Layout Understanding

A Knowledge Based Approach

by Bagepalli C. Krishna

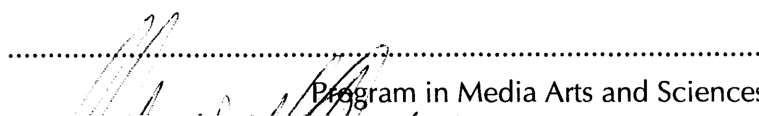
M.Sc. Computer Science
Birla Institute of Technology and Science
Pilani, Rajasthan, India
December 1981

Submitted to the Program in Media Arts and Sciences
School of Architecture and Planning
In partial fulfillment of the requirements for the degree of
Master of Science in Media Arts and Sciences

at the Massachusetts Institute of Technology
February 1994

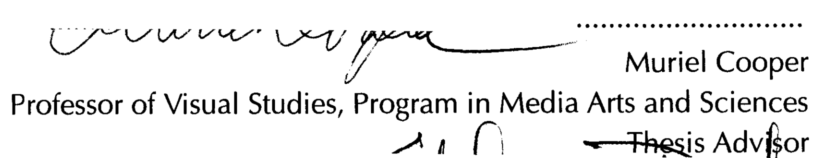
Copyright 1994 MIT All Rights Reserved

Signature of Author ....



Program in Media Arts and Sciences
January 14 1994

Certified by



Muriel Cooper
Professor of Visual Studies, Program in Media Arts and Sciences
Thesis Advisor

Accepted by ....

Stephen A. Benton
Chairperson, Departmental Committee on Graduate Students
Program in Media Arts and Sciences

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

FEB 23 1994

LIBRARIES

Layout Understanding

A Knowledge Based Approach

by Bagepalli C. Krishna

Submitted to the Program in Media Arts and Sciences
School of Architecture and Planning
On January 14 1994
In partial fulfillment of the requirements for the degree of
Master of Science in Media Arts and Sciences

at the Massachusetts Institute of Technology

Abstract

The layout of information is a complex task, and is usually entrusted to a professional graphic designer. Research into automatic layout attempts to develop computational models of problem solving techniques used by graphic designers to solve layout problems. Graphic designers develop their problem solving skills through years of learning by following three approaches: i) learning by being told, ii) learning by example, and iii) learning by doing. Each form of learning involves the development of internal representations of that which is learned. This research seeks to develop computer-based knowledge representations of the result of the learning experiences from each of the three approaches, with the eventual goal of developing a methodology to build page layout understanding systems.

In this research, understanding is defined as the task of automatically creating a high-level, structured representation of the layout from a structurally poor representation. As part of this research, we have built an experimental system which automatically converts a structurally poor representation of an existing layout — an image obtained by optically scanning an existing layout — into a high-level, structured representation — the input layout is encoded as a set of constraint networks that encapsulate symbolic relationships between the elements in the layout.

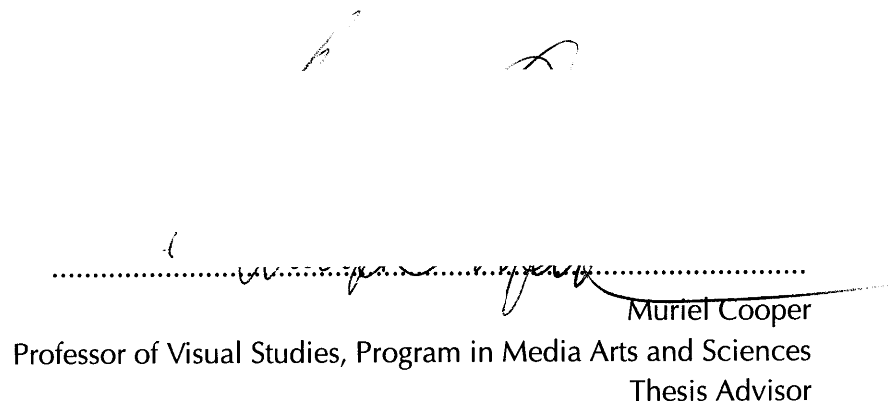
Thesis Advisor: Prof. Muriel Cooper
Professor of Visual Studies, Program in Media Arts and Sciences
This work was supported in part by Digital Equipment Corporation

Layout Understanding

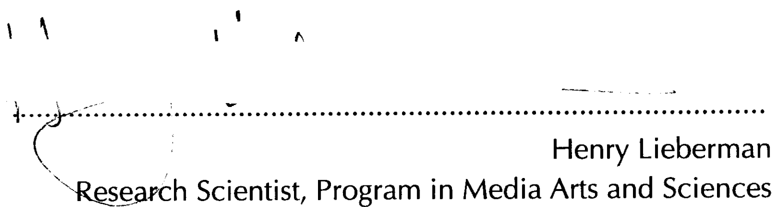
A Knowledge Based Approach

by Bagepalli C. Krishna

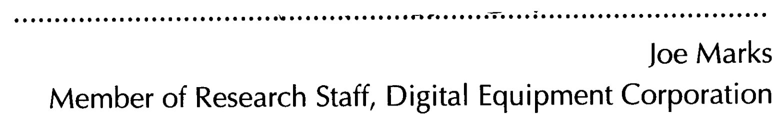
The following people have served as readers for this thesis.



Muriel Cooper
Professor of Visual Studies, Program in Media Arts and Sciences
Thesis Advisor



Henry Lieberman
Research Scientist, Program in Media Arts and Sciences



Joe Marks
Member of Research Staff, Digital Equipment Corporation

Acknowledgments

To my parents, who taught me to value education above all else.

To Muriel and Ron, for creating the Visible Language Workshop — where I'd rather be than any other place.

To my family — Geeta, Nikhil and Tara — for putting up with my idiosyncratic behavior over these past two years.

To my thesis readers, Alan Borning, Henry Lieberman, Joe Marks and Dick Rubinstein, for guidance, and for the many insightful comments.

To David Small, for initially suggesting that the grid would be a useful thing to infer. And a million other things, not the least of which is ensuring that things in the VLW work.

To Suguru Ishizaki, for being patient, for the many discussions, and for helping me *think* about design. Suguru, may you never be far away from the perfect *daal fry*!

To Tom Briggs, for being a design teacher, and for useful discussions during the initial stages of the project.

To Mahesh Viswanathan and Sharad Sheth, for pointing me in the right direction in my quest for Block Segmentation techniques.

To Louie Weitzman, for making me aware of Relational Grammars.

To all the folks at Digital Equipment Corporation who made my graduate studies possible — Seth Cohen, Peter Davis, Sue Gault, Branko Gerovac, Gene Morgan, John Morse, Lee Peterson, Terry Sarandrea, Shapoor Shayan and Howard Webber.

To Linda Peterson and Santina Tonelli, for constantly reminding me that there was, indeed, a method to the madness.

And to the many VLWers, who, in the tradition of Mrs. Field's Cookies, were always "open for business and pleasure" — Grace Colby, Karen Donoghue, Maia Engeli, John Grabowski, Toshi Ichioka, Tetsuaki Isonishi, Craig Kanarick, Eddie Kohler, Robin Kullberg, Ishantha Lokuge, Earl Rennison, Lisa Strausfeld, Alan Turransky, Jeff Ventrella, Yin Yin Wong and Xiaoyang Yang.

Table of Contents

Chapter 1	Introduction	6
Chapter 2	Application Scenarios	11
Chapter 3	Philosophical Underpinnings	14
Chapter 4	Related Research	22
Chapter 5	Approach	29
Chapter 6	Implementation	37
Chapter 7	Conclusions	48
Chapter 8	Bibliography	50
Appendix I	Khoros	54
Appendix II	CLIPS	57
Appendix III	DeltaBlue	59
Appendix IV	Relational Predicates	61

Chapter 1

Introduction

The design of a page layout is the task of carefully arranging the given content of a book or a magazine in order to facilitate the communication of the overall message implied by the content. The design of a good layout is a complex task, and is usually entrusted to a professional graphic designer. The graphic designer satisfies the communication goals of the layout by using both creative and organizational skills, developed through years of training and experience.

Traditionally, designers have employed their skills to design for the static, print medium. However, these skills are equally applicable when designing for a computer-based digital medium. Designing for the digital medium increases the complexity of the design problem: i) the classes of content can now include dynamic information such as sound and video in addition to static information such as type and image, ii) the problem domain extends beyond books and magazines to graphical user interfaces, and iii) the designed artifact must now support user interaction.

The realities of design for the digital medium have placed additional demands on scarce design resources: high speed computers and high bandwidth information delivery channels generate and move vast quantities of information to environments where there is no designer. When design resources are available, lessons learnt from solving design problems do not feed back to enrich the body of theoretical knowledge of design for the digital medium. These realities point to the need to study the current theory and practice of design in order to develop theoretical models for the creative and organizational skills of a graphic designer. Such models will, in turn, help us build design support systems, and when appropriate, build systems that can be surrogates for the graphic designer.

The goal of this research is to enrich the body of knowledge of automatic layout design systems. The specific contribution of this project is the development of a methodology to *understand* existing layouts. In this research, *understanding* is defined as the task of automatically creating a high-level structured representation of the layout from structurally poor representation. In this thesis we describe our approach to layout understanding; we

also describe an experimental system that was built using this approach. This system automatically converts a structurally poor representation of a layout — an image bitmap obtained by optically scanning an existing layout — into a high-level structural representation — the input layout is encoded as a set of constraint networks that encapsulate symbolic relationships between the elements of the layout.

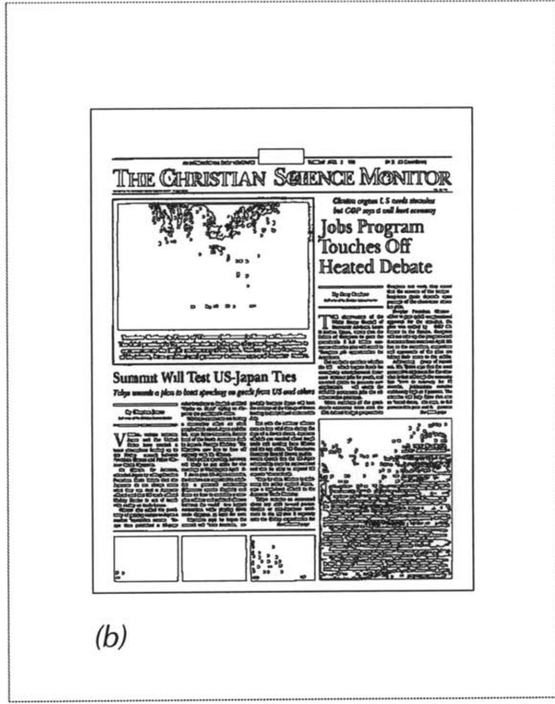
Although we have chosen an image bitmap as the representation for input layouts, it is easy to conceive of other structurally poor representations for input layouts: the layout can be roughly sketched into the computer system using a pen-like device, encoded in a page description language such as PostScript, acquired by means of a facsimile transmission, or created using desktop-publishing software such as Microsoft Word.

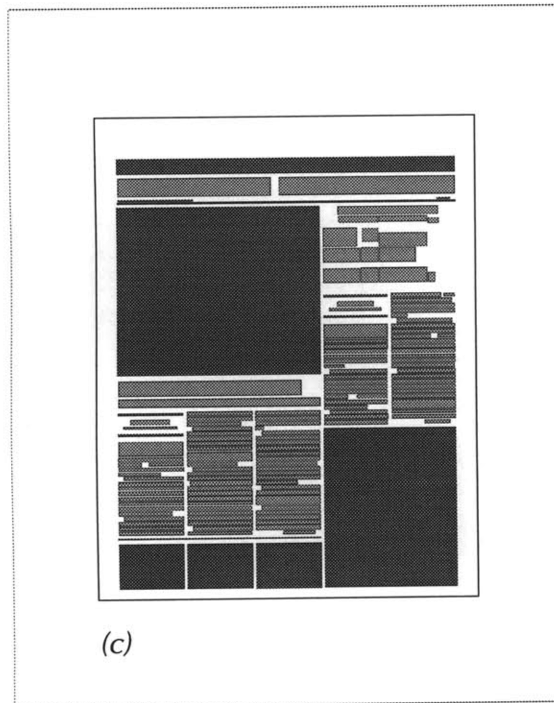
A layout represented in the form of a set of constraint networks can be used as a basis for automatic layout using case-based reasoning. In this approach, a case library of exemplar layouts is searched for an appropriate case, and the retrieved case is adapted to solve a new layout problem. Exemplar layouts encoded in relational form as a set of constraint networks are well suited for adaptation.

For example, a convention used to differentiate between a headline and a subheadline is to provide a visual contrast between them. This may take the form of a size relationship, such that the size of the subheadline is some percentage of the size of the headline. This relationship is an example of a constraint that must be satisfied. By encoding the layout in the form of a network of constraints, we can adapt the layout to a variety of different situations; by re-satisfying the constraints after adaptation, we are sure that the designer's intent is maintained. For example, when a new layout calls for a larger headline, the size relationship encodes the knowledge necessary to proportionately increase the size of the subheadline. Automatic layout using case-based reasoning suggests a way by which new layouts can be automatically generated by identifying an existing layout and adapting it to solve the new layout problem.

A complete description of a constraint includes a constraint predicate and a set of methods — program segments — that are used to satisfy the constraint when the values of the constrained variables are changed. Graphic designers are not programmers, and we cannot expect them to learn programming in order to encode constraints into the layout. In addition, encoding relationships as constraints is not a part of the design process — we cannot expect designers to spend time encoding relationships that, for all intents and purposes, have already been stated.

Our general approach to layout understanding consists of two broad stages: i) block segmentation, and ii) layout reasoning. The block segmentation stage is responsible for converting the image representation of the input layout into a set of text and non-text objects. This involves the application of image processing techniques in several steps — contrast stretching, thresholding, edge detection, edge linking, block detection and block classification. The goal of the layout reasoning stage is to try to infer the maximum number of *intentional relationships* between the elements of the layout. This in turn involves several





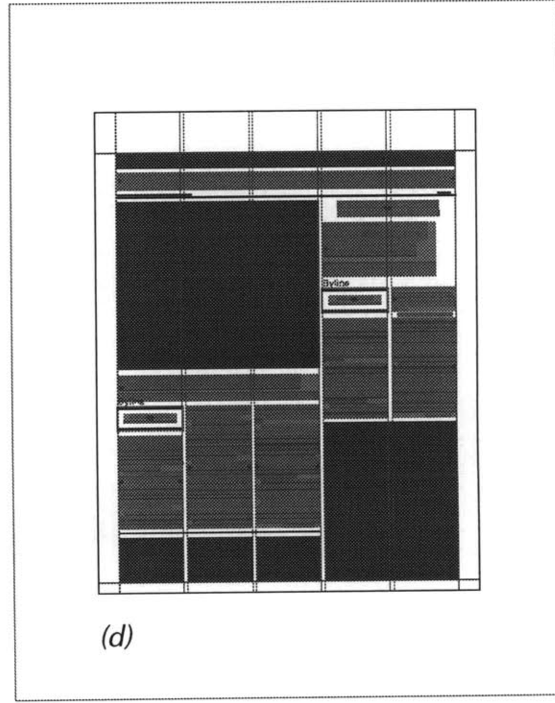
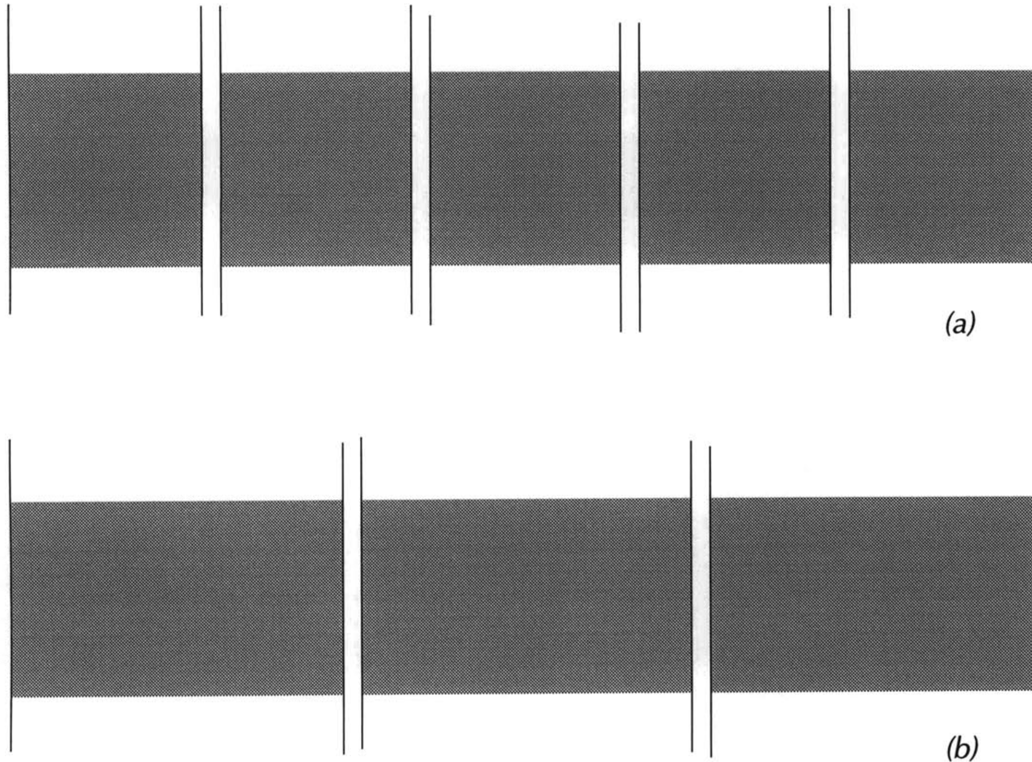


Figure 1.1 (a) an optically scanned image is presented to the system, (b) after contrast stretching and thresholding, the image is filtered to detect the edges as part of the block segmentation stage, (c) later steps in block segmentation assemble the edges into meaningful boundaries, and assemble the boundaries into meaningful blocks of text and non-text objects, (d) finally, in the layout reasoning stage, primitive objects are re-classified into higher level objects, related objects are grouped, and constraints are installed on the grouped objects.



Representing a layout as a network of constraint relationships allows it to be dynamically adapted. Shown here are two possible ways in which the Reefer group (the group of three graphic objects that appear at the bottom of the layout shown on the previous page) can be adapted to fill five columns. In (a), the members of the Reefer group are constrained to the width of the grid column, and that results in two new reefer objects being created. In (b), members of the Reefer group are all constrained to be the same width, and that results in their widths being increased.

reasoning steps — primitive text and non-text objects are classified into higher level objects such as *headlines* and *subheadlines*, objects that are likely to be related are grouped, and inferred constraints are installed on grouped objects. The constraints inferred by the layout reasoning stage are then solved by a constraint solver to verify that the constraints have been correctly inferred. Figure 1.1 shows an example of the input layout and its transformation by the various processing steps.

The representation and reasoning techniques used in the system are intended to infer the structure of a broad range of medium complexity layouts that meet the following requirements: i) primitive objects in the layout can be classified by matching them against prototypical objects in the knowledge base, ii) a typographic grid is used as the main organizing device in designing the layout, iii) logically related objects adhere to a specific spatial structure, and iv) the specific relationship between the attributes of objects can be described by a predefined function.

The rest of this thesis structured as follows. Chapter 2 provides some practical applications for the results of the research, and highlights one application towards which it is specifically

oriented. Chapter 3 provides the philosophical basis for the research. Chapter 4 presents related work in layout understanding and representation. Chapter 5 describes our approach to layout understanding. Chapter 6 describes the specific implementation details. Chapter 7 concludes by providing a summary of results and suggestions for future work.

Chapter 2

Application Scenarios

The problem of understanding a layout can be generally stated as the problem of automatically creating a structured representation where none previously existed. In this section, we explore some of the applications that could benefit from a layout understanding engine.

Automatically Encoding Style Catalogs

Desktop publishing software tools such as Microsoft Word and Frame Technologies Frame Maker are popular tools used to produce documents such as this thesis. These tools typically provide users with interfaces to "format" the content of their documents based on a catalog of predefined "styles". A style encapsulates the visual attributes of a specific class of object by assigning values to its attributes. These tools also provide an interface to define and add new styles to the catalog. Adding new styles to a catalog represents a significant amount of the document creation effort; more importantly, it is a process that involves dealing with nonstandard vocabulary and complex and often un-intuitive user interface artifacts.

The layout understanding scheme described in this thesis can be used to automatically create style catalogs in the manner shown in Figure 2.1. Note that our relational representation of layout structure is much richer than that typically used by the desktop publishing tools. In order to automatically create a style catalog, we would simply omit the part of the layout understanding process that instantiates constraints.

Automatic Layout Using Constraints and Case-Based Reasoning

Automatic layout using constraints and case-based reasoning is the primary application scenario for this project. In this scenario, described in Figure 2.1 automatic layout can be thought to consist of the following steps:

Object Recognition. The purpose of the object recognition front-end is to produce a description of the layout as a set of objects. The exact function of the object recognition

front-end depends on the nature of the input. Figure 2.1 describes the many ways in which input to this front-end could be provided. For example, the layout could be directly sketched, and the objects recognized using a sketch recognition techniques. Alternatively, the layout could be scanned into the system and the objects recognized using image processing methods. A third choice would allow the layout to be described in a low-level page description language such as PostScript, and the objects recognized using a PostScript preprocessor. In each case, the output of the recognition process is a description of the layout in terms of the primitive objects and their geometries.

In this project, the object recognition module takes as input the scanned image of an existing layout. It uses a combination of algorithmic and heuristic techniques to convert the input layout bitmap into a more structured representation as a set of text and non-text objects. In keeping with the terminology commonly used to describe this type of processing, we will henceforth refer to object recognition as Block Segmentation [Kasturi 92].

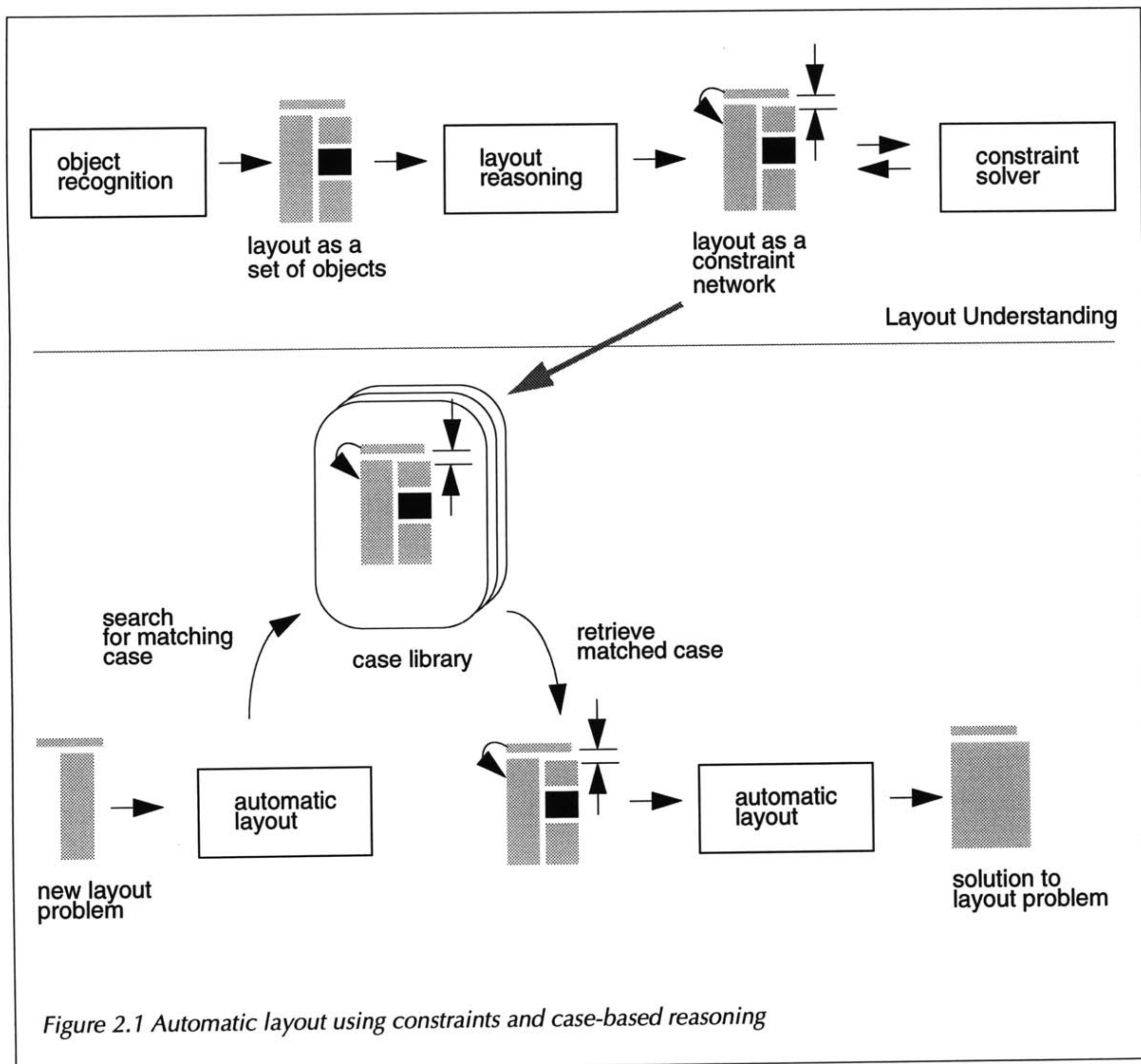


Figure 2.1 Automatic layout using constraints and case-based reasoning

Layout Reasoning. The output of the block segmentation module is then handed to the Layout Reasoning module. This module infers relationships between objects in the layout and produces a set of constraint networks that represent the layout structure. The nodes in the network represent objects in the layout, and the arcs represent constraining relationships between them. The constraint solver is used to solve the constraints and ensure that there are no inconsistencies. At this point, the layout may be designated an exemplar, and stored in a case library.

Automatic Layout. At a future point in time, the system is presented with a set of objects that constitute a new layout problem. The automatic layout process takes the layout problem and produces a layout solution. In the scenario presented in Figure 2.1, the automatic layout module searches the case library for a case that best matches the given problem. If a match is found, the case is retrieved from the library and adapted to solve the new layout problem. The result of the adaptation is the solution layout.

Figure 2.1 also highlights the main focus of this project — the development of layout understanding techniques. This in turn means the development of Block Segmentation and Layout Reasoning techniques; Chapters 5 and 6 describe these techniques in greater detail.

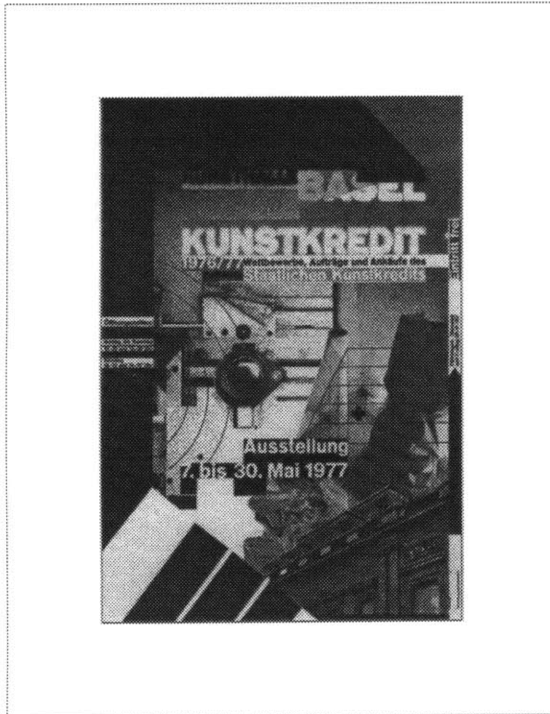
Chapter 3

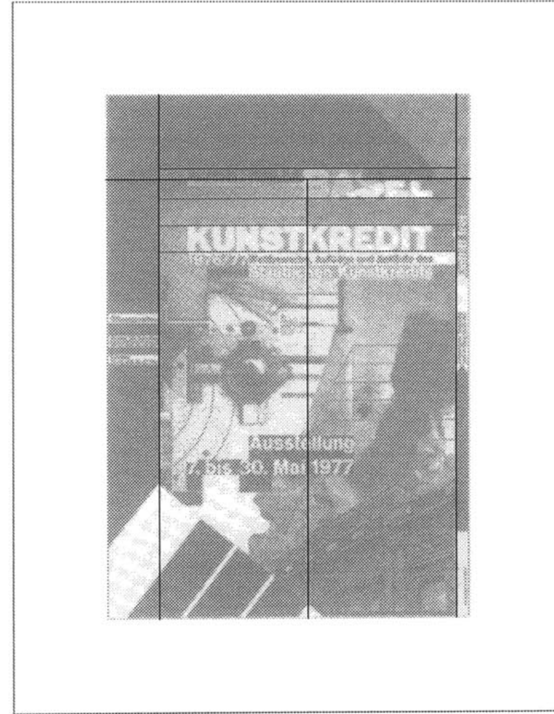
Philosophical Underpinnings

Design for page layout has been practiced for several centuries, but the formal study of design was not carried out until the 1920's and 1930's by theoreticians and practitioners at the Bauhaus and other pioneering design schools in Europe. These studies led to the formulation of several design principles — systems of conventions and rules which, when followed, helped the designer arrive at a solution for the design problem at hand. One such system of rules was formulated by Jan Tschichold in his book *Typographische Gestaltung* [Tschichold 67]. Both Wassily Kandinsky at the Bauhaus and Theo van Doesburg of the Dutch de Stijl movement identified and emphasized the importance of the *grid* in art and design [Lupton 91].

If the design of a page layout involves establishing order and structure among the elements constituting the content of the page, then the efforts of Tschichold, Kandinsky and van Doesburg have been attempts to formulate *design systems* — systems to help designers establish order among the elements of a page. Each system seeks to help the designer find a solution to the design problem which is "functional, logical and also more aesthetically pleasing" [Müller-Brockmann 81]. In practice, designers may draw upon many diverse design systems, including ones they have devised themselves. A good design system allows a designer to express his creativity, while placing constraints necessary to establish structure and organize the given contents of the page; the dual needs of expressing creativity and establishing order are at work in any design, and all designs exhibit some of the former and some of the latter, with every design seeking to achieve an appropriate balance of the two.

A graphic designer uses his creative skills to satisfy the creative communication goals of the layout. For example, playfulness and unconventionality are executed through a dynamic layout in which the elements do not exhibit any obvious structure, but are built using concrete structuring devices. An extreme example of this can be found in the works of Wolfgang Weingart and his students, who created apparently painterly compositions based on very complex grids and typography.





Shown on the left is a seemingly unstructured composition by Wolfgang Weingart, that is actually based on a very complex grid and typography. Shown on the right are some of the structuring devices used in the composition. All the measures shown bear proportional relationships with each other.

A graphic designer uses his organizing ability and the principles laid out by the design system to establish order so as to satisfy the rational communication goals of a layout. For example, to show the reader where to start reading, to differentiate between types of text information, or to reinforce relationships between elements.

How does the graphic designer come to possess creative skills and organizing ability? The answer to this question does not come easily, and any thorough treatment is well beyond the scope of this thesis. Creativity is hardly understood, and some believe that it is futile to attempt to develop a single, sweeping theory of creativity [Gardner 93]. We speculate, however, that *design systems,* such as those of Tschichold [Tschichold 91], Müller-Brockmann and Gerstner [Gerstner 63] are devices that help designers hone their organizing ability. Arguably, not all design systems are ubiquitous, and indeed, some are invented to solve a specific design problem. Still, as a part of design education, the novice graphic designer necessarily *learns* about certain design systems such as the typographic grid and variations of Tschichold's rules on typography in layout design.

Learning in general, and also specifically in the domain of layout design, is often subdivided into various types: i) *learning by being told* — be taught the principles embodying a design system either by reading books or by a design master, ii) *learning by example* — look at examples of designs created using the design system, and iii) *learning by doing* — solve new design problems using a design system. Evidently, each form of learning results in something new being learnt — new knowledge of facts, new knowledge of

models, and new skills, respectively [Boden 87]. However, when placed in the context of computer-based learning, the common thread that binds the three forms of learning is that each involves the development of internal representations of that which is learnt.

In this project, we are interested in creating computer-based *knowledge representations* of the result of the learning experiences from each of the three approaches, with the eventual goal of developing a methodology to build page layout understanding systems.

Armed with such knowledge representations, it is often possible to "reverse engineer" an existing layout and recount specific design conventions went into its design. For example, by comparing the left edges of two elements of a layout, we could claim that the two elements were left aligned - a single step inference arrived at by comparing two values. Making such inferences can also be termed *reasoning* about the layout.

In this project, we are interested in reasoning about the layout in order to create structures that represent symbolic relationships between the elements in the layout. Some amount of knowledge is necessary to reason about the layout, and the choice of a knowledge representation influences the kinds of the reasoning that are supported.

Davis et. al. [Davis 92], in addressing some of the fundamental issues related to the role of knowledge representations in building knowledge based systems, make the observation that "representation and reasoning are inextricably intertwined: we cannot talk about one without also, unavoidably, discussing the other". In fact, their discussion of the fundamental roles that knowledge representations play provides a suitable context within which to discuss some of the representation and reasoning choices that we have had to make in developing a methodology to understand layouts.

Davis et. al. argue that a knowledge representation plays five distinct roles: i) it is a surrogate, ii) it is a set of ontological commitments, iii) it is a fragmentary theory of intelligent reasoning, iv) it is a medium for efficient computation, and v) it is a medium of human expression. In the following sections, we will examine each of these roles in turn.

1. A Knowledge Representation is a Surrogate

> A knowledge representation is most fundamentally a surrogate, a substitute for the thing itself, used to enable an entity to determine consequences by thinking rather than acting, i.e., by reasoning about the world rather than taking action in it.

The goal of this project is develop a methodology to build page layout understanding systems. This in turn involves the development of computer-based representations for two broad categories of knowledge.

First, we need the knowledge that will help us understand the structure of an existing layout. For example, knowledge of the use of the typographic grid will allow us to reason about a given layout in order to determine the number of columns used in its design. In this thesis, we will refer to such knowledge as *conventional knowledge*. The conventional knowledge that we will represent is a surrogate for the real thing. Of course, in this case, the

real thing exists only in the mind of the designer, and we have no evidence, scientific or otherwise, of its actual form. The quality of inferences made by a surrogate can only be measured by comparing them against the inferences made by a human designer.

The goal of the reasoning process is to create structures that represent relationships between the elements of the layout. These structures are the second category of knowledge that we wish to represent. In this thesis, we will refer to such knowledge as *layout structure*. It is difficult to say that the perception of structure in a layout results in any representation of the structure in the mind of the reader; consequently, it is much harder to say that it results in an universal representation of the structure in the minds of all readers. Eventually, the choice of representation for layout structure is strongly influenced by the kinds of problems we can set out to solve given that structure. In this project, our representation of layout structure is chosen to help solve new design problems using a specific computer based automatic layout method involving the use of case-based reasoning [MacNeil 90] [Colby 92].

2. A Knowledge Representation is a Set of Ontological Commitments

> It is a set of ontological commitments, i.e., an answer to the question: In what terms should I think about the world.

Ontological Commitments for Conventional Knowledge

In this project, we use conventional knowledge to reveal the structure of an existing layout. Input layouts, obtained as a result of optical scanning, are encoded in the form of an image bitmap. The structure of the layout is gradually revealed as a result of several processing steps, which can be grouped into two broad stages — block segmentation and layout reasoning.

The block segmentation stage involves the use of image processing algorithms and heuristically derived rules that result in the input layout being represented in terms of the geometries of the elements it contains. Henceforth, we refer to the elements of a layout as *objects*. Objects are either text objects or non-text objects.Text and non-text objects are the most primitive classes of objects in our knowledge representation.

There are many steps that comprise the layout reasoning stage — initially, some primitive objects are reclassified into higher level objects (a non-text object that matches the prototypical horizontal rulebar is reclassified as a horizontal rule); during other steps, related objects are grouped together using rules (all objects that are left aligned are grouped by the left-aligned relation), and new classes of objects are instantiated (the inferred grid, instances of matched groups).

Each of the stages involves some reasoning over an object domain, and our view of the world is influenced by the inferences that need to be made. Objects, relations, rules and algorithms are some of the ontological commitments that we make in order to represent conventional knowledge.

For example, knowledge represented in the form of a *rule* states that:

```
if      the object is a non-text object and
        the object's width is more than 5 times its height
then    the text object is a horizontal rulebar
```

Ontological Commitments for Layout Structure

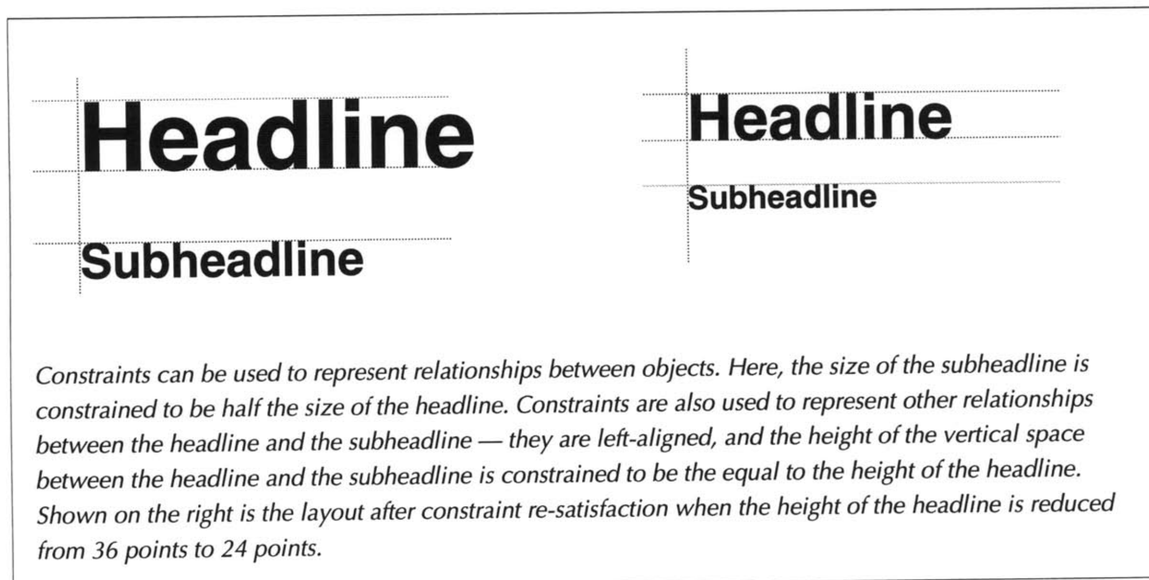
In this project, we represent layout structure in the form of symbolic relationships between objects. For example, the convention used to differentiate between a headline and a subheadline is to provide visual contrast between them. This may take the form of a size relationship, such that the size of the subheadline is some percentage of the size of the headline. The design rule may further mandate that the headline and the subheadline be *constrained* to this size relationship.

Constraints are a convenient way to represent the many inter- and intra-object relationships in a layout. The entire "look and feel" of a layout can be captured by encoding the relationships in the form of a set of constraint networks. Encoding the layout in this manner lends itself to adaptability. In other words, a constraint not only captures the current values of the related variables (for example, that the headline is 36 points and the subheadline is 18 points), but captures the relationship in the form of an equation that can be re-satisfied if the value of one of the related variables changes (for example, the size of the headline is twice the size of the subheadline). This constraint relationship between the headline and the subheadline thus has two parts. A predicate:

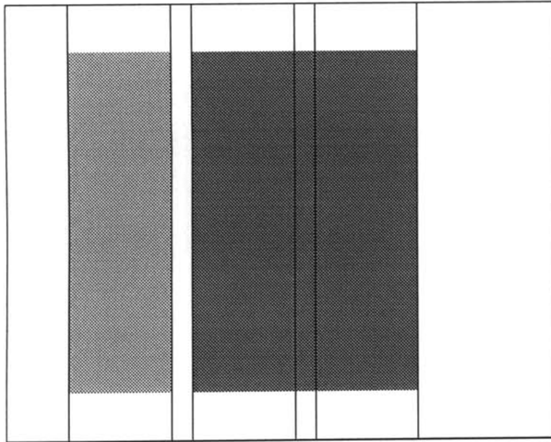
```
headline.size = 2 * subheadline.size
```

and a set of methods that are used to enforce the constraint:

```
headline.size <= 2 * subheadline.size          subheadline.size <= 0.5 * headline.size
```



Constraints can be used to represent relationships between objects. Here, the size of the subheadline is constrained to be half the size of the headline. Constraints are also used to represent other relationships between the headline and the subheadline — they are left-aligned, and the height of the vertical space between the headline and the subheadline is constrained to be the equal to the height of the headline. Shown on the right is the layout after constraint re-satisfaction when the height of the headline is reduced from 36 points to 24 points.



Domain specific knowledge of design conventions empowers us to make elaborate inferences. For example, given the rules governing the use of a typographic grid, we can proclaim the use of a three column grid, given two columns whose dimensions are shown here.

3. A Knowledge Representation is a Fragmentary Theory of Intelligent Reasoning

> It is a fragmentary theory of intelligent reasoning[1], expressed in terms of three components: (i) the representation's fundamental conception of intelligent reasoning; (ii) the set of inferences that the representation sanctions; and (iii) the set of inferences it recommends.

Insight into the nature of intelligent reasoning has come from many different fields; the psychological tradition suggests that intelligence is a complex natural phenomenon, and that it may be a large, ad-hoc collection of mechanisms which are impossible to describe in complete, concise terms. At the other extreme, the logicists have traditionally sought to characterize intelligence in precise mathematical terms.

Each view of the nature of intelligence has resulted in a different representation technology; from logic comes Prolog [Nilsson 91], and from the field of psychology come Frames [Minsky 75] and the notion of Knowledge-based Systems[2][Davis 77]. The key aspect of logic as a representation formalism is the claim that its features are capable of expressing knowledge about any domain [Moore 82]. That, taken together with the fact that reasoning in logic is usually done using the general purpose technique of deductive inference, means that logical representations are totally domain independent. In contrast, knowledge-based systems emphasize the accumulation of large amounts of knowledge in a single domain, and the development of domain specific reasoning techniques.

To represent conventional knowledge and to reason about it, we use a combination of objects, rules, domain specific procedural mechanisms and statistical reasoning methods. In this sense, we use a knowledge-based approach to understanding layouts.

---

1. Davis et. al. state that the theory is fragmentary because the representation typically incorporates only part of the insight that motivated it.

2. Psychology and logic are but two distinctly different positions on the nature of intelligence. Other fields of study have provided insights into intelligent reasoning, with each inspiring an associated representation technology; biology (resulting in Connectionism as representation technology), statistics (resulting in Causal Networks) and economics (resulting in Rational Agents).

On the other hand, layout structure is represented in a declarative fashion in the form of constraint relationships. Solving a problem in the domain of this representation means ensuring that all constraints are re-satisfied when the value of one of the variables is changed. This is done using a constraint solver, which, in our case [Freeman-Benson 90], implements a domain independent constraint satisfaction strategy.

Sanctioned inferences are best understood in the context of a logical representation. The only sanctioned inferences in logic are sound inferences - for example, those allowed by deduction. Conventional knowledge in our system is modelled after the knowledge stored by a human expert, and incorporates heuristic knowledge which may lead us to unsound inferences.

The problem of finding relationships among the objects in the layout can be decomposed into two sub-problems (i) finding objects that are related, and (ii) relating specific attributes of the related objects. For example, we must identify the specific headline and subheadline objects that are related, and know to relate their size attributes. Without some knowledge that guides us in the direction of these recommended inferences, we will attempt to create structures that were never intended by the designer.

4. A Knowledge Representation is a Medium for Efficient Computation

> It is a medium for pragmatically efficient computation, i.e., the computational environment in which thinking is accomplished. One contribution to this pragmatic efficiency is supplied by the guidance a representation provides for organizing information so as to facilitate making the recommended inferences.

Much as we would like to pontificate about the epistemological purity of our representation, there is no escaping the fact that we must compute with it. Some of the efficiency considerations are influenced by the inferences that are recommended by the knowledge representation. For example, not knowing which objects to relate means that the reasoning process will attempt to relate all pairs of objects (assuming that we restrict ourselves to binary relationships; if we allowed higher order relationships then we would have to reason about all disjoint subsets of cardinality 2..N, where N is the number of objects in the layout). For all our troubles, we would probably relate objects that the designer never intended to relate.

We have chosen to adopt several different representation technologies - objects, rules, procedures and constraints partly for efficiency considerations. An earlier implementation followed a strict paradigm of declarative knowledge, with all knowledge represented in the form of rules, but with the tools at hand, conceptually simple inferences took unacceptably too much computation.

5. A Knowledge Representation is a Medium of Human Expression

> It is a medium of human expression, i.e., a language in which we say things about the world.

Critical information about designed artifacts is notoriously hard to articulate. The vocabulary of critical thought is nonstandard, and the meaning of the language used is vague and imprecise. Indeed, it is not uncommon for the design master to dismiss a student's design attempts with a curt "Bad, try again" [Ishizaki 93]. Adopting a standard vocabulary may not be necessary when the communication is strictly between humans, but when some part of the design process includes a computer system, some transcoding of the design in order to create computer based representations is inevitable. In this research, we suggest encoding the layout structure in the form of constraints. Davis et. al., suggest that, in designing the knowledge representation, we also closely examine its viability as a basis for human communication. How feasible is it to talk about layouts as constraints? Constraints encapsulate symbolic relationships between the objects in a layout. Referring to a layout in terms of symbolic relationships is often appropriate, but is usually restricted to relationships that can be symbolized — as when objects are left aligned, or centered with respect to each other. The design vocabulary also includes certain other predicates, such as "visual balance", for which no formal definitions exist, and it is therefore difficult to encapsulate in the form of a symbolic constraint relationship.

Chapter 4

Related Research

Figure 4.1 shows the functional components of the system that is described in this thesis. The Block Segmentation module takes as input the scanned image of an existing layout, and uses a combination of algorithmic and heuristic techniques to convert the input layout bitmap into a more structured representation as a set of text and non-text objects. This representation of the layout is the input to the Layout Reasoning module, which uses domain specific techniques to infer symbolic relationships between the objects in the layout and encode them as a set of constraints. The constraints may then be solved by a constraint solver in order to ensure that there are no inconsistencies.

In this project, we draw from and build on past research in several different areas. Before reviewing specific research in each of the areas, we will list and describe each of the areas.

Document Image Analysis

Research in the field of Document Image Analysis addresses the problem of creating high level descriptions of the contents of paper-based documents. The objective of a typical document image analysis system is to take as input a raster image obtained from optically scanning a document page, segment the image into text, image and graphics regions, and further analyze each region so as to create higher level representations to help solve

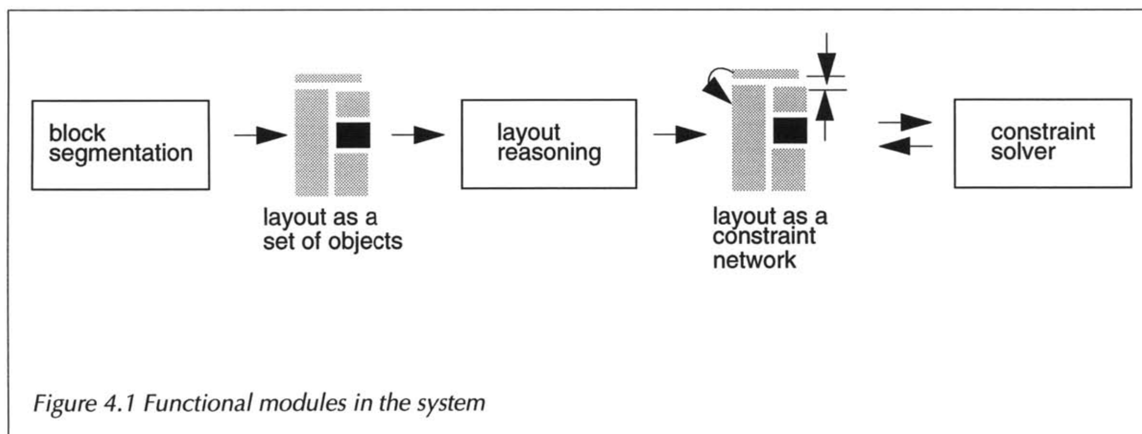


Figure 4.1 Functional modules in the system

specific problems. For example, segmentation into text non-text regions identifies the regions that can be input to an optical character recognition system. Approaches to segmentation can be classified as either top-down methods that work with some knowledge of the structure of the document, or bottom-up methods that create higher and higher level structures by layered grouping operations. However, many approaches use some combination of the two. Where appropriate, we identify the approaches described below as either top-down or bottom-up approaches.

Wong, et. al.

Wong et. al. [Wong 82] describe the requirements and components of a Document Analysis System that performs two types of analyses on document images digitized by an optical scanner — block segmentation, and character recognition. Their approach to block segmentation is described below.

For block segmentation, Wong et. al., describe a bottom-up, non-linear run-length smoothing algorithm (RLSA) that can be used to create a bitmap of white and black areas representing blocks containing various classes of data. The basic RLSA is applied to a binary sequence in which white pixels are represented by 0's and black pixels by 1's. The algorithm transforms a binary sequence x into an output sequence y according to the following rules:

1. 0's in x are changed to 1's in y if the number of adjacent 0's is less than or equal to a predefined limit C.

2. 1's in x are unchanged in y.

The RLSA has the effect of linking together neighboring black areas that are separated by less than C pixels. With the appropriate choice of C, the linked areas will be regions of a common data type. The RLSA is applied to each row and to each column in the document image, and the two resulting bitmaps are combined in a logical AND operation. Blocks in this bitmap representation are labeled using a boundary following algorithm, and as each block is labeled, several measurements are taken about the block. These measurements represent the *features* of the block; classification of the blocks is then done using a heuristic, rule-based approach where each rule matches patterns that are expected among a block's features.

Bayer, et. al.

Bayer, et. al. [Bayer 92] describe some of the conceptual modules that are claimed to be the minimum requirements of a document analysis system. However, it appears that their document domain is restricted to documents that contain only text objects.They state that every document analysis system must deal with two issues: analysis of layout structure, and classification of individual characters.

In their system, layout analysis is restricted to identifying primitive text objects in a page, and the task of classification deals with character recognition. They also argue that this kind

of analysis leads to a "weak form of understanding", and state that when attempting to understand documents on an higher level, more abstract logical concepts must be attached to objects of the document rather than just content.

In this context, they describe their work towards a document representation language called FRESCO (Frame representation language for structured documents) where each document "concept" is described by three properties: i) a set of attributes ii) a set of relationships between concepts and iii) a set of constraints. They outline a knowledge based top-down approach to automatically creating high level structured document representations, but, as with all top-down document analysis methods, a structured representation of document being analyzed is assumed to be available (in this case, encoded in FRESCO).

Structured-Document-Image Utility

Nagy [Nagy 92] describes work in progress on a Structured-Document-Image Utility that uses a syntactic approach to analyzing binary document images with the goal of producing rectangular blocks that are labeled in terms of functional components such as *title, abstract* and *subheading*. Unlike the system described by Bayer, et. al., their document domain is not restricted to text-only documents, but it is not clear if non-text blocks can be recognized and labeled.

The key aspect of their approach is use of a collection of publication specific *block-grammars* in order to simultaneously segment and identify functional components. Block-grammars are used in conjunction with a data structure known as an X-Y tree, which is used to store the document being analyzed as a nested tree of rectangular blocks. To start with, the root of the tree is the largest rectangular block — usually the original document image. Each segmentation of a block yields a set of labeled sub-blocks; each resulting sub-block can then be further decomposed according to the context-free block grammar applied to a unique string extracted for the just-segmented block.

The goal of the proposed system is to identify and label functional components of a document in order to then facilitate the search and retrieval of the content of the scanned document; it does not appear that the research is geared towards building a system to understand layouts. Nagy describes three application scenarios for the proposed system: i) view portions of a page image when screen resolution or low transmission rates prevent viewing the entire page image, ii) automatically selecting portions of the document for optical character recognition (for example, only the Abstract of a document), and iii) automatically link images with ASCII text files.

ANASTASIL

Dengel [Dengel 92] describes a knowledge-based document analysis system aimed at analyzing business letters. The system identifies conceptual parts (high level objects such as recipient, sender, and company logo).

Like the system that we describe in this thesis, document analysis in ANASTASIL consists of two parts: i) a low level geometric analysis phase that uses a top-down segmentation

scheme in order to represent the document in its component text and non-text parts, and ii) a high level geometric analysis phase that combines measures of belief with a rule-based approach in order to further classify the text and non-text objects into higher level objects. The final representation of the layout is a hierarchical representation, where each node in the hierarchy, is a layout object. Each layout object has "common" attributes (such as geometry), and a label (such as *recipient*) with its associated measure of belief.

As with all top-down approaches, ANASTASIL must rely on the existing representations of the business letters that are candidates for analysis.

docstrum

O'Gorman [O'Gorman 92] describes *docstrum*, a page layout analysis method based on bottom-up nearest-neighbor clustering of page components. The method can be applied to page images that are completely composed of text components, and results in the identification of text lines and text blocks, as well as their orientation. The docstrum method of analysis is independent of both global and local skew.

The page image is preprocessed to eliminate noise and determine the bounding boxes of connected components (each individual text character is the smallest connected component). The next step of the analysis creates the *k*-nearest neighbor[1] pairings for each of the connected components. Then, each nearest-neighbor pair, $\{i, j\}$, is described by a 2-tuple, $D_{ij}(d, \phi)$, of the distance, d, and the angle, ϕ, between the centroids of the two components. The docstrum is the polar plot of all $D_{ij}(d, \phi)$ for all nearest-neighbor pairs on the page. Using the docstrum as a global representation of the page image, orientation and text line information can be determined — the clusters on the docstrum plot yields this information directly. Given the text lines and their orientation, text blocks are inferred by grouping them on the basis of spatial and geometric characteristics.

The docstrum method is a bottom-up method that can be applied to arbitrarily oriented page images without any *a-priori* knowledge of page structure; however, the method places other restrictions on the page images: i) they must be composed only of text components, ii) since document features are measured as nearest-neighbor pairs it is essential that all characters be separated, and iii) the sizes of the individual characters must all fall into a fairly narrow range — when the page contains text of substantially different sizes, docstrum analysis must be performed on each set of sizes separately.

Constraint Based Representation

Constraint based representation mechanisms have been used in a wide variety of situations: from physics to electrical engineering to the social sciences. In the design domain, constraints have been used to represent relationships among the parts of the artifact being designed. Such a use of constraints probably has its genesis in Ivan Sutherland's Sketchpad [Sutherland 63] — a general purpose system for drawing and editing pictures on a

---

1. The *k*-nearest-neighbors to each component are the *k* closest components, where closeness is measured by the Euclidean distance in the image.

computer. In this section we review some recent research in the design domain that use constraints as the basic mechanism for representing relationships.

ThingLab

In ThingLab [Borning 79], Alan Borning created a simulation environment to facilitate the construction of dynamic models of experiments in geometry and physics. ThingLab was used to simulate the behavior of electrical circuits, mechanical linkages, and bridges under load. The main goal of ThingLab was to design and implement a language to help the user describe complex situations easily. Borning observes that "constraints are a particularly important tool for dealing with complexity, because they allow the user to specify all the relations independently of one another, leaving it up to the system to plan exactly how the constraints are to be satisfied".

Although not the principal goal of ThingLab, the techniques that evolved during its development have been used in several classes of layout design problems: dynamic documents with constraints on their contents, documents with layout constraints, and user interface components.

Adobe Type 1 Font Format

Constraint based representations for layout have traditionally focused document layouts. However, the form of a character in a specific font can also be represented declaratively as a set of constraint relationships. The goal of encoding the form of a font character as a set of relationships is to create a single, device-independent representation that can be used to render the character in a variety of display environments (from low resolution computer monitors to high resolution hard copy devices), and in a variety of different sizes.

In the Type 1 font format [Adobe 90], each character is encoded as a Type 1 program, which when executed by an interpreter results in the character being rendered. In the Type 1 terminology, constraints are called *declarative hints* — a predefined set of constraints that can be used to represent global and local relationships among the characters of a font. For example, a global Stem Width hint is used to describe the standard stem widths for all characters in the font, whereas a local Stem Width hint can be used to describe character specific variations.

Juno

Juno [Nelson 85] is a what-you-see-is-what-you-get drawing program that uses constraints to specify geometric relationships between the points in the image that is being created. For example, a Juno user can specify, in the Juno programming language, that points *a*, *b*, and *c* are collinear, or that the distance between points *a* and *b* is equal to the distance between points *b* and *c*. The Juno interpreter translates these geometric constraints into numerical constraints on the coordinates of the points, and then solves them by numerical methods. The result of solving the constraints is to render the image described by the program. The image can also be directly manipulated using the interactive features of Juno, and any manipulation results in the constraints described in the programming language being

modified.

CoDraw

CoDraw [Gross 92], is a drawing program that uses a constraint based representation to enforce and maintain simple spatial relationships. Users of CoDraw can rely on several built-in geometric constraints (alignment, tangency, offsets), or defining new constraint types using the Co programming language.

CoDraw contains many of the features that are typically found in constraint based drawing programs, but also includes some features that make it especially useful in creating constrained layouts. For example, an instance of the *grid* object can be used to constrain the position of a specific object (e.g., circle-12), or an entire class of objects (e.g., all circles).

Automatically Inferring Graphical Constraints

The idea of attempting to automatically infer graphical constraints in a scene is not new. The problem domains for the research have included user interfaces and simple line drawings; in no case has the domain been page layout.

Scene Beautification

Pavlidis and van Wyk [Pavlidis 85] have implemented a system that automatically "beautifies" a completed drawing that has been sketched roughly by trying to satisfy constraints inferred from the drawing. The system examines geometric relations between objects in the scene - collinearity of points, vertical and horizontal alignment of points. There is no guarantee that all the constraints can be inferred in one pass of the algorithm; also, it is not clear how many iterations of the algorithm are needed before it converges on a solution.

Inferring Constraints from Snapshots

Kurlander and Feiner [Kurlander 91] describe a system in which successive snapshots of a drawing are compared to determine which degrees of freedom the objects in the scene are allowed to have. Each snapshot is considered fully constrained, and the process is repeated until all the desired constraints are in place. A problem with the approach is that many redundant visual attributes that imply meaning but have none may be inferred, which may lead to a constraint network with no solution.

Peridot

In the domain of graphical user interfaces, Peridot [Myers 88] uses a rule-based system to infer constraints between pairs of objects. Object pairs in the scene are identified in the following manner. First, any explicitly selected objects are analyzed. Second, the current and last created objects are analyzed. Third, objects in the vicinity of the new object are chosen. The system stops searching when an object and a rule are found that completely specify all of the attributes of the new object. Each time a rule fires, the system asks the designer to confirm the inference. The system supports the idea of adding new rules to the

knowledge base. However, the new rules are added by encoding them in LISP.

Rockit

Rockit [Karsenty 92] looks for intersections between the position of a designated object and the gravity fields of other objects to determine the possible constraints. A rule system determines the most likely constraints to be applied between the designated object and the other objects in the scene. If the user is unhappy with the system's choice of "most likely" constraint, he has the option to cycle through the other potential constraints. Rockit supports the idea of changing the rule ordering and rule conditions, so that the quality of the inference is improved when similar actions are performed repeatedly. The system supports six kinds of constraints between objects: connectors, spacers, attractors, repulsers, containers and aligners. Spacers, repulsers and attractors can be used to represent relations about the space between objects. Rockit has the capability to dynamically modify the rule system.

Model-based Matching and Hinting of Fonts

The Type 1 font encoding format describes the form of characters of a font in a device-independent way that can be used to render the characters in a variety of different display environments. The description contains not just the character outline, but also "declarative hints" that are used to render the best looking character for a given size, resolution and display technology. Hints are a form of constraints that have to be applied to specific character parts (such as stems, bowls and serifs).

Hersch and Betrisey [Hersch 91] describe a method to automatically encode declarative hints in latin typefaces. In their approach, a topological model represents the essential formal features of a font character, and contains the information necessary to match existing "non-fancy" outline fonts to the model. For automatic hint generation, a table of applicable hints is incorporated into the topological model description. When a given input shape (described in outline form) matches the model, hints from the matched model are added to the given outline.

Chapter 5

Approach

Figure 5.1 shows the functional components of the system that is described in this thesis. The Block Segmentation module takes as input the scanned image of an existing layout, and uses a combination of algorithmic and heuristic techniques to convert the input layout bitmap into a more structured representation as a set of text and non-text objects. This representation of the layout is the input to the Layout Reasoning module, which uses domain specific techniques to infer symbolic relationships between the objects in the layout and encode them as a set of constraints. The constraints may then be solved by a constraint solver in order to ensure that there are no inconsistencies.

In this section, we describe the general problem solving approach used in each of the modules. Chapter 6 describes the implementation of each module in greater detail.

Block Segmentation

Block segmentation is a multistep process that is described in Figure 5.1. Block segmentation starts with optically scanning the original layout. The choice of the scanning

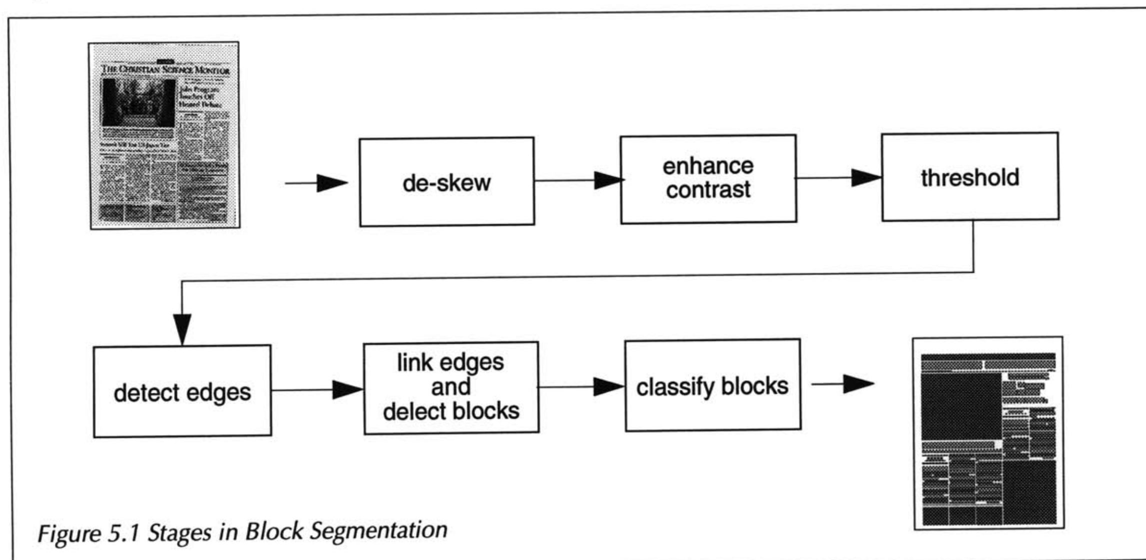


Figure 5.1 Stages in Block Segmentation

resolution is an important one; higher resolution yields a high fidelity image, but increases the storage and processing cost. For our application, since we are not interested in obtaining the fine character level details that are important for character recognition, the input layouts (typically 8.5 inches x 11 inches) are scanned at 100dpi. Since we are not using any color information, we acquire gray scale images.

1. De-skew scanned image

Subsequent processing steps are extremely sensitive to skew, so this step ensures that acquired image is free of skew. This is done by either carefully aligning the input layout on the scanner bed, or by manually rotating the acquired layout image. We have investigated the use of automatic skew correction methods [Baird 87], which could instead be used in this step.

2. Contrast Enhancement

Low contrast images can result from poor illumination or a lack of dynamic range in the imaging sensor. In this case, the optional step of contrast enhancement increases the dynamic range of the gray levels in the input image.

3. Thresholding

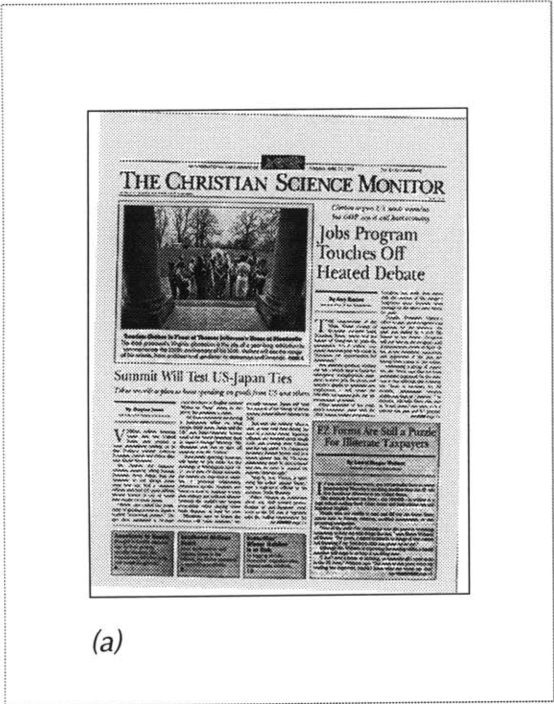
We use the technique of applying a fixed threshold to the input gray scale image. The main objective of thresholding is to separate object regions from the background and noise. Thus, the ideal threshold should be insensitive to noise, but sensitive to the edges of objects. In the process we use, thresholding also has the side effect of providing a heuristic to classify blocks. When the non-text objects in the page are full-tone gray scale photographic images (as is usually the case in newspapers and magazines), thresholding has the effect of transforming gray scale photographic images into solid black objects. We use this information to classify detected blocks as text or non-text by using a pixel density measure on the blocks.

4. Edge Detection

Edge detection is used to further subdivide the input layout image into its constituent objects. By detecting the edges of objects, we are identifying pixels in the image that correspond to object boundaries. As shown in Figure 5.2, this step converts the thresholded image into one in which the blocks can be roughly identified. Following this stage of the processing, the input layout image is still represented in the form of a bitmap.

5. Edge Linking and Block Detection

The image resulting from edge detection still far from completely characterizes the blocks corresponding to layout objects because of noise, breaks in the boundary resulting from nonuniform illumination, and other effects that introduce spurious discontinuities. The goal of this stage of the processing is twofold: i) assemble edge pixels into meaningful boundaries, and ii) assemble boundaries into meaningful blocks. Following this stage of the processing, the input layout image is represented as a set of unclassified blocks.



THE CHRISTIAN SCIENCE MONITOR

Jobs Program Touches Off Heated Debate

Summit Will Test US-Japan Ties

EZ Forms Are Still a Puzzle For Illiterate Taxpayers

(a)

THE CHRISTIAN SCIENCE MONITOR

Jobs Program Touches Off Heated Debate

Summit Will Test US-Japan Ties

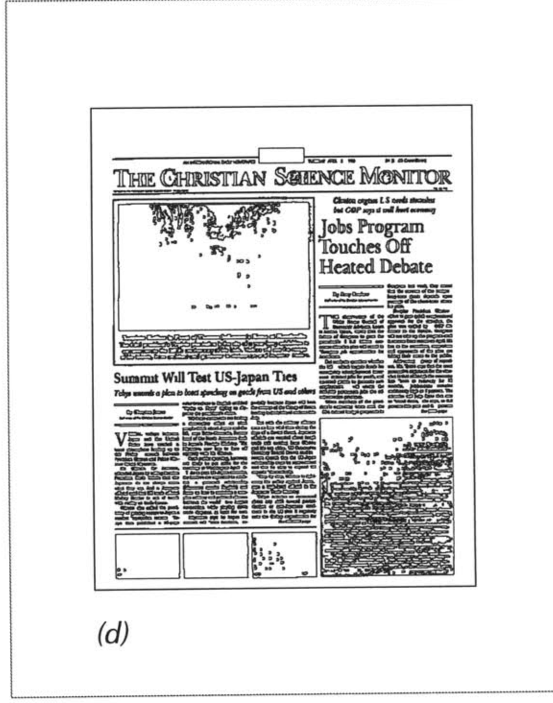
EZ Forms Are Still a Puzzle For Illiterate Taxpayers

(b)

Jobs Program Touches Off Heated Debate

Summit Will Test US-Japan Ties

(c)



Jobs Program Touches Off Heated Debate

Summit Will Test US-Japan Ties

(d)

Figure 5.2 (a) shows the layout after scanning and contrast enhancement, (b) after skew correction , (c) iafter thresholding, (d) the layout after edge detection.

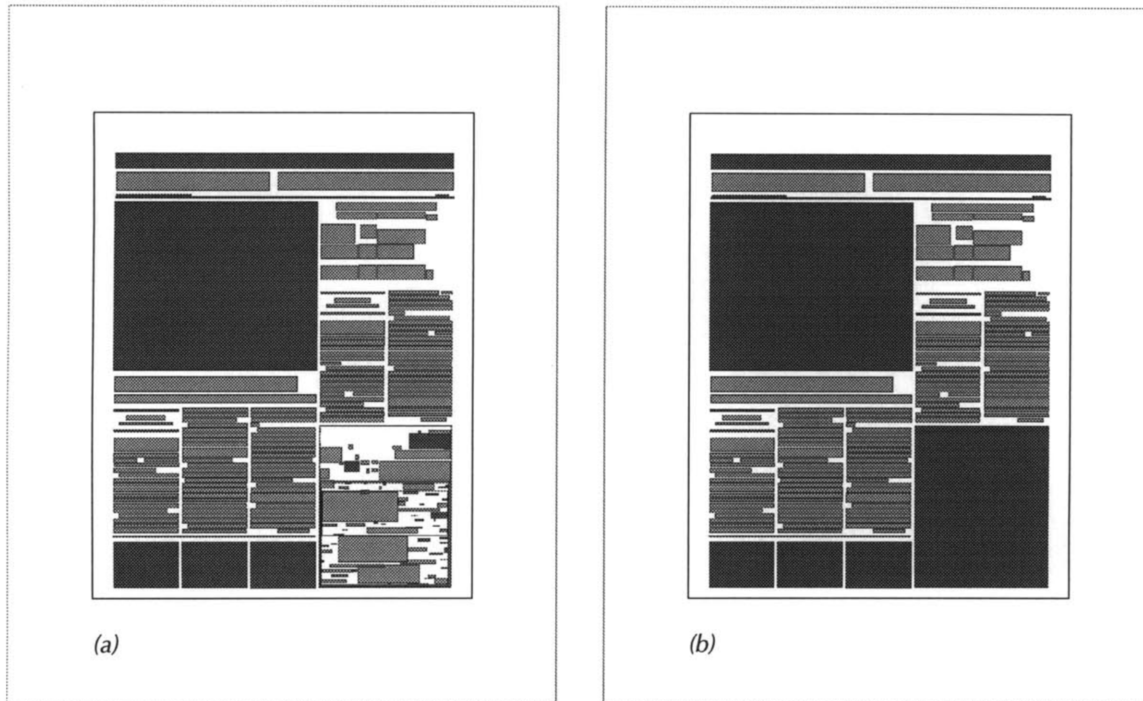


Figure 5.3 (a) shows the layout after block detection and classification, (b) the layout after manual cleanup.

7. Block Classification

Given the primitive blocks in the input image, this stage of the processing classifies the blocks as text and non-text objects. As described earlier, we use a simple pixel-density heuristic to classify blocks.

8. Manual Cleanup

The steps described above, while automatic do not correctly and completely segment all the blocks in the input layout image. There are two major source of errors: i) the use of the pixel-density heuristic sometimes results in horizontal and vertical rule bars being classified as text objects, and ii) text objects contained within text regions that composed against a non-white background are often not correctly recognized. During this stage of the processing we manually correct some of the block segmentation errors by reclassifying blocks and defining blocks that were not found.

Layout Reasoning

The goal of the layout reasoning module is to try to infer as many *intentional* relationships as possible. If we were to reason about the layout with no strategy other than to consider every pair of objects, and every pair of attributes of those objects, then we will certainly find the intentional relationships, but we'll also find relationships that imply meaning, but have none. We wish to minimize these unintentional relationships. In order to do this, we seek to answer the following questions:

1. What are the objects in the layout?

The output of the block segmentation module is a set of text and non-text objects described by their geometry (*x, y, width* and *height*). However, beyond this primitive classification, objects serve specific functional roles in the layout. For example, text objects may be further classified as *headlines* and *subheadlines,* and non-text objects may be further classified as *horizontal rulebars.* The layout reasoning module must employ some strategy to further classify primitive objects.

2. Given the objects in a layout, which objects are likely to be related?

How do we know, for instance, that headline and subheadline objects are likely to be related? If a page has several headlines and subheadlines, how do we know to pair them up correctly? The layout reasoning module must employ some strategy to group objects that are likely to be related.

3. Given a group of objects, which attributes of the objects are likely to be related?

How do we know, for instance that headline and subheadline objects are related by the *height* attribute?

4. Given a pair of attributes that relate two objects, what is the relationship between them?

Is the relationship between the attributes quantifiable? Can the relationship between the attributes be represented by some well known mathematical function?

There are two classes of intentional relationships that we will attempt to infer: i) spatial relationships between objects (object *a* is *above* object *b*, objects *a* and *b* are *left-aligned*), and ii) typographical relationships (object *a* is a *headline*, object *b* is a *subheadline*, the *height* of object *b* is *1/2* the *height* of object *a*).

The main hypothesis is that the spatial and typographical relationships in the layout are the result of the use of a specific set of organizing principles used by the designer of the layout. The representation and reasoning techniques that are used by the layout reasoning module encode organizing principles commonly used in the design of newspapers and magazines of average complexity.

Broadly stated, these are the following: i) primitive objects can be reclassified by matching them against prototypical objects in the knowledge base ii) a typographic grid is used to subdivide the page into regions, iii) horizontal and vertical rule bars are used to subdivide

the page into regions that contain objects that are logically related, iv) logically related objects adhere to a specific spatial organization, and v) when the specific attributes of objects are related, then the relationship can be described by a predefined function.

Layout reasoning is a multistep process that is described in Figure 5.4.

1. Object Classification and Identification

Some objects play specific functional roles in the layout. For example, a headline shows where to start reading, and rulebars are often used to subdivide the page into regions that contain logically related objects. The goal of this step is to reason about the layout objects in order to refine the primitive objects created by block segmentation into higher level objects such as horizontal and vertical rulebars, headlines and subheadlines.

The block segmentation process identifies text and non-text layout objects that are explictly created by the layout designer. However, many objects are implicit, and it is important to identify them for two reasons: i) they are required by subsequent stages of the reasoning process, and ii) they are first-class objects whose attributes are likely to be related to other layout objects.

As Figure 5.4 shows, object classification and identification is not all done in one step. Often, several stages of grouping must be performed before objects can be identified. For example, *column gutter* objects cannot be identified before all the *column* objects in the layout are inferred.

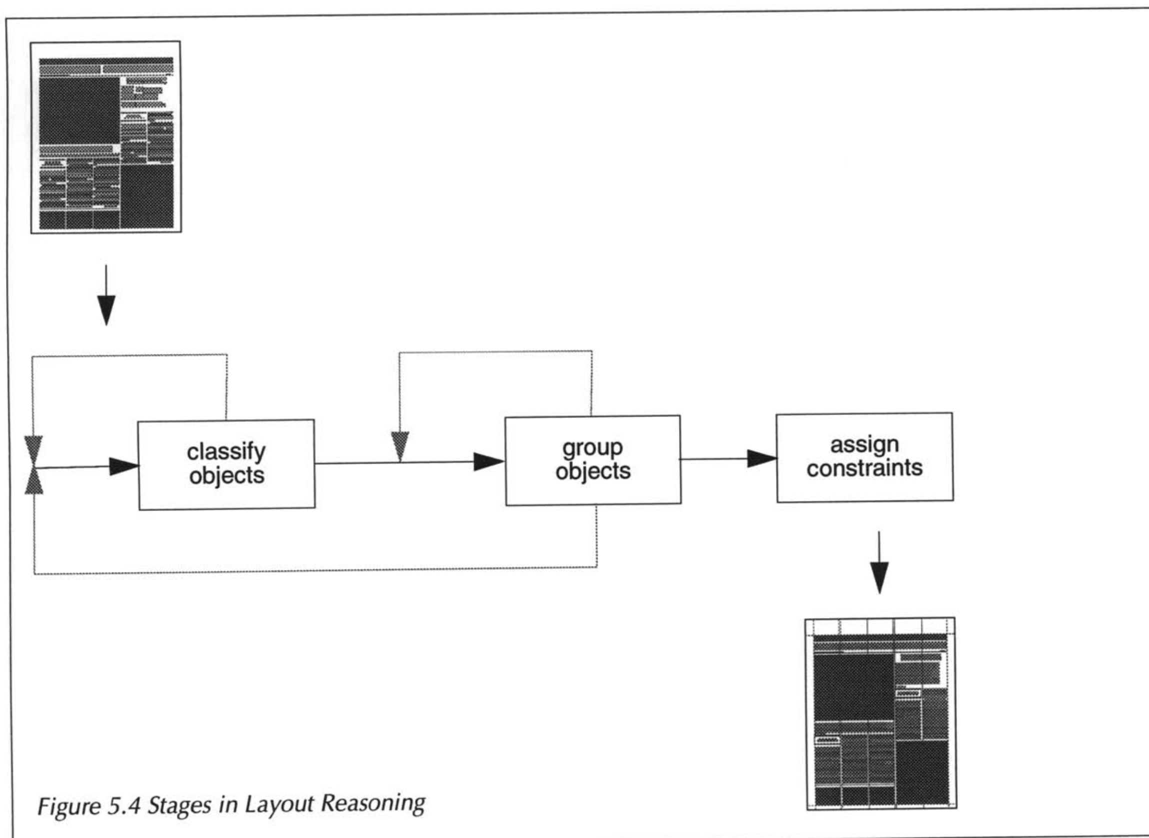


Figure 5.4 Stages in Layout Reasoning

2. Layered Grouping

The goal of this step is to identify all groups of related objects. Starting with a given set of objects and object groups (initially none), this stage employs a variety of different strategies to create higher level groups of logically related objects. As we stated earlier, it is necessary to identify these groups so that subsequent stages of the reasoning process can identify the intentional relationships between the objects in the layout.

Several different strategies are used to group objects — statistical methods are used to group text objects that belong to a single text-line, procedural knowledge is used to group objects that contained within a region created by the subdivision of the page using horizontal and vertical rulebars, and rules are used to match groups against predefined group templates which encapsulate knowledge of the class of the group members and their spatial relationships.

3. Assign Constraint Relationships

Given groups of logically related objects (note that a group may have exactly one member), this step is responsible for building the constraint network of object attributes and their relationships. This in turn consists of two steps: i) identifying the attributes that are related (and thus the constrained variables), and ii) inferring the actual relationship between the variables.

In order to identify the constrained variables, we rely on the existence of a corpus of *constraint-network templates* that identify the attributes of objects that are likely to be related. Figure 5.5 shows a constraint-network template that identifies a relationship between the left and right margins of the grid object. This knowledge allows us to retrieve the values of the attributes in order to infer the actual relationship between them.

In order to automatically infer the actual relationship between constrained variables, we assume that (and, in keeping with most standard design practice), the relationship between the variables can be described by predefined function. It is also most often the case that the predefined function is a simple linear function of the form $y = mx + c$. x and y are known, and the task of inferring the relationship between them is now the task of finding the two unknowns — m and c. Since we have only one equation from which to find two unknowns,

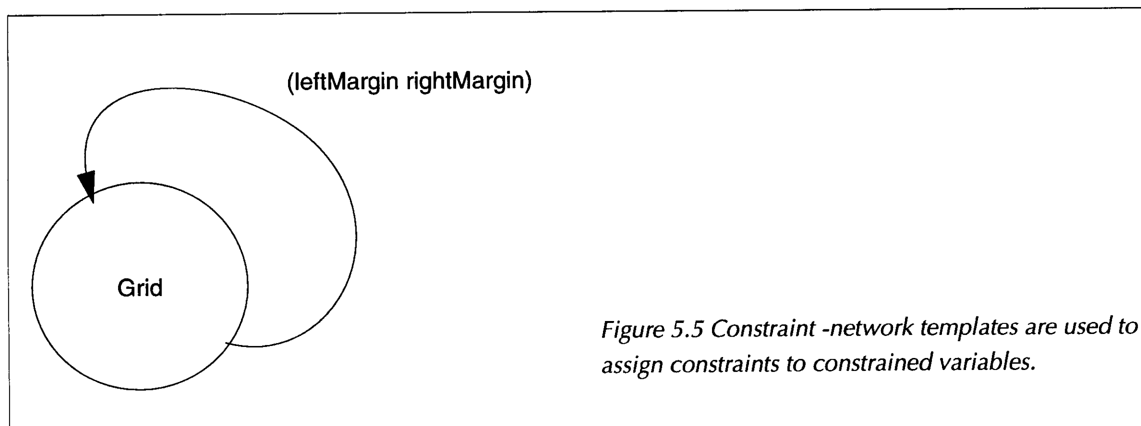


Figure 5.5 Constraint -network templates are used to assign constraints to constrained variables.

we use the additional heuristic that the relationship between the variables can most often be correctly encapsulated by the value of m alone, and equate c to zero.

Note that the relationship between constrained variables must be quantifiable. For instance, if the font of a headline was Helvetica, and the font of the subheadline was Times-Roman, then it is not possible for the system to assert anything other than what is given. One possibility is that the defining relationship here is a *compatibility* relationship that states that Helvetica and Times-Roman are compatible fonts in the specific context of headlines and subheadlines. If the font of the headline was changed to Futura, it is not clear what font the compatibility relationship should yield for the subheadline.

Constraint Solver

Constraints inferred by the layout reasoning module can then be solved by the constraint solver. The main reason for doing this is to verify that constraints have been correctly inferred. In addition, some constraints are displayed graphically, and with interactive control over the values of constrained variables; this gives us the opportunity to perturb the variables and study the effect after the constraints are re-satisfied.

Inevitably, some of the inferences will be incorrect, and we would like to have the opportunity to correct them. The current implementation of this system does not include this feature — in the future, we would like to add a Constraint Editor module that would allow us to correct mistakes during inferencing in the layout reasoning stage.

Chapter 6

Implementation

Figure 4.1 shows the functional components of the system that is described in this thesis. The Block Segmentation module takes as input the scanned image of an existing layout, and uses a combination of algorithmic and heuristic techniques to convert the input layout bitmap into a more structured representation as a set of text and non-text objects. This representation of the layout is the input to the Layout Reasoning module, which uses domain specific techniques to infer symbolic relationships between the objects in the layout and encode them as a set of constraints. The constraints may then be solved by a constraint solver in order to ensure that there are no inconsistencies.

In Chapter 5, we described the general problem solving approach used in each of the modules. In this chapter, we describe the implementation of each functional module. Physically, the system was created by integrating several freely available tools into an existing development environment within which we developed our code base.

Block Segmentation

Block segmentation is a multistep process that is described in Figure 5.1.

As described in Chapter 5, de-skewing is done by either carefully aligning the input layout on the scanner bed, or by manually rotating the acquired layout image.

The subsequent three steps of contrast enhancement, thresholding and edge detection are done using several existing, freely available tools. Contrast enhancement is done during image acquisition by using the features of the scanner driver. Thresholding and edge detection are done using tools that are part of the Khoros image processing library. Appendix I describes Khoros and its thresholding and edge detection algorithms.

1. Edge Linking and Block Detection

Figure 6.1 shows the input layout image after edge detection. The block boundaries have almost completely been identified, but several problems can be readily identified — noise,

THE CHRISTIAN SCIENCE MONITOR

Jobs Program Touches Off Heated Debate

Summit Will Test US-Japan Ties

Figure 6.1 The layout after edge detection. The block boundaries are fairly well defined, but the image is full of spurious artifacts — noise, boundary breaks and other discontinuities.

breaks in the boundary and spurious discontinuities. The goal of the edge linking process is to assemble the edge pixels into meaningful boundaries.

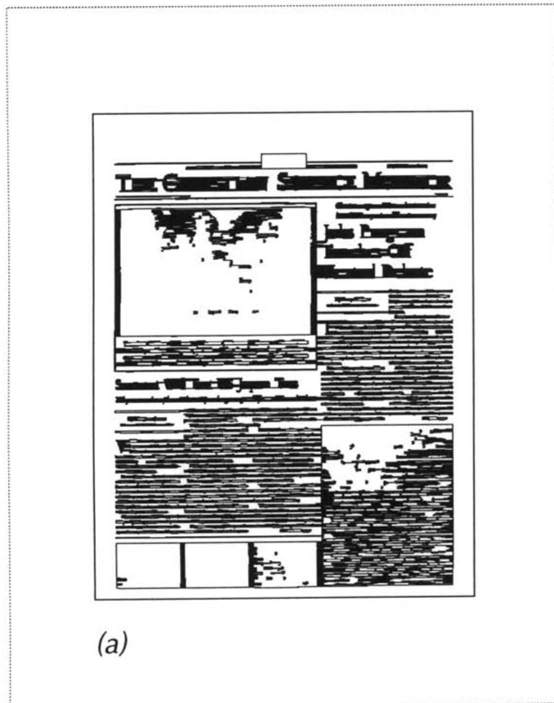
Our approach to block segmentation is a variant of the the Run-Length Smoothing Algorithm (RLSA) invented by Wong, Casey and Wahl [Wong 82]. They describe a bottom-up, non-linear algorithm that can be used to create a bitmap of white and black areas representing blocks containing various classes of data. The basic RLSA is applied to a binary sequence in which white pixels are represented by 0's and black pixels by 1's. The algorithm transforms a binary sequence x into an output sequence y according to the following rules:

1. 0's in x are changed to 1's in y if the number of adjacent 0's is less than or equal to a predefined limit C.

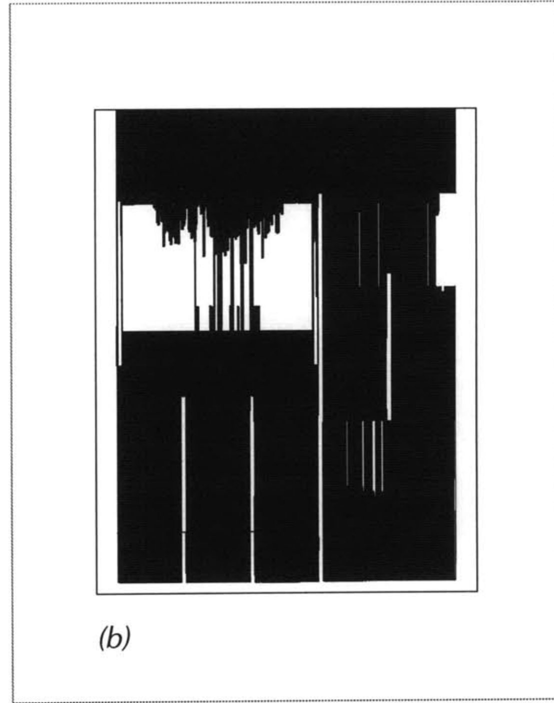
2. 1's in x are unchanged in y.

The RLSA has the effect of linking together neighboring black areas that are separated by less than C pixels. With the appropriate choice of C, the linked areas will be regions of a common data type. The RLSA is applied to each row and to each column in the document image, and the two resulting bitmaps are combined in a logical AND operation.

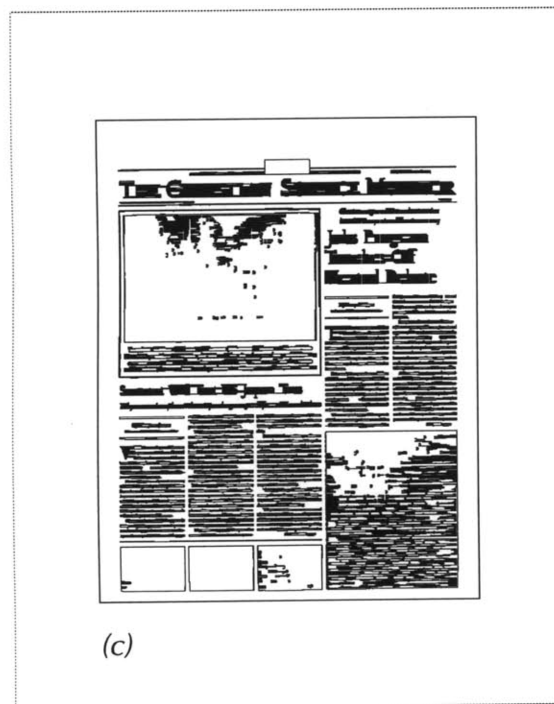
Wong et. al. apply the RLSA to a layout image after thresholding. Our approach to edge linking is different in that the RLSA is applied to the layout image after edge detection. This difference is significant, because it preserves an important attribute of the edge-detected version of the layout image that is used in the block classification stage — the interior of non-text objects are usually rendered hollow after edge detection. After the application of



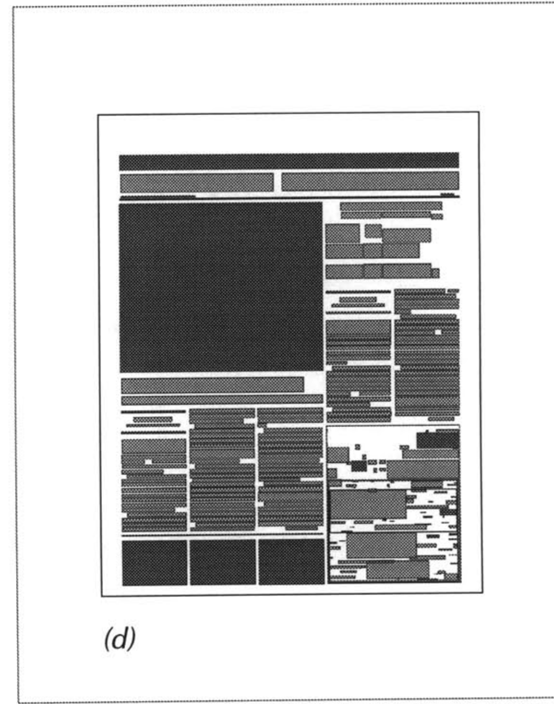
(a)



(b)



(c)



(d)

Figure 6.2 (a) Shows the layout after the RLSA has been applied in the horizontal direction, (b) shows the layout after the RLSA has been applied in the vertical direction, (c) is the result of the logical AND operation on the horizontally and vertically smoothed images, and (d) is the result after block detection and classification.

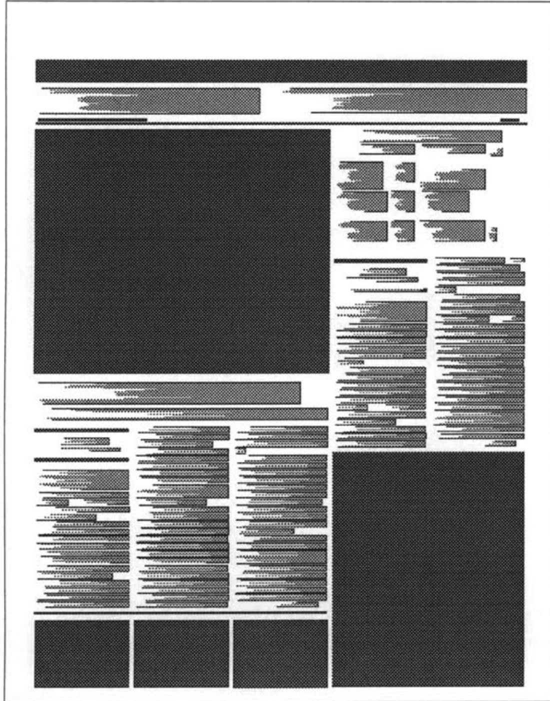


Figure 6.3 Shown here are the horizontal projection profiles for each of the text objects in the layout. The upper and lower peaks correspond to the tops and baselines of the characters in the text object.

the RLSA to the edge-detected image, blocks are identified by a boundary following algorithm.

2. Block Classification

One of the first steps in the block segmentation process is to threshold the image at a fixed, high, threshold level. This has the effect of turning full-tone gray scale photographic images into solid black objects. When the edge detection operation is subsequently performed, the interior of the photographic images is composed almost entirely of white pixels. This, and the heuristic that most non-text objects in our layout domain are photographic images (or solid black rule bars and diagrams on a gray background) are the main heuristics that we use to classify blocks. For each block, we compute the ratio of black pixels to white pixels. When this ratio is below some heuristically chosen threshold (currently 20%), then the block classified as a non-text object. Figure 6.2 shows the transformation of the edge-detected layout image as it is processed by the various stages of the Run-Length Smoothing Algorithm and block classification.

After block classification, we compute the horizontal projection profiles for the blocks that have been classified as text blocks. The horizontal projection profiles enable us to infer other attributes of the text objects. As shown in Figure 6.3, there are two peaks in the horizontal projection profiles for each text object. The distance between the peaks corresponds to the heights of the text characters. The upper peak corresponds to the tops of the characters, and the lower peak corresponds to the baselines of the characters. When adjacent text objects touch, then the edge linking process merges adjacent text objects. The presence of the peaks in the horizontal projection profiles can be used to break up incorrectly merged text objects.

3. Manual Cleanup

It is evident from Figure 6.2 that block segmentation does not correctly classify all the blocks in the source layout. There are two major source of errors: i) the use of the pixel-density heuristic sometimes results in horizontal and vertical rule bars being classified as text objects, and ii) text objects contained within text regions that composed against a non-white background are often not correctly recognized. During this stage of the processing we manually correct some of the block segmentation errors by reclassifying blocks and defining blocks that were not found. Specifically, we can choose to define text and non-text objects, and delete objects that were incorrectly detected. This is done by selecting the operation and then approximately specifying a rectangular region of the layout on which the operation is to be applied. Figure 5.3 shows the source layout after manual cleanup.

Layout Reasoning

Layout reasoning is a multistep process shown in Figure 5.4. The entire layout reasoning module is implemented in a LISP-like interpreted language called CLIPS. Appendix II describes CLIPS.

As described in Chapter 5, the execution of the object classification and layered grouping are repeated and interleaved, with a potentially different strategy being used during each execution of the object classification and grouping steps. The following code segment describes the overall control flow in the layout reasoning phase:

```
(deffunction infer-relationships ()
    (make-edge-relations)

    (infer-page-margins)

    (match-rule-bars)
    (match-columns)
    (match-text-lines)
    (match-text-classes)
    (match-textblocks)

    (infer-column-gutter-widths)

    (infer-alignment-relationships)

    (match-group-templates)

    (assign-constraint-relationships)
)
```

1. make-edge-relationships

Layout reasoning starts by forming several primitive groups that support later stages in the reasoning process. In this step, four classes of *edge relations* are formed: left-edge, top-edge, right-edge and bottom-edge. Each relation groups together all objects that are aligned to a

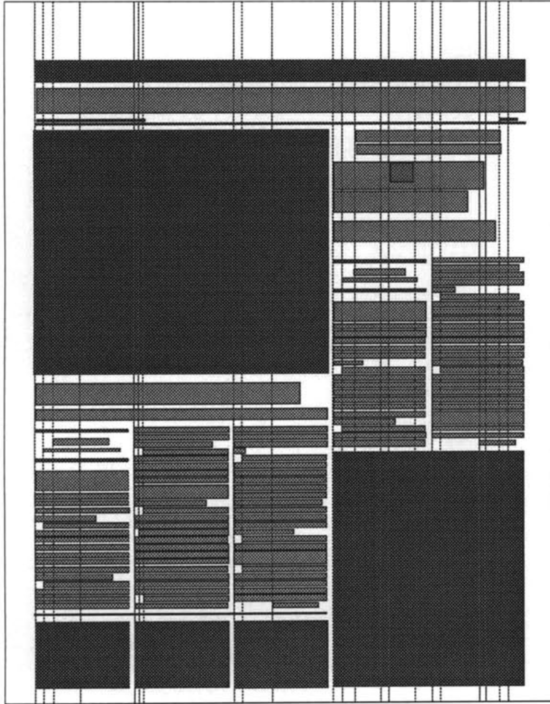


Figure 6.4 Left Edge relations

common edge. As shown in Figure 6.4, each vertical line represents a left-edge relation whose members share a common left edge. The main complication here is that errors during the block segmentation stage lead to logically aligned objects being physically offset from each other by a few pixels. We resolve this by defining a gravitational field that surrounds the edge relation that is currently being assembled, and allowing objects that are within this gravitational field to be members of the relation.

2. infer-page-margins

To infer each of the left, top, right and bottom margins of the page, the corresponding set of edge relations is used. To infer the left margin, we sort all the left-edge relations in ascending order, and for each one, compute the ratio of the total height of all members to the height of the page. If the ratio is larger than a heuristically determined threshold, then the left-edge relation defines the left margin of the page. When a left margin is found in this manner, we also assert that all left-edge relations to the left of the inferred margin also define potential left margins. When a left margin is not found in this manner, the left-most left-edge relation defines the left margin of the page. The analogous procedure is repeated to infer the top, right and bottom margins.

3. match-rule-bars

Rule bars are matched by computing the ratio of their width and height. Typical horizontal rule bars are much wider than they are tall, and conversely for vertical rule bars. If the ratio of height to width is smaller than a heuristically determined threshold, then the non-text object is classified as a horizontal rule bar. An analogous procedure is used to classify vertical rule bars.

4. match-columns

The procedure used to infer columns is similar to the procedure used to infer page margins. First, the left margin of the page is assumed to be the left edge of the first column. Having sorted the left-edge relations in ascending order, we examine each one that is to the right of the left margin. As before, for each left-edge relation, we compute the ratio of the total height of all members to the height of the page. If the ratio is larger than a heuristically determined threshold (whose value is the same as the threshold used to infer page margins), then the left-edge relation is a candidate to form the left-edge of a column. If any objects are *completely* contained within the candidate left-edge relation and the left-edge of the last column inferred, then the candidate left-edge forms the left-edge of a new column. We continue to examine left-edge relations in this manner until one is found that is to the right of the right margin of the page. Given the left-edge of all the columns, it is trivially simple to compute the widths of individual columns.

5. match-text-lines

In this stage, text objects belonging to a single line are grouped into a *text-line* object. The main issue here is to ensure that text objects that belong to other text lines are not subsumed by the text line that is currently being assembled. The issue is best described in Figure 6.5.

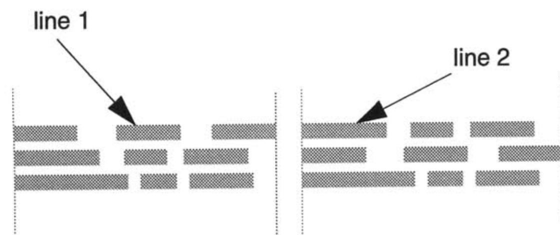


Figure 6.5 The text line matching strategy must ensure that text from line 2 is not subsumed into line 1. The situation is caused by the inter-word spacing in line 1 being greater than the inter-column spacing.

One possible heuristic that could be used to group text objects into text lines is that the horizontal space between text objects within a column of text is less than the horizontal space separating text objects in different columns. However this is often not the case, and relying on this heuristic will result in text lines that incorrectly group text objects that logically belong to several text lines. We cannot also restrict ourselves to matching text lines in a single column because text lines will often span several columns.

To match text lines, we start by scanning horizontally across the page from left to right, and column by column. Each column contains several rows of objects. A row is defined as the set of all objects that are members of the same bottom-edge relation. The following procedure is performed for all rows.

We start by sorting the members of a row in the increasing order of the left edge. If a text object is completely contained within the current column, then it is a member of the new text line object being assembled, and we obtain the next text object in the row. If the text object starts in the current column (i.e., has its left edge in the current column), but ends is a different column, then it is a member of the current text line, and we keep track of the

column that we are processing and the fact that we have a text line that spans columns. If the text line starts in a column different from the current column, or if there are no other text objects in the row, then we have matched a text line.

6. match-text-classes

In this stage, text lines are classified as headlines, subheadlines, and other text classes that server specific functional roles in the layout. Two general strategies are used: i) the use of statistical methods and text line height information to cluster text line objects that are logically part of a single class, and ii) the use of rules to infer the text class using the spatial organization of objects.

One complication arises from the fact that, because of errors during block segmentation, the heights of text line objects (that are logically part of the same class) varies by a few pixels. As before, we adjust for this discrepancy by using a heuristically determined determined deviation from a mean height within which objects are allowed to belong to the same class. Figure 6.6 shows a typical histogram for the text objects in a layout. The task then is to determine the clusters and assign class labels to the objects that are part of each cluster.

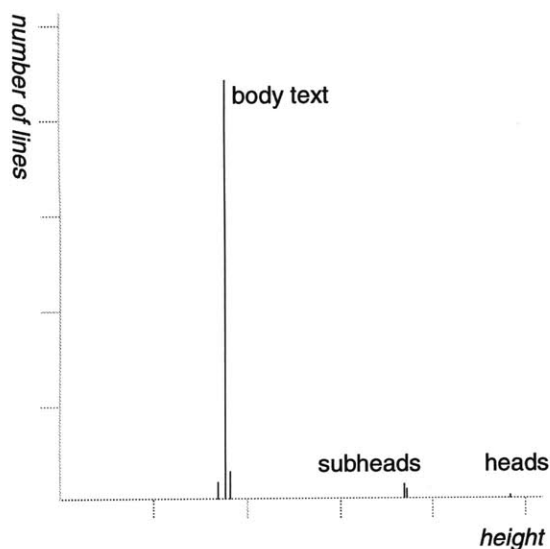


Figure 6.6 A text line height histogram is used to cluster classes of text lines. Shown here is a typical histogram where the body text lines vastly outnumber all other classes of text.

Rules are then used to further refine the classification of the text line objects. Rules encode knowledge of the spatial organization of objects to classify them. For example, if the cluster of largest text line objects has exactly one member, and that member is in the first row of text line objects on the page, then it is classified as a *Nameplate* object.

7. match-text-blocks

In this step, logically related text lines are grouped into text blocks. Two grouping rules are used: i) lines that start in the same column and span the same number of columns are

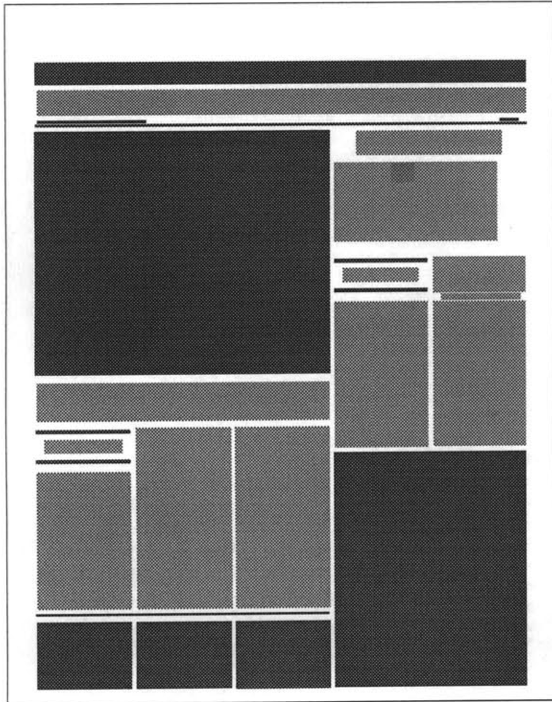


Figure 6.7 The layout after text blocks have been matched.

grouped, and ii) lines belonging to the same class are grouped. Figure 6.7 shows the layout after text blocks have been matched.

8. infer-column-gutter-widths

Gutters are the narrow band of white space that separate columns. The gutter width is often related to other measures in the layout, such as the leading of body text paragraphs. Having inferred the columns and their widths, the gutter width is relatively easy to infer. To do this, we find the right-most right-edge relation within the column that is the most significant. The most significant right-edge relation is one where the total height of its members is larger than some heuristically determined threshold.

9. infer-alignment-relations

In this step, we infer the alignment of text lines, text blocks and non-text objects with respect to the grid. For text lines and non-text objects, this means examining the relationships of the left and right edges of the objects with respect to the columns that they span. For example, if an object's left edge is aligned to the left edge of the column in which it starts, then it is *left-aligned.* If the right edge of a left-aligned object is aligned to the right edge of the column in which it ends, then it is *justified.* For text blocks, the task is complicated by the fact that text blocks may have a combination of left-aligned, right-aligned and justified text lines — as is the case for a typical multi-line paragraph. To infer text block alignment, we compute the ratio of the number of distinctly aligned lines to the total number of lines in the text block. The largest of the ratios determines the alignment of the text block. For example, if a 10 line text block has 8 left-aligned lines and 2 fully justified lines, then the text block is considered to be left-aligned.

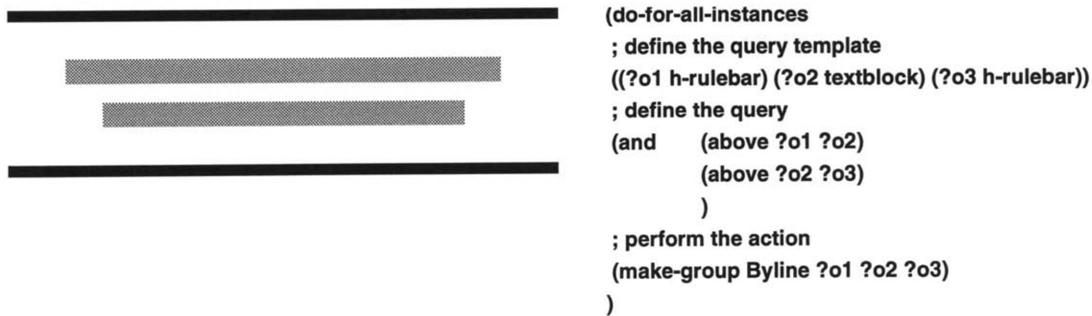


Figure 6.8 Groups of objects that are to be matched are defined using a simple visual query language. Shown here is a "by-line" group and the definition that was used to match it.

10. match-group-templates

Thus far in the process, we have reasoned about the layout in order infer higher level structures that are present in all layouts — structures like rule bars, text lines, headlines and text blocks. However, in most layouts, these structures still form the most primitive elements which are combined in various ways to form even higher level functional elements. An example of such an element in a newspaper is a *byline*, which may consist of two rule bars and two text lines as shown Figure 6.8. The task of this stage of the processing is to infer these higher level structures using entities in the knowledge base called *group templates*. A group template defines a group by encapsulating knowledge of the class of the group's constituents and their spatial relationships. As shown in Figure 6.8, a byline is defined as a set of 3 relations between its constituent rule bar and text line objects.

The problem of matching functional groups in the layout can be considered to be the problem of satisfying a query about the spatial relationships between the group's constituents. We have defined a small set of relational predicates that can be used to construct these queries. Appendix IV contains a list of these relational predicates and some examples of group templates defined using them.

11. assign-constraint-relationships

The final stage of the layout reasoning process is to represent the overall structure of the layout as a set of constraint relationships between the attributes of objects. Two strategies are used to do this: i) procedural knowledge, and ii) constraint templates.

Procedural knowledge is used to encode certain constraints that are generally known to be placed on layouts in our domain. For example, if we have inferred a multi-column layout in which all the columns have the same width, then we constrain the widths of all columns to be equal.

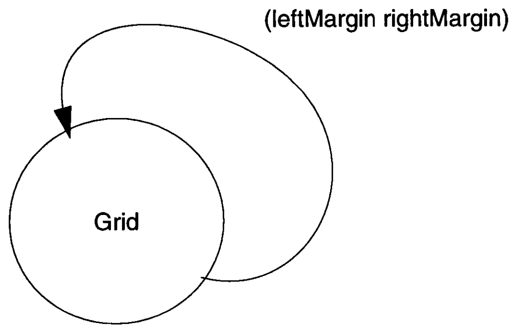


Figure 6.9 Constraint -network templates are used to assign constraints to constrained variables.

Constraint-network templates are used to encode constraints more automatically. Figure 6.9 shows a constraint-network template that is used to encode the specific relationship between the left and right margins of the inferred grid. Adding new constraint-network templates requires no changes to the inferencing strategy, and, if more relationships are to be encoded, then more constraint-network templates can be added to the knowledge base.

Constraint Solver

We have incorporated the DeltaBlue constraint solver into our system. DeltaBlue is a freely available tool and is described in Appendix III.

Chapter 7

Conclusions

Results

We have presented a knowledge based approach to understanding layouts in magazines and newspapers. We define "understanding" as the process of automatically creating a rich structural representation of an input layout from a structurally poor representation. We have used the approach to build a system that takes as input a layout obtained as a result of optically scanning it, and produces as its output a set of constraint relationships that represent the structure of the layout.

The primary application scenario for the system is automatic layout using case-based reasoning, wherein a new layout problem is solved by retrieving an appropriate case from a case library and adapting it to satisfy the constraints of the new problem.

The system was tested against scanned images of layouts of medium complexity that adhered to the following rules: i) a typographic grid is used to organize the objects in the layout ii) logically related objects adhere to a specific relationships, and iii) constraint relationships can be quantitatively described by a predefined function.

In our approach, knowledge of the use of the typographic grid proved to be the single most important piece of knowledge used to understand layouts. We use this knowledge to infer the various attributes of the grid — page margins, grid columns and the gutters between the columns. Knowledge of the specific grid used in the layout allows us to make inferences about the spatial relationships between layout objects. The presence of these spatial relationships can be used to group logically related objects; these groups then guide the creation of rules that are then used to match other groups with similar topologies. Rules used to match groups also encapsulate other relationships between the objects in the group; when rules fire, they install constraints representing the other relationships.

We have not had the opportunity to extensively test and measure the performance of the system on a broad range of layouts. Clearly, for the system to be used as a knowledge

acquisition tool for automatic layout using case-based reasoning, extensive testing is necessary. For the layouts that we have attempted to understand, the system worked well.

Future Work

Our experiences in building this system have suggested many areas for further research. Some of these areas are described in the sections below.

1. Interactive Group Definition

In the approach described here, groups to be matched in the layout have to be manually encoded by specifying the spatial relationships between the group members using the query language that we have defined. An area for further work is the interactive definition of the group templates.

2. Constraints by Example

As described earlier, we have assumed that the constraint relationship between two variables can be quantitatively described by a predefined function. This assumption has two flaws: i) it means that we can use this relationship for all instances when the given pair of variables are known to be related, and ii) it is sometimes the case that a relationship cannot be described by a predefined function. An approach that automatically infers the function that relates two variables by analyzing several examples of the use of the two variables might solve both the failings of the current approach. Further work in this area will lead to better representations of the relations between constrained variables.

3. Cycles in the Constraint Network

One of the biggest problems with our approach to automatically creating constraint networks is that there is no guarantee that it will be free of cycles. Cycles in the constraint graph means that a solution to the constraint network cannot be found by local propagation. The constraint solver that we have integrated into our system — DeltaBlue — solves constraints by local propagation. Further work is necessary to ensure that either: i) the constraint network is always free of cycles, or ii) a solution can be found to a network with cycles.

4. Understanding Dynamic Layouts

The approach described in this thesis can be used to understand "static" layouts drawn from the domain of newspapers and magazines. This approach can be used as a basis for future research to understand "dynamic" layouts. Such layouts can be drawn from two different domains; in each, the term "dynamic" connotes information that changes over time. The first category of dynamic layouts includes time varying information such as sound and video; the second category of dynamic layouts includes user interfaces, where the information in the interface (and consequently the layout relationships between the elements of the layout) changes in response to user interaction.

Chapter 8

Bibliography

[Adobe 90] Adobe Systems Incorporated, *Adobe Type 1 Font Format*, Addison Wesley, New York, 1990.

[Baird 87] H.S. Baird, "The skew angle of printed documents", in *Proceedings of the Conference of the SPSE*, 1987.

[Bayer 92] T. Bayer, J. Franke, U. Kressel, E. Mandler, M. Oberlander, and J. Schürmann, "Towards the Understanding of Printed Documents", in *Structured Document Image Analysis*, H.S. Baird, H. Bunke, and K. Yamamoto, editors, Springer-Verlag, New York, 1992.

[Boden 87] Margaret A. Boden, *Artificial Intelligence and Natural Man 2nd Edition*, Basic Books, New York, 1987.

[Borning 79] Alan Borning, "ThingLab — A Constraint-Oriented Simulation Laboratory", Stanford University Computer Science Department Report STAN-CS-79-746, 1979.

[Canny 83] J.F. Canny, "Finding Edges and Lines in Images", MIT Technical Report N 720, 1983.

[Colby 92] Grace Colby, "Intelligent Layout for Information Display: An Approach Using Constraints and Case-based Reasoning", S.M. Thesis, Massachusetts Institute of Technology, 1992.

[Dengel 92] A. Dengel, "ANASTASIL: A system for Low-Level and High-level Geometric Analysis of Printed Documents", in *Structured Document Image Analysis*, H.S. Baird, H. Bunke, and K. Yamamoto, editors, Springer-Verlag, New York, 1992.

[Davis 77] Randall Davis, Bruce Buchanan, and Edward Shortliffe, "Production Rules as a Representation for a Knowledge-Based Consultation Program", in *Artificial Intelligence*, January 1977.

[Davis 92] Randall Davis, Howard Shrobe, and Peter Szolovits, "What is a Knowledge Representation?", Memo, MIT AI Laboratory, 1992.

[Forgy 82] Charles Forgy, "RETE: A fast algorithm for the many pattern/many object pattern match problem", in *Artificial Intelligence*, January 1982.

[Freeman-Benson 90] Bjorn Freeman-Benson, John Maloney, and Alan Borning, "An Incremental Constraint Solver", in *Communications of the ACM*, January 1990.

[Gardner 93] Howard Gardner, *Creating Minds*, Basic Books, New York, 1993.

[Gerstner 63] Karl Gerstner, *Designing Programmes*, ABC Verlag, Zurich, 1963.

[Gross 92] Mark Gross, "Graphical Constraints in CoDraw", in *Proceedings of the Conference on Visual Languages*, 1992.

[Hersch 91] Roger D. Hersch and Claude Bétrisey, "Model-Based Matching and Hinting of Fonts", in *SIGGRAPH '91 Conference Proceedings*, 1991.

[Ishizaki 93] Suguru Ishizaki, Personal Communication.

[Karsenty 92] Solange Karsenty, James Landy, and Chris Weikart, "Inferring Graphical Constraints with Rockit", Memo, Digital Equipment Corporation Paris Research Laboratory, 1982.

[Kasturi 92] R. Kasturi, R. Raman, C. Chennubhotla, and L. O'Gorman, "An Overview of Techniques for Graphics Recognition", in *Structured Document Image Analysis*, H.S. Baird, H. Bunke, and K. Yamamoto, editors, Springer-Verlag, New York, 1992.

[Kurlander 91] David Kurlander and Steven Feiner, "Inferring Constraints from Multiple Snapshots", Columbia University Computer Science Department Technical Report CUCS 008-91, 1991.

[Lupton 91] Ellen Lupton and J. Abbot Miller, editors, *The ABCs of ▲■●: The Bauhaus and Design Theory*, Princeton Architectural Press, Princeton, 1991.

[MacNeil 91] Ron MacNeil, "Generating Multimedia Presentations Automatically using TYRO, the Constraint, Case-Based Designer's Apprentice", in *Proceedings of the IEEE Workshop on Visual Languages*, 1991.

[Minsky 75] Marvin Minsky, "A framework for representing knowledge", in *The Psychology of Computer Vision*, P.H. Winston, editor, McGraw-Hill, 1975.

[Moore 82] Robert C. Moore, "The Role of Logic in Knowledge Representation and Commonsense Reasoning", in *Proceedings of AAAI-82*, 1982.

[Müller-Brockmann 81] Josef Müller-Brockmann, *Grid systems in graphic design*, Arthur Niggli, Heiden (Switzerland), 1981.

[Myers 88] Brad Myers, *Creating User Interfaces by Demonstration*, Academic Press, Boston, 1988.

[Nagy 92] George Nagy, "Towards a Structured Document Image Utility", in *Structured Document Image Analysis*, H.S. Baird, H. Bunke, and K. Yamamoto, editors, Springer-Verlag, New York, 1992.

[Nelson 85] Greg Nelson, "Juno, a Constraint-Based Graphics System", in *SIGGRAPH '85 Conference Proceedings*, 1985.

[O'Gorman 92] Lawrence O'Gorman, "The Document Spectrum for Page Layout Analysis", Memo, AT&T Bell Laboratories, 1992.

[Nilsson 91] N. Nilsson, "Logic and Artificial Intelligence", in *Artificial Intelligence*, January 1991.

[Pavlidis 85] Theo Pavlidis and Christopher J. Van Wyk, "An Automatic Beautifier for Drawings and Illustrations", in *SIGGRAPH '85 Conference Proceedings*, 1985.

[Shen 86] J. Shen and S. Castan, "An Optimal Linear Operator for Edge Detection", in *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 1986.

[Shen 87] J. Shen and S. Castan, "Edge Detection Based on Multi-Edge Models", in *Proceedings of the SPIE*, 1987.

[Shen 88] J. Shen and S. Castan, "Further Results on DRF Method for Edge Detection", in *9th International Conference on Pattern Recognition*, 1988.

[Sutherland 63] Ivan SutherLand, "Sketchpad, A Man-Machine Graphical Communication System", Ph.D. Thesis, Massachusetts Institute of Technology, 1963.

[Tschichold 67] Jan Tschichold, *Typographische Gestaltung*, translated into English by Ruari Mclean as *Asymmetric Typography*, Reinhold Publishing Corporation, New York, 1967.

[Tschichold 91] Jan Tschichold, *The Form of the Book*, Hartley and Marks, Washington, 1991.

[Wong 82] K.Y. Wong, R.G. Casey, and F.M. Wahl, "Document Analysis System", in *IBM Journal of Research and Development*, November 1982.

Appendix I

Khoros

Initial steps of the block segementation phase involve the use of thresholding and edge detection algorithms. We use algorithms that are part of the Khoros image processing suite. In this Appendix, we provide an overview of Khoros, and describe in detail the thresholding and edge detection algorithms that it provides. The description that follows has been created using parts of the on-line documentation that is part of the Khoros distribution.

About Khoros

The Khoros system integrates multiple user interface modes, code generators, instructional aids, information processing, and data visualization to produce a comprehensive image and information processing research tool. This system can easily be tailored to other application domains because the tools of the system can modify themselves as well as the system. This attribute is important in a system that is designed to be extensible and portable.

The Khoros infrastructure consists of several layers of interacting sub-systems. A high-level user interface specification (UIS) combined with methods of software development, embedded in a code generation tool set, qualify as a user interface development system (UIDS). An interoperable data exchange format and algorithm library contain the application specific layer. One of the most powerful features of the system is its high-level, abstract visual language, *cantata*. These basic facilities have been used to build a set of applications for performing image processing research, algorithm development, and data visualization.

Khoros is available via anonymous ftp at no charge from *pprg.eece.unm.edu* in the */pub/khoros* directory. More information about Khoros can be obtained by sending mail to *khoros-request@chama.eece.unm.edu*.

Thresholding in Khoros

The thresholding algorithm supplied with Khoros can either be embedded into an application or can be run as a stand-alone program. In either case, the image to be

thresholded must first be converted into a format that is supported by Khoros. Khoros also supplies tools to convert from several popular image formats into the native Khoros image format. If thresholding is done using the stand-alone application, then it is invoked in the following manner:

vthresh -i <input image> -o <output image> -l <lower> -u <higher> -v <non-zero pixel output value>

vthresh generates a binary image by thresholding the input image. The output pixel value is given the specified non-zero value if: i) only the -l flag is given and the pizel value is less than the upper threshold, ii) only the -u flag is given and the pixel value is between the upper and lower threshold, iii) the -u and -l flags are given and the pixel value is between the upper and lower threshold, or iv) same as (iii), but both thresholds are equal and the pixel value is the same as the threshold level. Otherwise, the output pixel is given a zero value.

By choosing the appropriate limits it is possible to identify areas in an image that are above or below a certain value, or to isolate certain bands of intensities.

Edge Detection in Khoros

The edge detection algorithm supplied with Khoros can either be embedded into an application or can be run as a stand-alone program. In either case, the image to be processed must first be converted into a format that is supported by Khoros. Khoros also supplies tools to convert from several popular image formats into the native Khoros image format. If edge detection is done using the stand-alone application, then it is invoked in the following manner:

vdrf -i <input image> -o <output image>

vdrf encodes an algorithm invented by the team of Professor Serge Castan[1]. They argue that the edge detection filter that they have designed is superior to existing edge detection filters such as that designed by J.F. Canny [Canny 83]. Their argument is reproduced below. In the following paragraphs, "we" referes to Professor Castan and his team.

Edge detection is one of the most important subjects in image processing, which finds wide applications in pattern recognition, scene analysis and 3D vision, because the edge correspond in general to the important changes of physical or geometrical properties of objects in the scene and they are widely used as primitives in pattern recognition, image matching, etc.

The edges coincide, generally speaking, with a gray level transition, so they can be detected by maxima of gradient or the zero-crossing of the second derivatives calculated by some differential operators. Because differential operators are sensitive to noise, pre-processing such as smoothing is often necessary to eliminate the noise.

---

1. Professor Serge Castan
IRIT, URA 1399
118, Route de Narbonne 31602
Toulouse, France

One well-known smoothing filter is the Gaussian filter, and therefore edges can be detected by a Laplacian-Gaussian filter. But, there is an essential difficulty of the Laplacian-Gaussian filter, which is the contradiction between the smoothing effect and the precision of edge localization. To overcome this difficulty, we propose the optimal linear filter based on one step model (a step edge and white noise) and the multi-edge model. This optimal smoothing filter is a symmetric exponential filter of infinitely large window size and can be realized by a very simple recursive algorithm.

It is proved that the band limited Laplacian of an input image filtered by this filter can be computed from the Difference between the inout and the output of this Recursive Filter (DRF). The edges detected by the DRF method are less noisy and with a much better precision of localization. The theoretical analysis of the performance of the filters shows that the exponential filter is superior to other current filters [Shen 86] [Shen 87] [Shen 88].

Appendix II

CLIPS

In the system that is described in this thesis, the layout reasoning module has been entirely implemented in the CLIPS environment. In this appendix, we provide a description of CLIPS. The description of CLIPS has been reproduced verbatim from the documentation that is part of the CLIPS Version 5.1 distribution.

About CLIPS

The origins of the C Language Integrated Production System (CLIPS) date back to 1984 at NASA's Johnshon Space Center. At this time, the Artificial Intelligence Section (now the Software Technology Branch) had developed over a dozen prototype expert system applications using state-of-the-art hardware and software. However, despite extensive demonstrations of the potential of expert systems, few of these applications were put into regular use. This failure to provide expert systems technology within NASA's operational computing constraints could largely be traced to the use of LISP as the base language for nearly all expert system software tools at that time. In particular, three problems hindered the use of LISP based expert system tools within NASA: the low availability of LISP on a wide variety of conventional computers, the high cost of state-of-the-art LISP tools and hardware, and the poor integration of LISP with other languages (making embedded applications difficult).

The Artificial Intelligence section felt that the use of a conventional language, such as C, would eliminate most of these problems, and initially lookedto the expert system tool vendors to provide an expert system tool written using a conventional language. Although a number of tool vendors started converting their tools to run in C, the cost of each tool was still very high, most were restricted to a small variety of computers, and the projected availability times were very discouraging. To meet all of its needs in a timely and cost effective manner, it became evident that the Artificial Intelligence Section would have to develop its own C based expert system tool.

Originally, the primary representation methodology in CLIPS was a forward chaining rule

language based on the Rete algorithm [Forgy 82], and hence the Production System part of the CLIPS acronym. Version 5.0 of CLIPS, released in the spring of 1991, introduced two new programming paradigms: procedural programming (as found in languages such as a C and Ada) and object-oriented programming (as found in languages such as Common Lisp Object System and Smalltalk). The object-oriented programming language provided within CLIPS is called the CLIPS Object-Oriented Language (COOL).

Because of its portability, extensibility, capabilities, and low-cost, CLIPS has received wide-spread acceptance throughout the government, industry, and academia. The development of CLIPS has helped to improve the ability to deliver expert system technology throughout the public and private sectors for a wide variety of applications and diverse computing environments. CLIPS is being used by over 3300 users throughout the public and private community including: all NASA sites and branches of the military, numerous federal bureaus, government contractors, 170 universities, and many companies. CLIPS is available at a nominal cost through COSMIC, the NASA software distribution center. More information about CLIPS is available from COSMIC by calling *(706) 542 3265.*

Appendix III

DeltaBlue

In order to satisfy inferred constraints, we have embedded a freely available constraint solver into the system that is described in this thesis. This constraint solver is an implementation of the DeltaBlue algorithm invented by Bjorn Freeman-Benson, John Maloney and Alan Borning [Freeman-Benson90]. In this appendix, we provide a description of the DeltaBlue algorithm. The description that follows has been created using the description of DeltaBlue in [Freeman-Benson 90].

About DeltaBlue

In interactive applications, the constraint hierarchy often evolves gradually, a fact that can be exploited by an *incremental* constraint satisfier. An incremental constraint satisfier maintains an evolving "current solution" to the constraints. As constraints are added to and removed from the constraint hierarchy, the incremental satisfier modifies this current solution to find a solution that satisfies the new constraint hierarchy.

The DeltaBlue algorithm is an incremental version of the Blue algorithm. In Blue and DeltaBlue, each constraint has a set of methods that can be invoked to satisfy the constraint. For example,

```
c = a + b
```

has three methods:

```
c <= a + b        b <= c - a        a <= c - b
```

Therefore, the task of DeltaBlue is to decide which constraints should be satisfied, which method should be used to satisfy each constraint, and in what order those methods should be invoked.

The DeltaBlue algorithm uses three categories of data: the constraints in the hierarchy *C*, the constrained variables *V*, and the current solution *P*. *P*, also known as a *plan*, is a directed acyclic dataflow graph distributed throughout the constraint and variable objects: each variable knows which constraint determines its value, while each constraint knows if it is

currently satisfied, and if so, which method it uses.

The initial configuration is $C = \Phi$ and $V = \Phi$ (i.e., no constraints and no variables). The client program interfaces with the DeltaBlue solver through four entry points:

add_constraint	**add_variable**
remove_constraint	**remove_variable**

Variables must be added to the graph before constraints using them can be defined. Removing a variable removes any extant constraints on that variable as well. The current solution *P* is incrementally updated after each **add_constraint** and **remove_constraint** operation.

The key idea behind DeltaBlue is to associate sufficient information with each variable to allow the algorithm to predict the effect of adding a given constraint by examining only the immediate operands of that constraint. This information is called the *walkabout strength* of the variable, defined as follows:

> Variable *v* is determined by method *m* of constraint *c*. *V*'s walkabout strength is the minimum of *c*'s strength and the walkabout strengths of *m*'s inputs.

One can think of the walkabout strength as the strength of the weakest "upstream" constraint, i.e., the weakest ancestor constraint (in the current solution *P*) that can be reached via a reversible sequence of constraints. Thus, the walkabout strength represents the strength of the weakest constraint that could be revoked to allow another constraint to be satisfied. Freeman-Benson et. al., describe the use of walkabout strengths in adding and removing constraints.

In many applications, including the one described in this thesis, the same constraint hierarchy may be solved repeatedly. When the grid is initially inferred by the system, we create a constraint hierarchy that describes the manner in which the grid variables are changed when one of the grid columns is resized. Then, if the grid column is resized interactively by grabbing and moving its right edge with a mouse, the same constraint hierarchy is used to repeatedly recompute the related grid variables.

A constraint-driven algorithm, such as DeltaBlue, builds its dataflow graph solely from the hierarchy. Data-driven algorithms build the dataflow from both the hierarchy and the current values in the variables. Building the dataflow solely from the hierarchy has a performance advantage when the same hierarchy is used repeatedly because the dataflow can be chached and reused without the delay of reinvoking the constraint solver.

Appendix IV

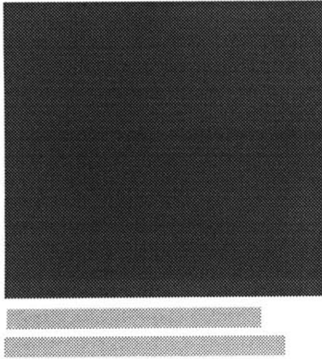
Relational Predicates

Relational predicates are used to define group templates that are then used to match high-level functional groups within the layout. In this Appendix, we describe these relational predicates, and show some examples of their use.

(above object1 object2)	Is true when object1 is above object2.
(below object1 object2)	Is true when object1 is below object2.
(left-of object1 object2)	Is true when the right edge of object1 is to the left of the left edge of object2.
(right-of object1 object2)	Is true when the left edge of object1 is to the right of the right edge of object2.
(left-aligned object1 object2)	Is true when the left edges of object1 and object2 are approximately aligned. The left edges are considered to be aligned even when they are spaced apart by a heuristically derived threshold amount.
(right-aligned object1 object2)	Is true when the right edges of object1 and object2 are approximately aligned. The right edges are considered to be aligned even when they are spaced apart by a heuristically derived threshold amount.
(top-aligned object1 object2)	Is true when the top edges of object1 and object2 are approximately aligned. The top edges are considered to be aligned even when they are spaced apart by a heuristically derived threshold amount.

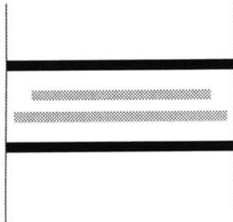
(bottom-aligned object1 object2)	Is true when the bottom edges of object1 and object2 are approximately aligned. The bottom edges are considered to be aligned even when they are spaced apart by a heuristically derived threshold amount.
(equal-width object1 object2)	Is true when object1 and object2 are approximately the same width. The widths are allowed to be different by a heuristically derived threshold amount.
(equal-height object1 object2)	Is true when object1 and object2 are approximately the same height. The heights are allowed to be different by a heuristically derived threshold amount.
(vertically-aligned object1 object2)	Is true when the vertical centers of object1 and object2 are within a heuristically derived threshold amount.
(horizontally-aligned object1 object2)	Is true when the horizontal centers of object1 and object2 are within a heuristically derived threshold amount.
(vertically-adjacent object1 object2)	Is true when object1 is above object2, and there are no other objects in the vertical space between them.
(horizontally-adjacent object1 object2)	Is true when object1 is to the left of object2, and there are no other objects in the horizontal space between them.
(grid-left object)	Is true if object is left aligned with respect to the grid columns it spans.
(grid-right object)	Is true if the object is right aligned with respect to the grid columns it spans.
(grid-centered object)	Is true if the object is centered with respect to the grid columns it spans.
(grid-justified object)	Is true if the object is justified with respect to the grid columns it spans.

The following examples illustrate the use of these relational predicates to match higher-level functional groups of objects in the layout.



Shown on the left is the group that was matched by the following query:

```
(do-for-all-instances
 ; define the query template
 ((?o1 graphic-object) (?o2 textblock))
 ; define the query
 (and   (vertically-adjacent ?o1 ?o2)
        (left-aligned ?o1 ?o2)
        )
 ; perform the action
 (make-group Feature ?o1 ?o2)
)
```



Shown below is another way of matching *by-line* groups by using grid relationships to construct the query.

```
(do-for-all-instances
 ; define the query template
 ((?o1 h-rulebar ?o2 textblock ?o3 h-rulebar))
 ; define the query
 (and   (equal-width ?o1 ?o3)
        (vertically-adjacent ?o1 ?o2)
        (vertically-adjacent ?o2 ?o3)
        (grid-centered ?o2)
        )
 ; perform the action
 (make-group Byline ?o1 ?o2 ?o3)
)
```