorporating a coarser-grained indentation alphabet (such as one that distinguishes only text blocks and vertical white space of different heights), to be called upon in such instances. We need to find a specification of the appropriate situations in which to apply the coarser alphabet.

- The indentation alphabet needs to incorporate further analysis of the role of the right margin. At the moment, the standards for both-justification and centering are very rigid; this may or may not prove to be a good choice. Furthermore, no information is currently derived from the length of lines that are neither centered nor both-justified. Very short lines do, however, seem significant; they either indicate paragraph breaks or the inclusion of some structure other than running text. We need a characterization of the situations in which the right edge of a text block or line is significant.

- The indentation alphabet might benefit greatly from an extension including a consideration of font. Although we can usually detect changes in font size by the spacing between lines, this approach may not always prove reliable. We can only make this distinction when the changes in font are sufficiently large with respect to the granularity of the document representation. In the case of a scanned-in document, we are not assured that all significant font changes are readily identifiable in this manner. Furthermore, font types as well as sizes carry significant information. For example, the LaTeX environment `description` [18] includes a boldface heading at the beginning of each item. An indentation alphabet that included font type would be able to distinguish between a list of this kind and one without such headings.

3 Algorithms for Layout Classification and Interpretation

In this section we present algorithms for labeling the nodes in the layout tree. We focus on three types of high-level layout structures that occur in documents: tables, figures consisting of line drawings and function graphs, and itemized lists. Recognition of these document elements combined with a small amount of text analysis enables the execution of conceptually specified retrieval tasks, such as “find precision recall measures for the CACM collection,” “find performance graphs of the Byzantine agreement protocol,” and “find a list of assumptions in this paper” [1]. In the future, we would like to create a comprehensive library of structure detectors with performance guarantees.

3.1 A Classifier for Tables[8]

Webster's Seventh Dictionary defines a table as a *systematic arrangement of data usually in rows and columns for ready reference.* Implicit in this definition is a *layout* component and a *semantic* component: the data is organized in columns of similar information. For instance, the table in Figure 11 has a marked 3-column layout with uniform structure (*date,—, Cassini Mission event*). The table in Figure 11 is completely regular: the columns line up perfectly, the entries stay within their allotted space, and each record has the same format. Furthermore, the content similarities of the column entries are reflected lexically. Not all tables are this regular. Consider the example in Figure 12: the records are two lines long, the columns in the second line of a record do not align with the columns in the first, some columns extend into adjacent ones, and there are lexical irregularities in its records. The layout and lexical structures are still clear. Our goal is to check for column and content structure while tolerating irregularities to within specified error bounds.[9]

```
08/22/96 - Titan IV/Centaur Launch
03/29/97 - 66 Maja Asteroid Flyby
06/08/98 - Earth Gravity Assist
02/06/00 - Jupiter Gravity Assist
12/06/02 - Saturn Arrival
03/27/03 - Titan Probe Release
03/29/03 - Orbiter Deflection Maneuver
04/18/03 - Titan Probe Entry
06/30/03 - Iapetus Flyby
05/20/04 - Dione Flyby
09/12/04 - Enceladus Flyby
08/14/05 - Iapetus Flyby
12/31/06 - End of Primary Mission
```

Figure 11: Table of Key Scheduled Dates for the Cassini Mission

3.1.1 The Table Recognition Algorithm

The measure for the column layout of a block of text is given in terms of a data structure called the *white space density graph* and denoted by WDG. (It corresponds to the inverse of the vertical character projection.) Let B be a block of text of n rows and m columns and $w : \{c | c \text{ is a character }\} \rightarrow \{0, 1\}$ with $w(\text{" "}) = 1$ and $\forall c \neq \text{" "}, w(c) = 0$.

Definition 3.1 Vertical structure: *The* white space density graph *of B is the polygonal line* $\text{WDG} : [0, m] \rightarrow [0, 1]$ *defined by the points* $\text{WDG}(i) = \frac{1}{n} \sum_{j=0}^{n} w(B_{i,j}),\ \forall i \in N \cap [0, m]$.

Figure 14 shows the WDG associated with the table in Figure 12.

Definition 3.2 Deviations in vertical structure: *Given an error tolerance ϵ_v, a block of text has* column structure *if it occurs between two successive local maxima in the* WDG *above* $(100 - \epsilon)\%$.

[8] This section is based on [19].

[9] In this, we differ from [24], where the goal is to find the grid structure in a text component known to be a table.

Each local maximum is a candidate column separator. A candidate column is a real table column only if it has corresponding horizontal lexical structure. We are far from being able to identify row structure based on semantic content, but semantic uniformity in rows is highly correlated with lexical uniformity. We exploit this correlation in the design of a robust table detector.

The process of discerning lexical structure is facilitated by the presence of non-alphabetic characters. For example, it is easy to recognize that the entries of the first column in Figure 11 represent similar information, since they have a very regular and distinct lexical pattern. In distinguishing lexical structure, we identify the following equivalence classes of characters: alphabetic, numeric, and special (each special character is in its own class). Let $c_0, c_1, \ldots c_n$ denote the columns of a text block. We use regular expressions for generalizing the contents of a column. In Table 11, all items in the first column (the mission dates) can be described by the conjunctive regular expression $NN/NN/NN$, where N is a symbol denoting a number and / is a special character. The *lexical description* of a column c_i is a regular expression r_i that describes the smallest possible language that includes all elements of c_i, subject to the requirement that r_i is a non-trivial generalization of these elements. The regular expression $r_1 + \ldots + r_n$ is a *trivial* generalization of a given set of elements $r_1, \ldots, r_n$; all other generalizations are *non-trivial.*

Definition 3.3 Horizontal structure: *Consider the columns $c_1 \ldots c_n$ of a block of text satisfying Definition 3.2, and consider the lexical descriptions $r_1 \ldots r_n$ of these columns. This text also has* row structure *if and only if the language described by $r_1 r_2 \ldots r_n$ is non-trivial.*

```
100a Introduction to Computer Programming 4    Lec1 TR    9:05 Ives 120
         Wagner                                Lec2 TR   11:15 Ives 120
100b Introduction to Computer Programming 4    Lec1 TR    9:05 Olin 255
         Van Loan                              Lec2 TR   11:15 Ives 110
101  The Computer Age                     3         TR    1:25 Upson B17
         Segre
102  Intro Microcomputer Applications     3    Lec1 TR   10:10 RR 105
         Hillman (Ag. Engr.)                   Lec2 TR   12:20 RR 105
108  A Taste of UNIX and C                1(4wk)    MWF   3:35 Olin 255
         Glade                            su
```

Figure 12: A Schedule of the Introductory Computer Science Courses

Now consider Figure 12. This table lacks the lexical regularity of Table 11, because there are small irregularities in the lexical structure of the columns. To express this more rigorously, let M be a metric for string comparison (we use the Levenstein metric [31].) Given $\epsilon > 0$, two strings a and b are ϵ-similar if $M(a, b) \leq \epsilon$. We use ϵ_h-typings, defined below, of the regular expressions that correspond to the entries of a column in order to control the imperfections we allow in lexical structure.

Definition 3.4 Deviations in horizontal structure*: Given $\epsilon_h > 0$ and a set of strings, an ϵ_h-* typing *is a division of the set into disjoint subsets such that any two strings in the same subset are ϵ_h-similar.*

Lexical typing for a table is done in three parts. First, an ϵ_h-typing is performed on the candidate column elements. This generates an alphabet of types, to be used in the next step.

1. Form the WDG of the text block.
2. Find column separators of height (100 - ϵ_v)% blanks. If none exist, the text is not a table, so quit; otherwise continue.
3. Perform an ϵ_h-typing on the regular expression representations of the entries of each column.
4. Find the lexical structure these typings imply, if any. If such a structure is found, the text is a table; otherwise, it is not a table.

Figure 13: The Table Detection Algorithm

Then, each candidate column is analyzed to determine a regular expression for its type (over the newly formed type alphabet). Finally, the lexical type of the table is obtained by computing the minimum regular expression over the column types. The ϵ_h-typing in the first step allows for the occurrence of multi-line records in a table and for ϵ_h tolerance in the record units.[10].

Figure 13 describes the table detection algorithm. An example application of this algorithm to the table in Figure 12 follows. The first step in the algorithm is to create the WDG associated with it, by calculating the percentage of blank spaces in each column of the block. Figure 14 shows the WDG associated with the text in Figure 12. The second step is to look for column separators, *i.e.*, peaks in the graph of height at least $1 - \frac{\epsilon_v}{100}$. This graph has six high peaks, which are associated with the rivers of white space flowing between the seven columns in Figure 12. In the third step, each candidate column is analyzed for lexical structure. If the column description patterns can be combined into a regular expression across the entire table, the block of text is a table. Otherwise it is not.

3.1.2 Details and Analysis

We now analyze the robustness of the algorithm. Computing peaks in the WDG is quite easy. What is not obvious is how to determine a reasonable threshold value ϵ_v that robustly and efficiently filters columns from basic text. We measure efficiency as the cost of the actual computation, and the probability that base text is passed through unnecessary lexical analysis. One approach is to require the user to specify the value of ϵ_v using his knowledge about the data environment. Another approach is to have the algorithm statistically learn the value of ϵ_v by analyzing the WDG of tables identified by the user. A third solution does not rely on user assistance, but rather makes use of a probabilistic analysis of WDGs of basic text. The question we ask is: for a high peak value in the WDG, what is the probability that it corresponds to a true table column rather than a random distribution of spaces in basic-text? From this analysis, we extract a tolerance parameter that can be used as an absolute lower bound on ϵ_v for detecting tables with irregularities in layout.

The analysis makes the following assumptions:

[10] A minimal ϵ_h typing partitions the elements of the column in the coarsest possible way; unfortunately, as shown in 3.1.2, finding such a typing is NP-complete. We instead find a useful typing that may not be minimal, as described in 3.1.2

- The average word length that occurs in text is known. For English, [17] have determined that the average word length of distinct words is 8.1 characters, but of word occurrences in written text, it is 4.7 characters. For simplicity, we assume that in basic text, the average word length is 4 characters.

- The blank spaces in base text are distributed independently. This is due to the fact that the lengths of words and of the spacing between them are variable, and their occurrences in a line of text are random. We have tested the independence of the distribution of white space by extensive experiments with *Splus* [20]. This implies that the blank spaces of a line have a binomial distribution.[11]

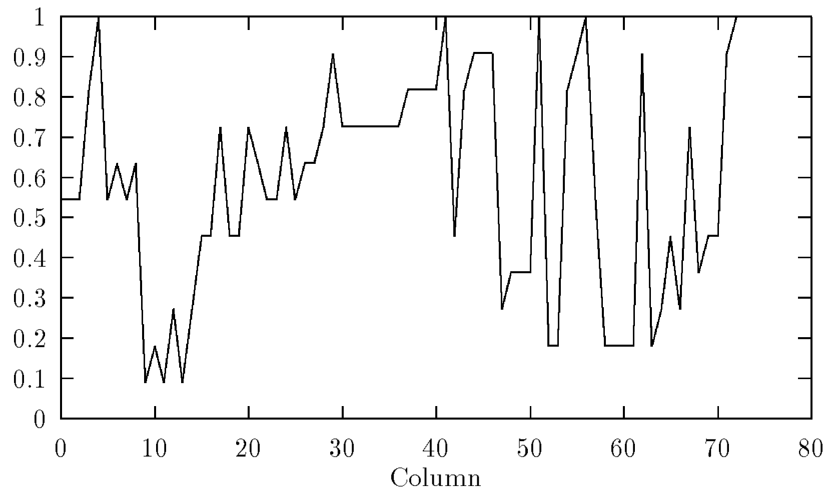


Figure 14: A White Space Density Graph for Table 2

Let B be a block of text of n rows and m columns. Let p be the probability that a character c in a row is blank, and let $q = 1 - p$ be the probability that the character is non-blank.[12] Let WDG be the white space density graph for B. Denote by WDG(k) the value for the k^{th} column of B. Application of Chebyshev's theorem[13] yields

Corollary 3.1 *If the absolute value of a peak in the* WDG *is greater than* $np + h\sqrt{npq}$, *the probability that the peak is an occurrence of base-text is* $\frac{1}{h^2}$.

In other words, by setting the peak threshold to $np + h\sqrt{npq}$, with probability $\frac{1}{h^2}$ the presence of any value above the threshold is a candidate column separator in a table. The

[11] An interesting problem is to determine when the binomial distribution approaches a normal distribution; at present, we have no solution to this problem.

[12] An average word length of 4 characters yields $p = 0.2$ and $q = 0.8$.

[13] For any distribution with standard deviation σ, at least a fraction $1 - \frac{1}{h^2}$ of the measurements differ from the mean by amounts at most $h\sigma$.

user can specify required confidence in identifying columns ($\frac{1}{h^2}$), and we can calculate the peak threshold, since n, p, q and h are all known.

Recall that the a block of text is a table if it has both vertical and horizontal structure. We now consider the complexity of lexical component analysis.

Theorem 3.2 *Finding the minimum ϵ-typing is NP-complete.*

Proof: Reduction to partitions into cliques. □

Although finding the minimum typing is a hard problem, a useful ϵ-typing can be found efficiently. An element is placed in a partition only if it is $\frac{\epsilon}{2}$-similar to the original element of that partition; when an element is encountered for which no such partition exists, it becomes the original element of a new partition.

The types of the entries of each column are assembled into the *type matrix.* A $m \times n$ type matrix is constructed for a block of text of m lines and n columns. If $t_{ij} = t_{i'j'}$ the data in row i and column j and the data in row i' and column j' are ϵ-similar. The type matrix for the table in Figure 12 is given in Figure 15. A GCD algorithm [16] can be used to determine the type, if any, of the overall matrix and thus to decide whether the matrix represents a table.

This efficient and robust table detector has been implemented and used to build information agents for retrieval tasks whose answers are found in tabular form [19].[14]

$$\begin{bmatrix} t_1 & t_2 & t_4 & t_6 & t_8 & t_9 & t_{10} \\ t_0 & t_3 & t_0 & t_6 & t_8 & t_9 & t_{10} \\ t_1 & t_2 & t_4 & t_6 & t_8 & t_9 & t_{10} \\ t_0 & t_3 & t_0 & t_6 & t_8 & t_9 & t_{10} \\ t_1 & t_2 & t_4 & t_0 & t_8 & t_9 & t_{10} \\ t_0 & t_3 & t_0 & t_0 & t_0 & t_0 & t_0 \\ t_1 & t_2 & t_4 & t_6 & t_8 & t_9 & t_{10} \\ t_0 & t_2 & t_0 & t_6 & t_8 & t_9 & t_{10} \\ t_1 & t_2 & t_4 & t_6 & t_8 & t_9 & t_{10} \\ t_0 & t_3 & t_5 & t_0 & t_0 & t_0 & t_0 \end{bmatrix}$$

Figure 15: Type Matrix For Table 2

3.2 A Classifier for Line Drawings

The white space density graph can be used to determine far more about a document component than whether it is arranged in columns. In particular, graphs of functions, convex polygons, and figures made up of multiple convex polygons all generate identifiable types of white space density graphs. Moreover, with respect to the WDG, closed curves and polygons lie in the same equivalence class.

Detecting polygonal structures has several applications. Given representations of the figures in a document, their WDGs can be generated, and the characterizations of these types of graphs

[14] Many thanks are due to Christopher Lewis and Matthew Scott for providing this implementation.

can be used to gain information about the figures. This would enable a user to query a system based on a description of relevant figures. For example, a desired paper could be retrieved based on the recalled location of a function graph, or the section of a paper recalled to contain a convex polygon could be retrieved. This usage presumes prior knowledge of the portions of the document that constitute figures. In many cases, such an assumption is reasonable. For example, in a PostScript document representation, identifiable commands indicate the inclusion of figures; similarly in the XDOC format, generated by the ScanWorX optical character recognition software [6], figures are separated from text and represented differently. Nonetheless, this distinction may not always be readily available. An ASCII text file, such as an electronic news or mail message, will not provide such convenient separations. In this case characterizations of WDGs may aid in the attempt to distinguish figures from text; a portion of the document is especially likely to be a figure if its WDG has a shape that may describe a function, convex polygon, or combination of convex polygons.

Theorem 3.3 *For every closed curve there exists a polygon with the same number of line crossings and the same* WDG.

Proof: By construction. □

For example, consider the sketch of a snowman in Figure 16. Its WDG is identical to that of the polygon in Figure 16, and the two contain the same number of line crossings. In essence, the curves in the original figure have simply been "straightened" into lines. The WDG for these figures is given in Figure 17.[15]

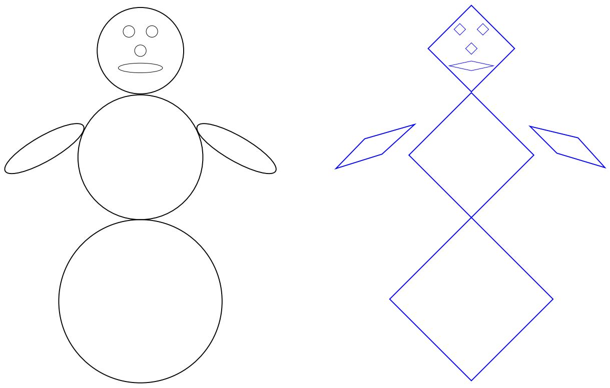


Figure 16: A curved figure and its polygonal equivalent

The WDG's of this class of figures have special geometry that can be characterized in terms of their heights and slopes. An important special case is called a *plateau*.

Definition 3.5 *A segment of a* WDG *is a* plateau *if it has slope* 0.

Then, if the correct adjustments are made for the possible inclusion of axes and possible white space adjacent to the figure on the left and right, the following characterizations hold, so long as the granularity of the figure representation is sufficiently fine. The block under consideration is presumed to have been divided into n lines.

[15] The block containing exactly the figure has been divided into 20 lines of 40 characters.

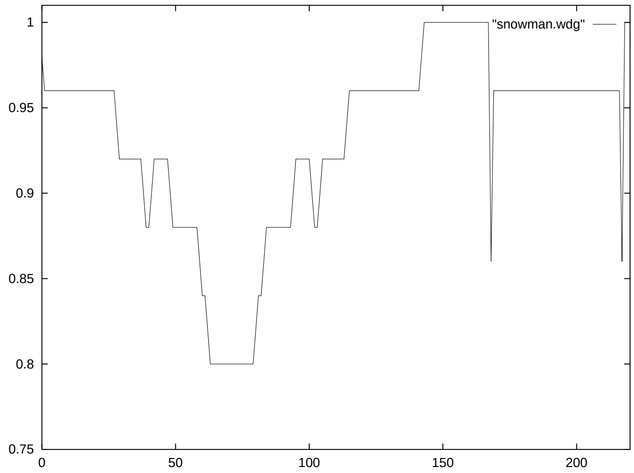


Figure 17: The WDG for the preceding figures

Theorem 3.4

- *The* WDG *of a continuous function consists of a plateau of height* $\frac{n-1}{n}$.
- *The* WDG *of a discontinuous function whose only discontinuities occur in its domain includes only heights of* 1 *and heights of* $\frac{n-1}{n}$.
- *The* WDG *of a convex polygon consists of two arbitrary endpoints, with a plateau of height* $\frac{n-2}{n}$ *between them.*
- *The* WDG *of a figure made up of convex polygons consists of a series of plateaux separated by transitions of length* 2. *The heights of the plateaux differ by multiples of* $\frac{2}{n}$.

For example, consider Figure 17, the WDG for Figure 16. In this case, $n = 20$. The graph consists of 5 plateaux, separated by transitions of length 2. The height of each plateau differs from the height of its neighbors by .1, *i.e.*, $\frac{2}{20}$.

3.3 A Classifier for Itemized Lists

By itemized lists, we mean lists of the standard form generated by LaTeX list environments. Such lists have an easily characterized indentation structure: the first line is indented from the margin, and all subsequent lines are indented with respect to the first line but left-justified with respect to each other. This description yields the specification for the text elements of the pattern: *Left-In, Left-In, Left-Justified**. (The blank elements can be ignored, in this case.) The algorithm in Section 2 finds such patterns, and we characterize them as lists.

For example, consider again the first section of Figure 2. The early steps of the application of the layout hierarchy generation algorithm to this document portion appear in Subsection 2.4. In the first step, its lists are identified as patterns. The text portions of the patterns found are *Left-In, Left-In* and *Left-In, Left-In, Left-Justified*. These both match the above specification, so they are characterized as lists.

3.4 Future Extensions

The classification of layout hierarchy nodes requires additional exploration in at least the following areas.

- We would like to find a general characterization of the kinds of structures that can be identified using the WDG, as well as those that can be identified using any kind of white space aggregate. For structures of these types, we will seek algorithms that in fact rely on white space; for others, we will need to represent more complex information and consider symbols as well.
- The structures we classify at present are characterized by complete descriptions that must be precisely matched. Ideally, we envision a set of partial descriptions that may be matched by unforeseen patterns; classification would be performed by comparing the observed patterns to these partial descriptions. Some elements of this process will, of necessity, be heuristic. For example, distinguishing between a document's title and its author is not always possible without extensive cultural knowledge.[16] Formal investigation of the types of descriptions that can and should be provided is greatly needed.

4 Experiments

In this section we describe our implementation of the previous algorithms and give performance evaluations. Our data consists of the electronic library of computer science technical report at Cornell University, as well as the domain of newsgroups. The technical reports are represented as scanned-in bitmaps corresponding to pages. We discuss the performance of the segmentation algorithm on a random subset of 400 technical report pages. Finally, we present a sophisticated information access application that would be impossible without the tools discussed here.

4.1 Segmentation Experiments

4.1.1 The System

The system we use to segment technical report pages functions in the following manner. The pages are originally scanned in and represented as bitmaps. We then perform OCR on the pages, using the Xerox ScanWorX software, which returns a file in the XDOC format. We use a perl script to parse this file and return a file with the information needed by our algorithm[17]: the page is divided into lines, the left and right edges of each line (with respect to the page margins) are given, and the height of the white space between lines is provided. (This space is considered significant only if it is greater than the page's minimum; thus the space between ordinary lines in a paragraph, for example, is not considered.) Then, the segmentation algorithm

[16] Titles are usually printed in a larger size than authors' names or printed above them, but consider the novel *Daniel Deronda* by George Eliot. The cover to the paperback Oxford edition [9] contains two lines: one with the author's name, and one with the title. They are printed in the same font, and George Eliot's name appears above her novel's title.

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

(implemented in Lisp) is run on this file.[18] The output is a file that specifies the lines that are grouped together as blocks at each level of the tree. This file, together with one that specifies line positions, is read by a tcl script, which then displays the results.[19]

4.2 The Results

We evaluated the algorithm's performance by looking at the blocks formed on each page and comparing these line groupings to the groupings that a human being considers logical. We counted the number of good (*i.e.*, logical) groupings, the number of bad groupings in the results, and the number of missing groupings (*i.e.*, those groupings that we considered to be required for a logical segmentation and were not in the results), keeping track of *input* errors that were due to ScanWorX or the XDOC parser's interpretation of its information and those that were due to the algorithm. Since the algorithm works in a bottom-up fashion, errors at lower levels often generate errors at higher levels, and we separated these *propagated* errors from the *original* errors.

In evaluating the performance and errors, we rely on the ideas of precision and recall from information retrieval. In the realm of IR, these are defined as follows[28]:

$$Precision = \frac{Relevant\ retrieved}{Total\ retrieved}$$

$$Recall = \frac{Relevant\ retrieved}{Total\ relevant}$$

We analogize logical groupings to relevant documents and formed groupings to retrieved documents. This gives us:

$$Precision_{segment_{original}} = \frac{Good\ groups}{Good\ groups + Bad\ groups}$$

$$Recall_{segment_{original}} = \frac{Good\ groups}{Good\ groups + Missing\ groups}$$

These definitions specify a gross measure of precision and recall. They do not, however, take into account the difference between an original error and a propagated error. Since errors may be propagated by good behavior, this amounts to counting original errors multiple times. A more accurate and fair measure discards all propagated errors and all input errors. To understand the problem with propagated errors, suppose the final paragraph on a page is not recognized as a group because of an algorithmic failure. A propagated error might be to group all the other paragraphs that belong to its section together. This would be correct, if the final paragraph did not exist. We do not, of course, count these cases as good groupings. We also do not count them as incorrect (bad or missing); we simply remove them from consideration. The following measures reflect these observations:

[18]The version of the algorithm we use is a slight variation on what we have presented. It includes pattern elements as blocks at a level of their own, and it includes two heuristics for handling common special cases: a sequence of centered lines at the top of a page is presumed to be a title and is consequently grouped by vertical distance, and non-classifiable blocks surrounded by white space are merged into surrounding classifiable blocks. (The latter handles cases like the inclusion of an equation in the middle of a paragraph.)

[19]Again, thanks to Jim Davis for providing the basis for this script.

$$Precision_{segment} = \frac{Good\ groups}{Good\ groups + Bad\ groups - input\ bad - Propagated\ bad}$$

$$Recall_{segment} = \frac{Good\ groups}{Good\ groups + Missing\ groups - input\ missing - Propagated\ missing}$$

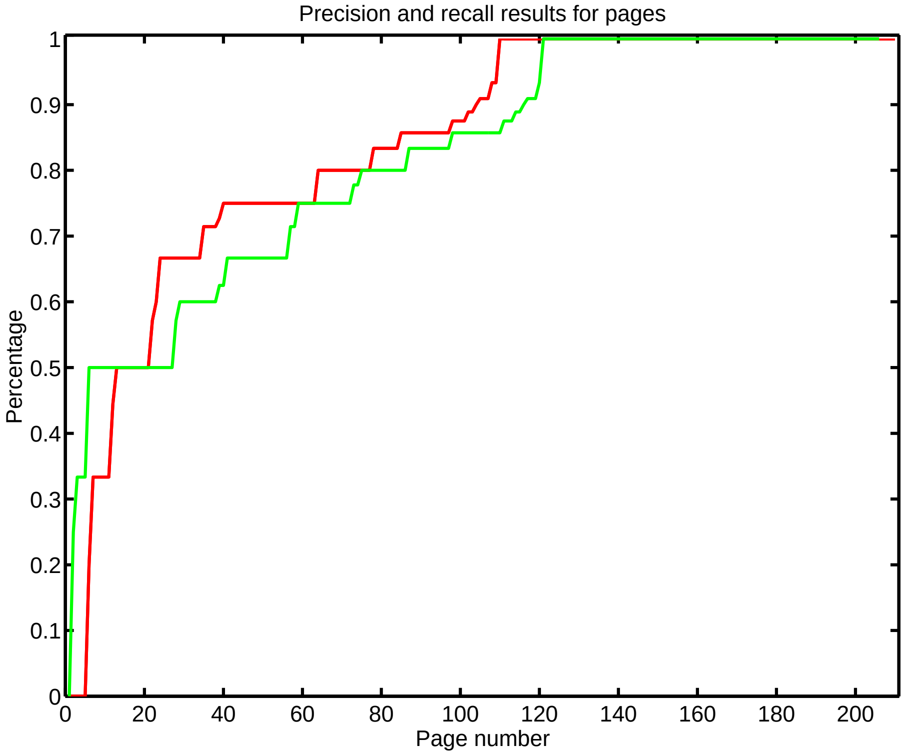


Figure 18: The red graph shows the (sorted) precision statistics for a subset of the pages we analyzed. The green graph shows the recall statistics for the same subset of pages.

Results are shown in Figure 18 and Figure 19. We obtain over 75% precision and 65% recall for approximately 75% of our pages; this includes 100% precision on over 45% and 100% recall on over 40% of the pages. Overall, we obtain over 80% precision and 70% recall for 12 of our 15 technical reports. We reach these results without considering block classification; as the approach to classification matures, its results can be used to modify the segmentation results. This should improve performance substantially. We discuss the most common error types and their implications for future work below.

Two sample pages are shown in Figure 20; their results are shown in Figures 21 and 22. In the first example, a list is recognized and grouped, while sections are kept apart. The second example illustrates errors due to element length differences; the two-line paragraph is not recognized as part of a pattern, causing the algorithm to halt too early.

A great many of the errors we found fall into a few general categories. These point the way for future work.

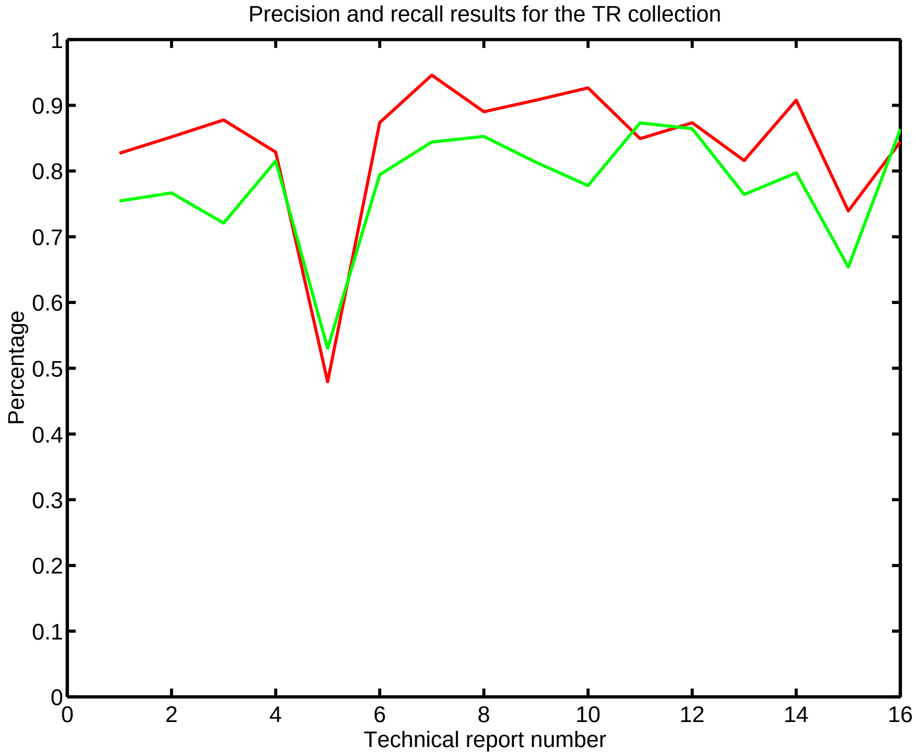


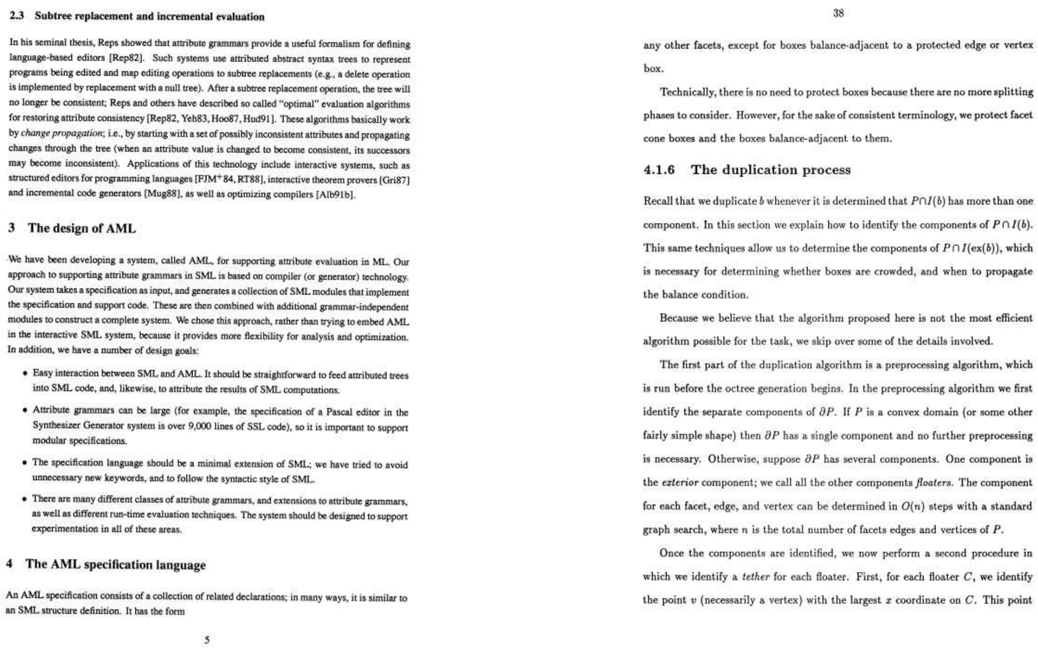
Figure 19: The red graph shows the precision statistics for each of the technical reports we analyzed. The green graph shows the recall statistics for each of the technical reports we analyzed.

- *Isolated structures* This occurs most often with paragraphs. Paragraphs are recognized as elements of a repeating pattern, when two or more appear together. Sometimes, however, a page contains a paragraph surrounded by other kinds of blocks, such as left-justified theorems or headings. Since a standard paragraph contains no repetition, an isolated one is not found as a structure.

  This kind of special case requires some handling; perhaps a frequently occurring structure in a document should be recognized even when it is not currently part of a pattern. Also note that this case occurs more often in our page-by-page application than it would in an application applied to whole documents. Often, a paragraph is part of a pattern, but the rest of the pattern appears on the following page.

- *Inconsistent element sizes* This occurs primarily with paragraphs and lists items. Our indentation alphabet distinguishes between instances of these elements that have one line, instances with two lines, and instances with more than two lines.[20] We would like to find an approach to determining when these kinds of observed differences should be considered significant.

[20] The difference between instances with two lines and with more than two comes from the inclusion of the *Left-Justified* character, as in the second list of our miniature document example.



2.3 Subtree replacement and incremental evaluation

In his seminal thesis, Reps showed that attribute grammars provide a useful formalism for defining language-based editors [Rep82]. Such systems use attributed abstract syntax trees to represent programs being edited and map editing operations to subtree replacements (e.g., a delete operation is implemented by replacement with a null tree). After a subtree replacement operation, the tree will no longer be consistent; Reps and others have described so called "optimal" evaluation algorithms for restoring attribute consistency [Rep82, Yeh83, Hoo87, Hud91]. These algorithms basically work by *change propagation*; i.e., by starting with a set of possibly inconsistent attributes and propagating changes through the tree (when an attribute value is changed to become consistent, its successors may become inconsistent). Applications of this technology include interactive systems, such as structured editors for programming languages [FJM$^+$84, RT88], interactive theorem provers [Gri87] and incremental code generators [Mug88], as well as optimizing compilers [Alb91b].

3 The design of AML

We have been developing a system, called AML, for supporting attribute evaluation in ML. Our approach to supporting attribute grammars in SML is based on compiler (or generator) technology. Our system takes a specification as input, and generates a collection of SML modules that implement the specification and support code. These are then combined with additional grammar-independent modules to construct a complete system. We chose this approach, rather than trying to embed AML in the interactive SML system, because it provides more flexibility for analysis and optimization. In addition, we have a number of design goals:

- Easy interaction between SML and AML. It should be straightforward to feed attributed trees into SML code, and, likewise, to attribute the results of SML computations.
- Attribute grammars can be large (for example, the specification of a Pascal editor in the Synthesizer Generator system is over 9,000 lines of SSL code), so it is important to support modular specifications.
- The specification language should be a minimal extension of SML; we have tried to avoid unnecessary new keywords, and to follow the syntactic style of SML.
- There are many different classes of attribute grammars, and extensions to attribute grammars, as well as different run-time evaluation techniques. The system should be designed to support experimentation in all of these areas.

4 The AML specification language

An AML specification consists of a collection of related declarations; in many ways, it is similar to an SML structure definition. It has the form

5

38

any other facets, except for boxes balance-adjacent to a protected edge or vertex box.

Technically, there is no need to protect boxes because there are no more splitting phases to consider. However, for the sake of consistent terminology, we protect facet cone boxes and the boxes balance-adjacent to them.

4.1.6 The duplication process

Recall that we duplicate b whenever it is determined that $P \cap I(b)$ has more than one component. In this section we explain how to identify the components of $P \cap I(b)$. This same techniques allow us to determine the components of $P \cap I(\mathrm{ex}(b))$, which is necessary for determining whether boxes are crowded, and when to propagate the balance condition.

Because we believe that the algorithm proposed here is not the most efficient algorithm possible for the task, we skip over some of the details involved.

The first part of the duplication algorithm is a preprocessing algorithm, which is run before the octree generation begins. In the preprocessing algorithm we first identify the separate components of ∂P. If P is a convex domain (or some other fairly simple shape) then ∂P has a single component and no further preprocessing is necessary. Otherwise, suppose ∂P has several components. One component is the *exterior* component; we call all the other components *floaters*. The component for each facet, edge, and vertex can be determined in $O(n)$ steps with a standard graph search, where n is the total number of facets edges and vertices of P.

Once the components are identified, we now perform a second procedure in which we identify a *tether* for each floater. First, for each floater C, we identify the point v (necessarily a vertex) with the largest x coordinate on C. This point

Figure 20: Two sample pages. The left page is grouped correctly by the algorithm. The right page is not.

- *Hanging indents* In a page like a table of contents or some bibliographies, an indentation style is used that is the reverse of the usual paragraph indentation. When the entries are separated by blank lines, this does not pose a problem. When the entries are not so separated, the algorithm finds the wrong pattern. This is a limitation of the problem as we have posed it; with our set of observables, a page of hanging indents without blank lines is not distinguishable from a page of paragraphs (many of which may be one line long), beginning with some left-justified structure. Our indentation alphabet was chosen to prefer the latter interpretation, as this phenomenon is more common.

- *Spacing* At times, the requirement that vertical white space be of equal height in pattern elements can prevent good patterns from being formed. For example, if elements of a list are found at different times (due to the size issue above), forming two or more list pieces, these should be combined. They are not combined, however, if the spacing within the list differs from the spacing surrounding the list.

  At other times, our consideration of spacing is insufficient to prevent bad groupings. This occurs because document layouts do not reserve one spacing distance for each kind of structure. For example, the spacing after a figure caption is often the same as the spacing after a heading. Thus, a left-justified caption and a left-justified heading are grouped together.

  The information to be gleaned from the height of white space seems to depend on more

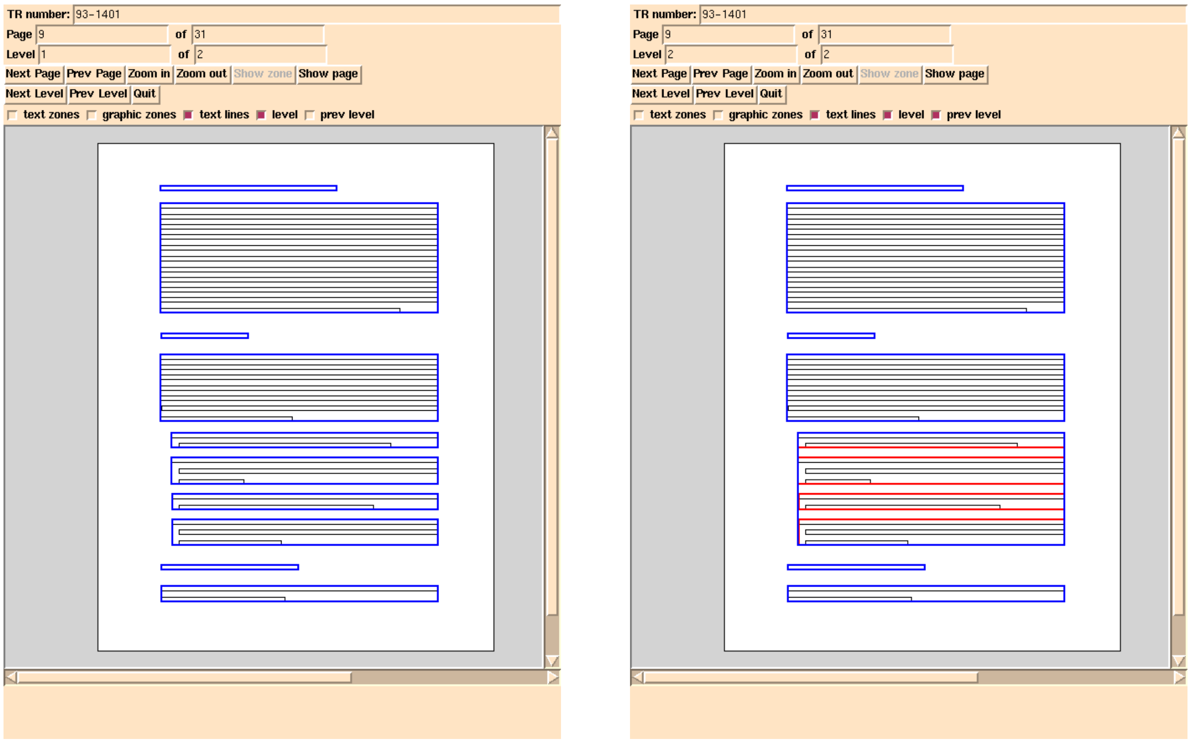


Figure 21: Correct groups for the left page of the prior figure. The left side shows level 1. The right side shows level 2 in blue, with the divisions from level 1 in red.

context than our indentation alphabet considers. Characterizing this information and the relevant context warrants further investigation.

4.3 Application to Information Access

We now illustrate the use of automated segmentation and classification of non-textual components in information access. Starting from the data in the CS-TR collection, the goal is to support queries that are based on any non-textual structural unit that we can detect, along with keywords.

Imagine reading a FOCS paper that contains some cryptic definition. Starting from this definition, we would like to get a different version of the paper that explains the definition and perhaps even has figures illustrating the definition. This process can be automated by using a combination of algorithms for segmenting, classifying, indexing, and hyperlinking. We are able to execute this task with the following sequence of operations.

Before a query comes in, each document is preprocessed by: (1) segmentation, to highlight the layout components; (2) classification, to associate types with the noded from the tree; and (3) using the nodes of the tree, the document can be automatically marked-up in an SGML-like fashion. We have implemented this by using our segmentation and classification algorithms

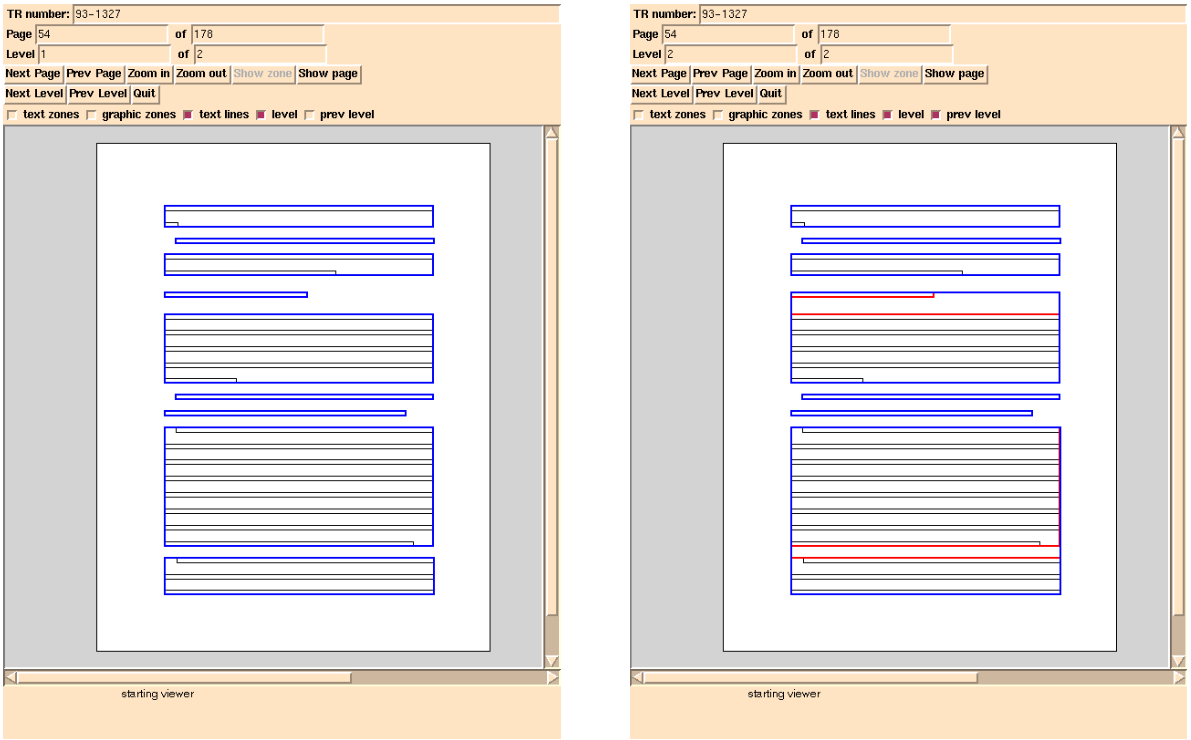


Figure 22: The groups for the right page of the earlier figure, with errors caused by the two-line paragraph. The left side shows level 1. The right side shows level 2 in blue, with the divisions from level 1 in red.

on postscript file and by using the mark-up required by the Smart system for information retrieval[21] [28]. By restricting ourselves to the postscript files rather than generic, scanned-in images, we are able to label fully the nodes of our tree. That is, we are able to classify nodes as *sections, subsections, theorems, proofs, definitions, figures, figure captions, etc.* We achieve this by examining a combination of spacing and expected keywords. Although this implementation is not general, it illustrates the power of the approach we advocate for information access.

Preprocessing allows us to refer to words, as well as paragraphs, lists, tables, and other layout structures directly in a query. When a query comes in, all the layout structures from the documents being searched are sent to Smart as a collection of mini-documents. The system is then used to compute the hyperlinks between layout pieces. Hyperlinks between the figures in the collection of documents are computed using the Hausdorff matching algorithm [**?**].

Figure 23 shows the results of this process to automatically find related pieces inside a document. The document is represented as a circular arc. Each layout piece at the granularity level of a paragraph is represented as a block within this circular arc. In figure 23 only the blocks that are found to be relevant are shown. Starting with a figure caption, the task is to find

[21] Smart indexes documents by keywords and uses a vector space model for retrieval.

everything related to the figure within the document. This query is processed by organizing the layout pieces as a collection of documents. This collection is processed by the Smart system to find a related figure caption, some text describing it, as well as a theorem referring to the figure, its proof, and an application.

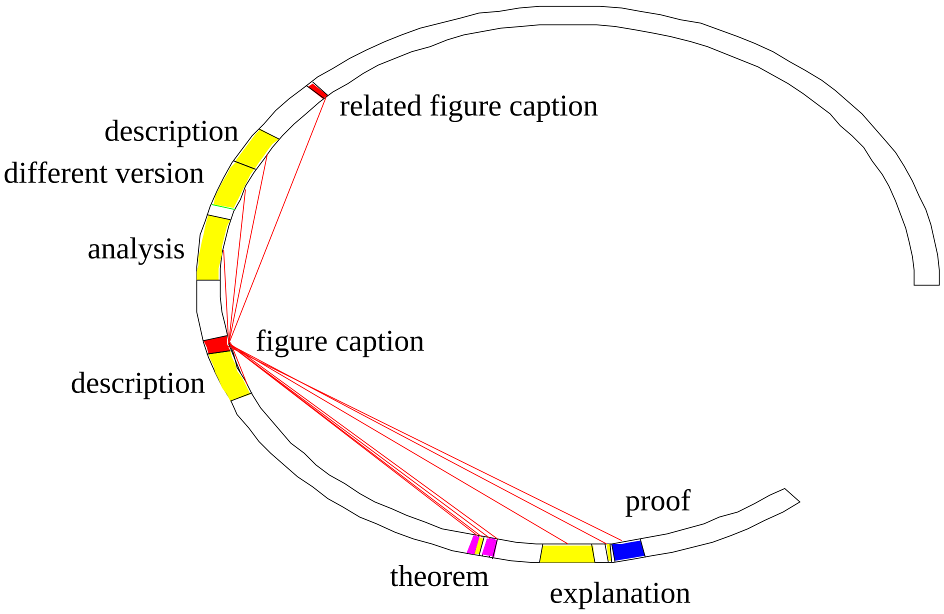


Figure 23: Structural hyperlinking for centralized information access. The circular arc represents a document. Various of its layout pieces are highlighted and edges are drawn between related parts.

The example in Figure 23 is centralized, but nothing prevents it from working on a distributed collection. The obstacles in solving the FOCS query described in the beginning of this section are minor. We are currently working on building the necessary extensions.

5 Discussion

Our research agenda is to develop and prototype a methodology for conceptual retrieval tasks in large, heterogeneous, distributed electronic libraries. We consider the key question to be how to associate information with content. The natural language problem is still unsolved, but we can call upon other computational techniques to make progress. We believe that, in the environments we consider, content is partially encoded in extant underlying structures, which are found at varying levels of granularity. Hence, we seek to automatically discover structural hierarchies in documents. We believe that by developing efficient tools that allow users to specify tasks in visual terms we provide computational support for a search technique that people use naturally. At the same time, this approach enlarges the class of retrieval tasks supported by keyword-based systems.

The specific problem addressed in this paper is the characterization of the logical structure of documents in terms of high-level geometric layout components. We have proposed a method for using the indentation contours to construct hierarchical layout representations and algorithms for classifying some of the nodes in the hierarchy. We have not solved the general problem of automatic document structuring, but we have obtained insights into how a simple notion of layout (namely, indentation) can be formalized and used to capture geometric content.

The algorithms presented in this paper have implications for (1) creating rich, user-friendly browsing tools based on the automatic and on-line creation of indices and hyperlinks, *e.g.*, customizing the browsing of information by providing on-line automatic support for computing a variety of views; (2) distributed access of remote electronic repositories, by providing partial models of documents that essentially "compress" the data at a various levels of detail; (3) conceptualizing the search of electronic documents by providing support for retrieval with non-linguistic cues, an approach supported by our belief that the inherent structures in electronic repositories encode content in a task-specific way [26] and illustrated by examples throughout this paper and in [1, 26]; and (4) uniformly marking-up generic documents for other retrieval purposes (for instance, an SGML mark-up for the Smart retrieval system [30]) by automatically recognizing the logical components of documents. This last would allow users the flexibility to create documents in the format of their choice; these could be automatically integrated with retrieval systems (like the Smart system) that usually rely on mark-up for indexing.

Acknowledgments

We are very grateful to John Hopcroft for proposing the problem and for his guidance and support. We thank Allen Back for many enlightening discussions on probability theory. Special thanks to James Allan, Jim Davis, Bruce Donald, Carl Lagoze, Dean Krafft, Matthew Morgenstern, T.V. Raman, and Devika Subramanian for many invaluable suggestions. Thanks also to Christopher Lewis, Matthew Scott, and Amit Singhal for useful comments and support.

References

[1] J. Allan, J. Davis, D. Krafft, D. Rus, and D. Subramanian. Information agents for building hyperlinks, 1993.

[2] A. Bookstein. Fuzzy requests: An approach to weighted boolean searches. *Journal of the American Society for Information Science*, 30(4):240–247, 1980.

[3] A. L. Brown, S. Mantha, and T. Wakayama. A formal specification of document processing. Xerox Corporation, Webster Research Center.

[4] A. L. Brown, T. Wakayma, and H. A. Blair. A reconstruction of context-dependent document processing in SGML. Xerox Corporation, Webster Research Center and Syracuse University, School of Computer and Information Science.

[5] D. A. Buell and D. H. Kraft. Threshold values and boolean retrieval system. *Information Processing and Management*, 17(3):127–136, 1981.

[6] D. Conelly and B. Paddock. XDOC data format: Technical specification. Xerox Imaging Systems, Inc., Peabody, MA, 1993.

[7] W. B. Croft. Document representation in probablistic models in information retrieval. *Journal of the American Society for Information Science*, 31(6):451–457, 1981.

[8] D. Cutting, D. Karger, J. Pedersen, and J. Tukey. Scatter/gather: A cluster-based approach to browsing large document collections. In *Proceedings of the Fifteenth Annual International ACM/SIGIR Conference on Research and Development in Information Retrieval*, pages 318–329, Copenhagen, June 1992. ACM Press.

[9] G. Eliot. *Daniel Deronda*. The World's Classics. Oxford University Press, Oxford, 1984.

[10] A. Feng and T. Wakayama. SIMON: A grammar based transformation system for structured documents. Xerox Corporation, Webster Research Center.

[11] 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.

[12] Charles F. Goldfarb. *The SGML Handbook*. Clarendon Press, Oxford, 1990.

[13] M. A. Hearst. Contextualizing retrieval of full-length documents. Technical Report UCB/CSD-94-789, University of California, Berkeley, 1994.

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

[15] B. Kahle. Overview of wide area information servers, 1991. WAIS on-line documentation.

[16] D. Kozen. *The Design and Analysis of Algorithms*. Springer-Verlag, New York, 1992.

[17] H. Kucera and W. Francis. *Computational Analysis of Presentday American English.* Brown University Press, Providence, RI, 1967.

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

[19] C. Lewis, D. Rus, and M. Scott. A structure detector for tables. Forthcoming Technical Report.

[20] Splus Reference Manual. Statistical Sciences, Inc., 1991.

[21] M. Mizuno, Y. Tsuji, T. Tanaka, H. Tanaka, M. Iwashita, and T. Temma. Document recognition system with layout structure generator. *NEC Research and Development*, 32(2):430–437, July 1991.

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

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

[24] M. A. Rahgozar, Z. Fan, and E. V. Rainero. Tabular document recognition. In *SPIE Proceedings*, San Jose, February 1994.

[25] D. Rus and D. Subramanian. Information retrieval, information structure, and information agents. Submitted to the *ACM Transactions on Information Systems*, 1993.

[26] D. Rus and D. Subramanian. Multi-media RISSC Informatics: Retrieving Information with Simple Structural Components. In *Proceedings of the ACM Conference on Information and Knowledge Management*, Washington DC, November 1993.

[27] O. Sacks. *The Man Who Mistook His Wife for a Hat and Other Clinical Tales*, chapter 9, pages 80–84. Harper & Row, New York, Perennial Library edition, 1987.

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

[29] G. Salton and C. Buckley. Improving retrieval performance by relevance feedback. *Journal of American Society for Information Science*, 41(4):288–297, 1990.

[30] G. Salton and M. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, New York, 1983.

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

[32] S. Tsujimoto and H. Asada. Major components of a complete text reading system. In *Proceedings of the IEEE*, volume 80, 1992.

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