1995

Structural interpretation of line diagrams

Mir Hoseyn Dezfulian

University of Wollongong

Recommended Citation

NOTE

This online version of the thesis may have different page formatting and pagination from the paper copy held in the University of Wollongong Library.

UNIVERSITY OF WOLLONGONG

COPYRIGHT WARNING

You may print or download ONE copy of this document for the purpose of your own research or study. The University does not authorise you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site. You are reminded of the following:

Copyright owners are entitled to take legal action against persons who infringe their copyright. A reproduction of material that is protected by copyright may be a copyright infringement. A court may impose penalties and award damages in relation to offences and infringements relating to copyright material. Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.
I hereby declare that I am the sole author of this thesis. I also declare that the material presented within is my own work except where duly acknowledged.

Mir Hoseyn Dezfulian

March 1995
Structural Interpretation of Line Diagrams

A thesis submitted in fulfilment of the requirement for the award of the degree of Doctor of Philosophy

from
The University of Wollongong

Advisor: Assoc. Prof. Neil A. B. Gray

by
Mir Hoseyn Dezfulian, MSc.
Department Of Computer Science

March 1995
Abstract

Much of the information contained in scientific documents comes in the form of graphs and line diagrams such as circuit diagrams or chemical structure diagrams. If the contents of such documents are to be properly represented in computerized information systems, then these diagrammatic data must somehow be converted to computer-manipulable and searchable form.

In general, it is possible to convert structural diagrams into "graphs" with labelled nodes and edges. Graphs are easily represented as data structures within computer programs and searches of collections of such data are possible using graph matching ("isomorphism") algorithms.

This thesis addresses the general problem of converting, into such computer manipulable graphs, the data presented in structural diagrams on scanned images of pages taken from scientific documents.

The interpreter system developed as part of this thesis uses a multi-step process to extract information from scanned images. The original image data are transformed through several preprocessing steps to obtain a representation in terms of line segments and arcs. These data are then processed using a general purpose matching system that uses "templates" which define those groupings of graphic elements that are significant within a particular domain. These groupings become the nodes of the graph. Other elements extracted from the image become the edges.

In domains where line diagrams are used extensively, there are specific grouping of lines and arcs that are semantically meaningful: For example, a combination of parallel lines and complex arcs that represents a transformer. These groupings, or "templates", are specified by a domain expert, and are stored in a dictionary. Each template represents a standard component as used in a particular problem domain, and is defined in terms of a finite set of primitives and their topological relationships.

We have implemented and tested the idea of templates on different types of diagrams including circuit diagrams, chemical structures, flowcharts, and even cursive script writings. Although these sorts of diagram are very different in appearance and application, they could all be interpreted by defining appropriate templates.

The thesis presents the results for a variety of applications and reviews some of the limitations of the approach taken.
Acknowledgement

I would like to express my deep sense of gratitude to my supervisor Assoc. Professor Neil A. B. Gray for his guidance, advice, support, and encouragement during the course of this research.

I would also like to thank Mr Derek Hanley for his moral support.
Dedication

To my wife
Contents

Abstract ..........................................................................................................................iii
Acknowledgement ........................................................................................................iv
Dedication .......................................................................................................................v
Contents .........................................................................................................................vi
List of Figures ................................................................................................................x

1. Introduction ...............................................................................................................1
   1.1. Motivation .........................................................................................................1
   1.2. Objectives .......................................................................................................4
   1.3 Thesis Organization ...........................................................................................4

2. Literature Survey .....................................................................................................7
   2.1. Introduction .......................................................................................................7
   2.2. What Is Involved? ...........................................................................................9
      2.2.1. Data Capture and Preprocessing ...............................................................10
      2.2.2. Region Segmentation ...............................................................................12
      2.2.3. Extraction and Description of Lines and Solid Regions .........................13
      2.2.4. Vectorization and Line Feature Extraction ............................................15
      2.2.5. Recognition of Graphics .........................................................................16
   2.3. Existing Diagram Interpretation Systems .......................................................21
      2.3.1. Circuit Diagrams and Electronic Logic Diagrams ..................................21
      2.3.2. Chemical Structure Diagrams .................................................................25
4.2. Definitions ................................................................. 75
  4.2.1. Attributed Relational Graphs and Structures ............... 76
  4.2.2. Primitives ............................................................ 78
  4.2.3. Relations ............................................................. 83
  4.2.4. Correlations ........................................................ 95
  4.2.5. Templates ............................................................ 96
4.3. System Overall Structure ............................................. 99
4.4. Matching ................................................................. 104

5. Experimental results ..................................................... 111
  5.1. Introduction ............................................................ 111
  5.2. Circuit Diagrams Interpretation .................................. 112
    5.2.1. Symbols and Templates ........................................ 112
    5.2.2. Interpreted Example ........................................... 114
    5.2.3. Problems .......................................................... 121
  5.3. Flowchart Interpretation ........................................... 123
    5.3.1. Symbols and Templates ........................................ 123
    5.3.2. Interpreted Example ........................................... 125
    5.3.3. Problems .......................................................... 129
  5.4. Chemical Structure Diagram Interpretation ..................... 130
    5.4.1. Symbols and Templates ........................................ 130
    5.4.2. Interpreted Example ........................................... 133
    5.4.3. Problems .......................................................... 136
  5.5. Cursive Script Writing Interpretation ........................... 136
    5.5.1. Symbols and Templates ........................................ 136
    5.5.2. Interpreted Example ........................................... 140
    5.5.3. Problems .......................................................... 141
6. Conclusions .................................................................................................................. 143

6.1. Future Work ............................................................................................................... 144

BIBLIOGRAPHY .............................................................................................................. 147
List of Figures

Fig. 2.1 Traditional Expert System Model. .............................................. 18
Fig. 2.2 Example of a basic Blackboard Model. ........................................ 19
Fig. 2.3 The Blackboard Model. ............................................................... 19
Fig. 2.4 A part of a circuit diagram. ...................................................... 23
Fig. 2.5 Sample of a Decision Tree. ...................................................... 23

Fig. 3.1 Image with nine pixels. ............................................................... 34
Fig. 3.2 Addition Algorithm for Two Images. .......................................... 35
Fig. 3.3 Multiplication Algorithm for Two Images. ................................. 36
Fig. 3.4 Finding Maximum of Two Images. ............................................ 36
Fig. 3.5 Finding Minimum of Two Images. ............................................. 36
Fig. 3.6 Cutting an Image from another one. .......................................... 37
Fig. 3.7 90 Degree Rotation Algorithm.................................................. 38
Fig. 3.8 180 Degree Rotation Algorithm................................................ 38
Fig. 3.9 Reflection Algorithm. ................................................................. 38
Fig. 3.10 Examples of MIN and EXTMAX operations.............................. 39
Fig. 3.11 Examples of structural elements.............................................. 39
Fig. 3.12 Dilation Algorithm................................................................. 40
Fig. 3.13 Example of Dilation operation................................................ 40
Fig. 3.14 Erosion Algorithm................................................................. 41
Fig. 3.15 Example of Eroding Operation................................................ 41
Fig. 3.16 Example of Opening Operation.............................................. 42
Fig. 3.17 Opening Algorithm................................................................. 42
| Fig. 3.18 | Example of Closing Operation | 43 |
| Fig. 3.19 | Closing Algorithm | 43 |
| Fig. 3.20 | Original Schematic Diagram | 44 |
| Fig. 3.21 | Schematic Diagram Dilated by a 2X2 Structural Element | 45 |
| Fig. 3.22 | Line Diagram Closed, using a 2X2 Structural Element | 45 |
| Fig. 3.23 | Eroded Schematic Line Diagram | 46 |
| Fig. 3.24 | Opened Schematic Diagram by a 2X2 Structural Element | 46 |
| Fig. 3.25 | Filling in voids, via CLOSE operation | 47 |
| Fig. 3.26 | Contour Tracing via Cutting Eroded Image from the image | 48 |
| Fig. 3.27 | Chain code definition for the eight neighbours of pixel p | 49 |
| Fig. 3.28 | Break point configurations | 49 |
| Fig. 3.29 | Left edge safe point configurations | 50 |
| Fig. 3.30 | Thinning Algorithm | 51 |
| Fig. 3.31 | Thinned Image of the Circuit Diagram | 52 |
| Fig. 3.32 | Example image for discussion of line finding | 53 |
| Fig. 3.33 | Structure used for line following | 54 |
| Fig. 3.34 | Small horizontal stairs on a line segment | 54 |
| Fig. 3.35 | Algorithm for finding contiguous points of thinned image | 55 |
| Fig. 3.36 | Different parts of a circle to be verified | 56 |
| Fig. 3.37 | A line segment, and its chain code | 58 |
| Fig. 3.38 | Vectorization Algorithm | 59 |
| Fig. 3.39 | Two Connected arcs with a straight line segment | 61 |
| Fig. 3.40 | Two Connected arcs via straight line segments | 62 |
| Fig. 3.41 | Arc Detection Algorithm | 62 |
| Fig. 3.42 | Circle Finding Algorithm | 63 |
| Fig. 3.43 | Triangle Finding Algorithm | 64 |
| Fig. 3.44 | Blackboard model used for circuit diagrams' interpretation | 65 |
Fig. 3.45  Symbols used to show Capacitor.................................66
Fig. 3.46  Perception Capacitors Algorithm.................................67
Fig. 3.47  Symbols used to show OP-Amps..................................67
Fig. 3.48  Amplifier Perception Algorithm..................................67
Fig. 3.49  Symbols used for Resistors. ..................................68
Fig. 3.50  Resistor Perception Algorithm.................................68
Fig. 3.51  Algorithm to find connections..................................70
Fig. 3.52  Sample part of a schematic diagram............................71
Fig. 3.53  Preprocessed image of the sample diagram....................71
Fig. 3.54  Elements found in the diagrams and their connections....72
Fig. 3.55  Connection table of the schematic diagram....................72
Fig. 3.56  Distortion makes unexpected results............................73

Fig. 4.1  Straight line segment and its attributes............................78
Fig. 4.2  Digital line segment and its nearest and furthest support....79
Fig. 4.3  Defined "start" and "end" points for the straight line segments.....................80
Fig. 4.4  Circle primitive and its attributes..................................80
Fig. 4.5  Circular arc and its attributes......................................81
Fig. 4.6  Structure for declaration elements of a diagram................82
Fig. 4.7  Structure to declare the relation between two elements....83
Fig. 4.8  Cases of "parallel" relation with line segment..................84
Fig. 4.9  Arc "parallel" cases to other primitives..........................85
Fig. 4.10 "Parallel" relation between circles.................................85
Fig. 4.11 Line segment "contacted" other primitives........................86
Fig. 4.12 "Contact" of arc and other primitives..............................87
Fig. 4.13 "Contacted" circles..................................................88
Fig. 4.14 Line and arc with "bicontact" relation.............................88
Fig. 4.15  "Bicontact" relation with circle. ........................................... 88
Fig. 4.16  Some cases of "tjunc" relation. ........................................... 88
Fig. 4.17  "Tjunc" relation of arc with line or arc. .............................. 89
Fig. 4.18  Line and arc in "tjunc" relation with circle. ......................... 89
Fig. 4.19  "Cross" relation cases of lines and arcs. .............................. 90
Fig. 4.20  "Crossed" circle with line or arc. ..................................... 90
Fig. 4.21  "Above" relation between two primitives. ............................ 90
Fig. 4.22  "Not above" relation between two primitives....................... 91
Fig. 4.23  Some cases of "right" relation. ....................................... 91
Fig. 4.24  primitives that are "not right" of the others. ....................... 92
Fig. 4.25  "Continuation" relation. ................................................ 92
Fig. 4.26  "Inclusion" relation with circles. .................................... 93
Fig. 4.27  "Inclusion" relation with arcs. ....................................... 93
Fig. 4.28  "Midpoint close" relation of arcs and lines. ....................... 93
Fig. 4.29  "Midpoint Close" to circle. ............................................. 94
Fig. 4.30  Ambiguity in relation definition. .................................... 95
Fig. 4.31  Structure for declaring correlation of relations. .................. 95
Fig. 4.32  Template structure. .................................................... 96
Fig. 4.33  Connection structure.................................................... 97
Fig. 4.34  Example of a triangle template. ...................................... 97
Fig. 4.35  Two interpreted instances of triangle template..................... 98
Fig. 4.36  Example of a resistor template. ...................................... 98
Fig. 4.37  An instance of a resistor.............................................. 99
Fig. 4.38  System’s Flow Diagram. ................................................ 100
Fig. 4.39  Sample document page. ............................................... 101
Fig. 4.40  Segmented document page with "W=1" and "H=1"...................... 102
Fig. 4.41  Segmented document page with "W=9" and "H=5"..................... 102
Fig. 4.42  Arc shape parts of an AND gate is vectorized as line......................... 104
Fig. 4.43  Matching Algorithm. ........................................................................... 106
Fig. 4.44  A defined template for amplifier............................................................ 109
Fig. 4.45  An amplifier that would be interpreted in different directions. .......... 109
Fig. 4.46  Example of wrongly matched templates.................................................. 110

Fig. 5.1  Symbols considered for schematic diagrams interpretation..................... 112
Fig. 5.2  Example of a resistor templates. ................................................................. 113
Fig. 5.3  Sample document containing a schematic line diagram.......................... 114
Fig. 5.4  Text parts of diagram segment removed and contours are traced .......... 115
Fig. 5.5  Thinned image of the diagram segment.................................................... 116
Fig. 5.6  Found symbols of the diagram with their links marked............................ 116
Fig. 5.7  Thinned image of the diagram segment.................................................... 117
Fig. 5.8  Interpreted diagram part of the Fig. 4.39................................................... 118
Fig. 5.9  Connection table of interpreted diagram (Fig. 5.8). ................................ 118
Fig. 5.10 Another circuit diagram to be interpreted.............................................. 119
Fig. 5.11 Annotated text are removed from Fig. 5.10.......................................... 119
Fig. 5.12 Thinned image of circuit diagram of Fig. 5.11.......................................... 120
Fig. 5.13 Rebuilt circuit ....................................................................................... 120
Fig. 5.14 Connection table of the circuit............................................................... 121
Fig. 5.15 A circuit diagram that contains 8 symbols............................................. 121
Fig. 5.16 Incomplete interpreted diagram of Fig. 5.15 (6 out of 8 symbols) ....... 122
Fig. 5.17 More incomplete interpreted circuit diagram (3 out of 5 symbols)...... 122
Fig. 5.18 Flowchart Symbols.............................................................................. 123
Fig. 5.19 A sample template defined for "start box"............................................. 124
Fig. 5.20 Misinterpretation of symbols................................................................. 124
Fig. 5.21 Some templates defined for "decision box"........................................... 125
Fig. 5.22  Sample document containing a flowchart diagram.......................... 126
Fig. 5.23  Cleaned up, and thinned image of the diagram......................... 127
Fig. 5.24  Found symbols from the flowchart image and their links........... 128
Fig. 5.25  Connection table of the found flowchart symbols..................... 129
Fig. 5.26  Some symbols used in chemical formulas................................. 131
Fig. 5.27  Template examples defined for chemical structures interpretation. . 132
Fig. 5.28  Sample document page containing chemical structures............... 133
Fig. 5.29  A chemical structure diagram segmented from Fig. 5.28............ 134
Fig. 5.30  Preprocessed sample diagram........................................... 134
Fig. 5.31  Rebuilt chemical structure............................................... 135
Fig. 5.32  Connection table of found atoms and their bonds..................... 135
Fig. 5.33  Different cases of some Arabic/Persian characters.................... 138
Fig. 5.34  Some templates for Arabic/Persian characters recognition.......... 139
Fig. 5.35  Sample document of Arabic/Persian text................................ 140
Fig. 5.36  Preprocessed image of the Fig. 5.35.................................. 140
Fig. 5.37  Some characters ("Mim" and "Daal") are recognized and marked....... 141
Fig. 5.38  Some characters ("Kaaf" and "Gaaf") are recognized and marked...... 141
1. Introduction

1.1 Motivation

The maintenance of information, traditionally confined to Journals and books, can be vastly expensive and wasteful of natural resources. The contemporary solution is to utilise an electronic medium — computerisation. Once information has been converted to a computer compatible form it becomes much easier to access than printed documents. If entry into the computerized library systems require the retyping of documents, it would generally be too costly. But in most cases, text can be processed with Optical Character Recognition (OCR). Modern OCR systems can read a variety of printed materials and produce temporary files that require relatively little editing before they are used to build the files of a computerized data library. There are numerous systems that utilise keyword searching of collections of full text articles.

Unfortunately, many scientific, engineering, and technical disciplines use structural diagrams as a primary means of conveying information. Obvious examples include chemical structural diagrams as used by organic chemists, circuit diagrams as used by electrical engineers and the technical drawings of civil engineers and architects. In these domains, journal articles, books, and reports all rely extensively on structural diagrams.
While the diagrams of chemical structures and architectural drawings differ greatly, they and other types of line diagrams can be reduced to "graphs" of nodes and edges. These graphs can be stored in computer files and be scrutinised by search engines that employ graph isomorphism (graph matching) algorithms.

If scientific and technical articles are to be properly represented in computerized data libraries, they will require both their text and diagrams ("graphs") to be present. The text can be captured by OCR, but the diagrams still present a problem.

One solution that has been attempted involves the use of special purpose editor programs. Articles are entered into the data system as scanned text, then a skilled operator enters the diagrammatic data using a special editor program. Companies such as Molecular Design and Beilstein have editor based systems that can be used to enter details of organic chemical reactions. These systems are used by clients to build libraries containing records of chemical syntheses that have appeared in journal articles. These records can then be searched when a chemist needs suggestions on how to perform a particular synthetic reaction. However, the editing of structures for data entry is much more costly and requires greater skill than the simple retyping of text.

A system designed for the automatic conversion of diagrams to graphs has many potential applications. Such a system would involve a variety of components. It would require low-level image processing routines that perform tasks like cleaning up "noise" in a scanned image and separating the "diagram" from surrounding (and embedded) text. Other components would analyze the bit-map image that results from the scanning process and then convert it to a higher level representation involving lines, arcs, and so on. These would then have to be grouped.
Some of the lines would become the graph's edges and other features in the image would become the nodes. This part of the processing depends on the application's domain.

The image features corresponding to "gates" for a logic circuit are quite different from the features used to represent capacitors, resistors etc in an electrical circuit. Each domain has its own repertoire of standard parts like the arcs and lines that make a gate, or the parallel lines of a capacitor, or the special junctions of groups of lines that are meaningful in engineering drawings.

Although the domains differ, they have something in common; they all involve little groups of arcs and lines that if found together in a particular arrangement signal the presence of a particular kind of node. Such groupings could be defined externally to the interpreter system and held in data files specific to a particular application domain. Each grouping, defining some kind of node, would identify arcs and lines together with any constraints on their interconnections. The interpreter system could work by loading the appropriate "nodes" file. There are usually further constraints involving how "nodes" may be interconnected; these additional constraint relations (used to verify edges) can also be specified in the files created for each problem domain.

The motivation for this thesis is to create a "general purpose" diagram interpreter which will combine low-level image processing, feature extraction, and a component matcher for the components defined for a particular domain.
1.2 Objectives

The objectives are as follows:

- to implement all the low-level image processing routines needed for the handling ("cleaning up") of scanned images from journal articles, and books;

- to discover and implement algorithms that convert a "cleaned up" bit-map image into a more abstract, higher level representation that involves lines and arcs;

- by considering a wide variety of application domains, to establish the range of different constraints that are required to describe the groupings of arcs and lines that are to be recognized;

- to implement a matching system to identify these patterns in an image;

- to create an interpretation system with sufficient flexibility to work with real image data taken from a variety of domains.

1.3 Thesis Organization

Chapter 2 reviews existing work. This chapter also summarizes the essential preprocessing steps. These steps include scanning, segmentation, contour tracing, thinning and vectorization. Some previously proposed schemes for diagram interpretation systems are reviewed, and the idea of "blackboard" architecture is briefly discussed. This chapter concludes with a section summarizing the existing experimental use of interpretation systems for circuit diagrams, chemical structure diagrams, engineering drawings and some other forms of mapping.
Chapter 3 starts with the analysis and implementation of image preprocessing algorithms (for example segmentation, contour tracing, thinning, and so on). Some of these algorithms are refined versions of existing algorithms from the literature or are newly developed by the author. Next, the chapter covers the design and implementation of a system that uses a "blackboard" model for the interpretation of schematic line diagrams of simple electrical circuits. This special purpose interpretation system was constructed to exercise all the low-level image analysis components that had to be developed, and to provide a base from which to explore more general interpretation schemes. Some outputs of the simple interpreter are shown at the end of chapter, and the weak points of the interpreter are discussed.

Chapter 4, presents a generalized line diagram interpreter which utilises special templates. These "templates" define the constrained groups of arcs and lines that are to be recognized as representing the nodes for a particular domain. "Templates" are defined for components such as a diode in a circuit diagram, a decision box in a flowchart, and so on. Different domains tend to have different kinds of constraints in regards to how lines and arcs are interrelated. A large section of this chapter details the various kinds of constraints that have been identified as necessary for the definition of flexible templates. Later sections of this chapter present the overall structure of the interpretation system and discuss the algorithms used for finding and matching defined "templates" with the "primitives" (ie. straight line segments, circular arcs, etc.) found in an image.

The general interpreter system introduced in chapter 4 has been tested with different data types and the results are summarized in chapter 5. The application domains — electrical engineering, chemistry, flowcharts and Arabic/Persian character recognition — are deliberately diverse so as to explore the range of possibilities for this kind of interpreter. Chapter 5 has a number of sections. Each illustrates a number of
templates for a particular domain together with an example image and its interpretation. The specific problems arising during the interpretation processes are discussed at the end of each section of this chapter. Parts of this work were presented briefly in a paper given at AI94, the 7th Australian Joint Conference on Artificial Intelligence.

Chapter 6 summarizes the achievements of this thesis and reviews some of its practical limitations. Some areas for further work are suggested.
2. Literature Survey

2.1 Introduction

Today most printed text can be entered reliably and economically into computers. There are numerous machines in the market that are able to do this, but most of these machines are not able to handle illustrations. At best, they can identify illustrations and skip those parts of a page where these are printed. For many applications (e.g. patent data bases) it is desirable to include diagrams together with the text. Therefore there is a need for the recognition of drawings in order to store their information in computer processable form, for later searching as part of data bases. For example a user might want to list all documents that contain chemical diagrams incorporating a particular substructure, or to list all papers that contain diagrams with a particular circuit structure as a component.

This work is based on considerations of general problems related to patent libraries, the construction of chemical structures and reaction libraries, engineering drawings, circuit diagrams, and so forth. Being graphical in nature, these drawings represent complex information in a concise manner. Surveys [Karima et al. 85] have indicated that a typical engineering project of reasonable magnitude could contain well over 30,000 drawings as part of its documentation, with design lives ranging from about 10 to 40
years. Recent studies [Pferd 84] have shown that there are over 2 billion active engineering drawings in the U.S. alone. There are also very large amounts of printed materials, in journal articles, books and patent listings that must somehow be entered into a data-retrieval system. Most companies file a large number of engineering drawings in their vaults, and roughly 20% of these are active each year. One factor that has slowed down the introduction of Computer Aided Design in some industries is the existence of such large sets of paper documents that must be integrated into any new system (for example engineering drawings into CAD style formatted files). A large number of personnel hours are expended in creating, updating, and maintaining these drawings using conventional drafting techniques, however their manipulation is still incredibly complex.

Although CAD has become more and more important, the initial design of circuit and logic diagrams is carried out by experienced engineers. Manually drawn circuit diagrams are still used in practice as important means for communication. Therefore, there is a need to extract relevant data from a given diagram and store it in a computer in such a way that could be easily manipulated later.

Many journals are already available in computer readable form, and could be manipulated by any sort of data-retrieval software. Unfortunately, in all the areas mentioned above, significant data is often presented in diagrammatic form and such data cannot be handled by existing software. Such pictorial information must be interpreted to form a reasonable structure, which is acceptable by a computer for later processing. For example chemical diagrams printed in journal articles have to be converted into canonicalized connection table representations that could be used for sub-graph matching and for other forms of structural query. Another example would be engineering drawings; these must be interpreted so that they can be entered in CAD systems and subsequently manipulated. This is still a current area of activity.
2.2 What Is Involved?

Whilst the recognition of diagrams may be viewed as an extension of recognition of text, it is much more difficult because it must reproduce both names and descriptions. For example, "A circular arc of radius 2.5 cm and angle of 36 degrees". In some cases it may require an even higher recognition level such as the symbol for a logic circuit element. This usually comes as the last step of a series of processes. When dealing with paper input, attention must be paid to the following sequences of processes [Kasturi et al. 92]:

1. Data capture and preprocessing;
2. Region segmentation;
3. Extraction and Description of Lines and Solid Regions;
4. Vectorization and Feature extraction; and
5. Graphic recognition and interpretation.

Scanning, noise filtering and thresholding operations, which are included in the data capture and preprocessing step, convert a paper-based document to a reasonably noise-free binary image. These are discussed in section 2.2.1. A typical image of a document is a mixed image of text, graphics, and half-tone images. These should be segmented into separate regions to facilitate the application of appropriate algorithms to each group. For example, text regions can be recognized by an OCR system; graphics regions by a graphics interpretation system; and image segments by an image interpretation system. Segmentation algorithm are discussed in section 2.2.2. Graphics regions contain filled regions (for example arrowheads in engineering drawings, or symbols representing components such as capacitors in schematic circuit diagrams) and objects made up of thin lines. Thin lines should be represented by their core-lines and filled areas are represented by their contours(boundaries).
Dashed lines, should be tracked to generate a compact line description. Algorithms for these operations (vectorization; line thinning and following and contour tracing) are discussed in section 2.2.3. Since a line diagram is composed of straight lines and curves, the next step is to detect feature points such as corners and points of transition from straight lines to curves. The extraction of feature or critical points is discussed in section 2.2.4. The next step is to interpret the lines and curves by an appropriate semantic interpretation system. For example, a simple polygon recognition algorithm is adequate for processing a hierarchical organization chart. Various electrical components can be identified in an electric schematic diagrams by graph matching approaches. Complex knowledge-based algorithms are needed to interpret 2-D projections of 3-D objects. Section 2.2.5 is to discuss some of these techniques.

2.2.1 Data Capture and Preprocessing

Scanning

Scanners may be one of the three general types: drum type, flat-bed type or continuous roll feed type. Drum scanners contain a rotating drum on which the document is mounted and a sensor that captures the changes in the light intensity along a line of the document. Flat-bed type scanners use a linear array of light sensors mounted on a carrier. The carrier moves in the plane of the flat table containing the document. In continuous roll feed type scanners, the document is fed into the scanner at a controlled speed and a linear sensor array captures the intensity changes along the entire width of the document [Ejiri et al. 90].

An important parameter to be determined in selecting a scanner is its resolution. There is a trade-off between image fidelity and the memory required; higher resolution scanning yield data with greater fidelity but with much larger memory requirements. For example, changing the resolution from 200 dpi (dots per inch) (8 dots per mm) to
300 dpi (12 dots per mm) more than doubles the image size. Higher resolutions are
needed to achieve benefits such as the recognition of font and style rather than simply
character recognition. Higher resolution scans may also be needed to compensate for
poor quality in the original documents. For applications where features are larger and
less complex, such as for recognizing lines in a circuit diagram, it may be adequate to
use lower resolutions. For example, Okazaki et al. [85] use 80 dpi for line following
and 240 dpi for character recognition in a circuit diagram. Many scanners include the
thresholding operation, and output a binary image. Thresholding is applied separately
if the images are captured using video digitization or other data capture techniques that
output grey level images.

For documents with a good contrast against a uniform background, data capture is
straightforward. Binary scanners are available that combine scanning with thresholding
at a fixed threshold. However, many documents have a wide range of background and
object grey levels. For these, a fixed threshold applied to a grey level image often does
not generate images with a clear separation between the object and background. For
these cases it is best to (first) obtain grey-scale images from the scanning stage, which
will enable separate digital image processing methods to extract the binary information.

Holdermann and Kazmierczak [72], Peleg and Rosenfeld [79], Ting and Prasada [80],
Wang et al. [81], Weszka [78], Nagin et al. [82] all describe general pre-processing
techniques, entailing filtering and thresholding. The objective is to separate object
regions from background and noise. Due to variations in average intensities, spatially
adaptive thresholding is performed by analyzing grey-level values within local
windows across the image to determine local thresholds (Rosenfeld and Kak [82],
Casey and Wong [90]). These local neighbourhoods should be large enough to
guarantee that both foreground and background pixels are included, but not so large as
to average over non uniform background intensities. Ideally, the threshold should be
Insensitive to noise but sensitive to edges of characters and lines [Wong 78]. For more extensive treatment of thresholding techniques, Sahoo, et al. [88] contains a recent survey.

Preprocessing

Simple noise reduction filters are generally applied to the binary images data obtained from scanning and thresholding. In a simple noise reduction process, single pixel voids are filled and protrusions are deleted by iteratively passing a 3 x 3 pixel filter window over the image [Shih and Kasturi 88]. Morphological operations [Jain 89] such as erosion, dilation, open, close, and prune may also be applied to filter some of the digitization artefacts and noise. For most manually generated graphics, such as characters, it is desirable to remove noise, but retain corners. The choice of preprocessing filters and their parameters is clearly application dependent, since what is considered to be noise in one application is possibly a feature of another.

2.2.2 Region Segmentation

After binarization, the document image consists of regions of text, graphics and halftone images. Since different techniques are applied to process each of these, they are (usually) first segmented into these different regions. Srihari and Zack [86] and Casey and Wong [90] classify block segmentation algorithms into two types: top-down, or knowledge based methods, that work with a knowledge of the nature of the document; and bottom-up, or data driven methods, that continuously refine data by layered grouping operations. However, many of the practical algorithms use a combination of these two methods.
Fletcher and Kasturi [88] describe a Hough transform-based algorithm for isolation of text-strings from any mixed text/graphics image. (For a review of Hough transform techniques, their implementation, and performance see Illingworth and Kittler [88]) Since the Hough transform method is invariant to document skew, text strings in any orientation are detected. This algorithm has been enhanced to handle text strings that are connected to graphics [Gattiker 88].

The algorithm consists of five steps: connected component generation; preprocessing the components to eliminate those that are most likely graphics; grouping the remaining components using the Hough transform; logically grouping these strings into textual word and phrase groups; and post-processing these strings to further refine them. The final step is described in detail in Kasturi et al. [90].

2.2.3 Extraction and Description of Lines and Solid Regions

Solid Region Segmentation

After the region segmentation process has been carried out, a graphics region image with two major types of objects is the result:

- THIN line-like structures, representing various line, arcs, and text in the drawing and
- THICK regions or filled in areas corresponding to various special symbols; (for example arrowheads commonly found in dimensioned machine drawings; or diode symbols in schematic circuit diagrams, or company logos and so on)

The lines and the solid regions must first be differentiated. One approach to finding solid regions is by erosion and dilation operations (Harada et al. [85], Shah [88], Kasturi et al. [90]). In this approach, the image is eroded by a predetermined number of pixels to completely remove all thin entities such as lines. The eroded image is
dilated and a logical AND operation is performed to recover solid symbols. The contours for these solid areas can then be obtained, and these contour lines combined with the original line features present so as to obtain a representation of the original image in terms of thin lines and boundary contours.

Contour Tracing

Contour tracing finds the boundaries of the thick regions of an image. Those pixels of a thick region that have at least one neighbour which does not belong to the region form the boundary of that region. Contour tracing is a technique used to determine the sequence order among boundary pixels based on a counter-clockwise or clockwise traversal.

The most recent algorithm is that proposed by Liow [91]. His algorithm has the following advantages: a: it provides more detailed topological information in each contour; b: the implementation in a table lookup method is easier and more efficient; c: the complexity can be reduced to half because only one of each pair of neighbouring regions needs to be traced; d: parallel processing is feasible; and, e: the shapes of regions are well preserved.

Thinning the image

The lines in the image (after discrimination between THIN and THICK regions) may be a few pixels wide, and a thinning operation is necessary to reduce them to one pixel in width. The thinned pattern must however preserve the connectedness and shape of the original structure. Thinning usually consists of successively deleting dark pixels on the contour of the pattern until it is reduced to unit width. During any such iteration, a dark pixel that is deleted is required to satisfy the following criteria [Naccache and Shinghal 84]:

14
1. It is an edge point; i.e. it lies along the contour of the pattern.
2. It is not an end point; i.e. it does not lie on the extremity of a stroke.
3. It is not a break point; i.e. it is not a point whose removal will break the connectedness of the pattern.
4. The deletion of this point should not cause excessive erosion.

Several algorithms have been proposed for thinning patterns (Stefanelli and Rosenfeld [71], Deutsch [72], Arcelli et al. [75], Tamura [78], Woetzel [78], Pavlidis [82a], Nakayama et al. [84], Zhang and Suen [84], Kuehn et al. [85], Lu and Wang [85], Xia [89], Arcelli and Baja [85], Chen and Hsu [89], Eckhardt and Maderlechner [91] and [92], Sur and Datto [91], Kwok [92], Nagendraprasad et al. [92], Gibbons and Niblack [92]). They usually differ in their methods or tests applied to determine whether a dark pixel satisfies the above criteria. Lee et al [91] have evaluated the performance of some skeletonization (Thinning) algorithms and discussed both the weak and strong points of each algorithm.

### 2.2.4 Vectorization and Line Feature Extraction

The contour tracing and line thinning processes will have reduced the image of the line diagram to a thinned image where all line widths are at most one pixel. Vectorization is a technique for the extraction of "line structures" from this thinned image. Many different vectorization algorithms have been reported in the literature (Clement [81], Black et al. [81], Pavlidis [82b], Zen and Ozawa [85], Chakravarty [81], Shimisu [82], Takagi et al. [82], Kikkawata et al. [84], Tombre [85], Meynieux et al. [86]).

A simple but efficient algorithm for finding lines, starts by running through the image considering each raster line in turn from top to bottom. Each set pixel in a raster line is considered in turn and is either used to extend an existing chain of contiguous set
pixels, or is used to form the start of a new chain. This process converts the somewhat unwieldy bitmap representation of the data to a more amenable form — a list of lists (with each of these lists representing a chain of contiguous pixels, i.e. a "line structure").

The pixel chains must then be converted to achieve actual straight line segments. This is done in a second processing step in which the "critical points" or "break points" of the pixel chains are defined. The pixel chains are broken into short linear sequences at these break points. Some of the resulting short linear segments can then be recombined to get the straight lines of the original diagram. Other short linear sequences will form parts of digital approximations to curved arcs. The arcs have to be identified in a subsequent processing step.

Curved arcs can be approximated by polygons. In the iterative endpoint algorithm (Ramer [72], Duda and Hart [73]), a straight line segment is first connected between the endpoints of the digital line segment. The perpendicular distances from the segment to each point on the curve are measured. If any distance is greater than a chosen threshold, the segment is replaced by two segments each from a segment endpoint to the curve point where the distance to the segment is greatest. This process is iterated until all segments are within the threshold of the curve.

2.2.5 Recognition of Graphics

Generally speaking, there are two major approaches for recognition of graphics in line diagrams. The first is a statistical and numerical or decision-theoretic approach [Fu 82]. In this approach, a pattern $X$ is represented by a feature, or attribute, vector:

$$X = [x_1, x_2, ..., x_n]$$
Each pattern is compared or matched with stored models in order to determine that individual model, or class of models, which is most similar to the unknown pattern. The comparison process is based on a numerical distance function or on concepts from probability theory. However when the patterns under study are very complex or when the number of pattern classes is very large, the number of features 'n' required for recognition can also become very large, and the statistical approach becomes ineffective or computationally infeasible for solving such problems.

The second approach is a systematic or structural one that attempts to interpret a line diagram in a manner analogous to that of a human interpreter. A person looking at an image of a line diagram typically interprets it by identifying its components in a bottom-up manner. For example a chemist who understands conventions for chemical structural diagrams, will see basic shapes like hexagons and infer the existence of a standard six carbon structural unit. Then looking at the lines connected to the hexagon and reading the atom names written at the end of bonds, the chemist can identify the "substituents on the molecular skeleton" and so determine the formula and structure of the compound represented. Once the overall structure has been inferred, the chemist reviews and interprets additional stereochemical information present in the form of "wedging" or "hashing" on particular lines (bonds) along with details in the image, that would have been largely ignored earlier, such as "bullet marks" or "underscores" at line intersections.

Typically, an image is interpreted first in terms of lines. A second level of interpretation groups some of these lines into larger composites (like the chemist recognizing a group of six lines that form the outline of a hexagon and associating this with a standard chemical component). At a third level, components and residual lines are combined to obtain larger structures. When going between each level, somewhat different kinds of knowledge are used.
Such an approach to processing can be modelled using "blackboard systems" as originally developed by researchers using Artificial Intelligence to solve signal processing problems. A blackboard system (architecture) is a generic term that covers both applications and frameworks [Engelmore et al. 88]. A blackboard application is a system that solves a particular kind of problem, such as the interpretation of a visual scene. A blackboard framework is either a specification of the components of a blackboard model (that is a particular kind of problem-solving model) or an implementation of the specification.

![Diagram](image)

**Fig. 2.1: Traditional Expert System Model.**

Fig. 2.1 illustrates the structure of a classical expert system. This structure has two weak points. First, control of the application of the knowledge is implicit in the structure of the knowledge base, for example, in the ordering of the rules for a rule-based system. The second weak point is dependency of knowledge representation on the nature of the inference engine. For example, a rule interpreter can only work with knowledge expressed as rules. To eliminate these two weaknesses, the expert model could be redesigned as a blackboard model as shown in Fig. 2.2. In this model the knowledge is segmented into modules, with each module containing related entities. Each module may provide a separate inference engine. The weaknesses of the expert systems is now eliminated and there is no need to use the same representation for the separate parts of the knowledge base, and no need for the separate inference engines to be identical in operation. The structured working memory is called the "blackboard" and each pair of the inference engine and related knowledge base is called a "knowledge source".
Therefore the blackboard model is composed of two basic parts. There are the knowledge sources that partition the knowledge needed to solve the problem; these are kept separate and independent. Then there is the blackboard data structure which is organized into one or more application-dependent hierarchies (Fig 2.3). Knowledge sources produce changes to the blackboard which lead incrementally to a solution to the problem. The purpose of the blackboard is to hold computational and solution-state data needed by, and produced by, the knowledge sources. Communication and interaction among the knowledge sources take place solely through the blackboard. There is a set of control modules that monitor the changes on the blackboard and decide what action to take next. The solution is built one step at a time. Any type of reasoning step (data driven, goal driven, model driven, and so on) can be applied at each stage of solution formation. As a result, the sequence of knowledge source invocation is dynamic and opportunistic rather than fixed and preprogrammed.
Techniques used for graphic recognition (as mentioned before) are strongly application dependent. Simple template matching techniques, applied directly to the bit-mapped image, may be adequate for the recognition of isolated symbols of fixed size and orientation. In certain applications, it may be adequate to approximate closed contours with polygons. In others, for example face recognition, more sophisticated algorithms that hypothesize possible matchings, compute scene/model transformations and verify the hypothesis are used. In more complex images such as maps and engineering drawings, context-dependent, knowledge-based graphic recognition and interpretation techniques have been used.

An object recognition system that creates a library of parts by hierarchical decomposition is described in Ettinger [88]. The library organization and indexing is designed to avoid linear searching of all the model objects. Object representations are based on the Curvature Primal Sketch [Asada and Brady 86]. Features used are corners, end, crank, smooth-join, inflection and bump, which are derived from discontinuities in contour orientation and curvature. The recognition engine is structured as an interpretation tree [Grimson and Lozano-Preze 84].

For the recognition of drawings such as flowcharts, block diagrams, electrical schematics, mechanical part drawings, etc, a structural analysis of the lines connecting components is required to generate meaningful descriptions. Kasturi et al [92] describe structural analysis methods used to find closed loops in the drawings. The approach is based on an earlier algorithm (El-Masri [88], Kasturi et al. [90]). Using these methods all closed loops (such as triangles, rectangles, etc) are found and their properties are listed in a table for finding their relationships according the image type.

Ejiri et al. [90] apply structural analysis methods for the recognition of engineering drawings and maps. Fahn et al. [88] describe a topology-based component extraction
system for recognizing symbols in electronic circuit diagrams. The objective is to extract circuit symbols, characters and connecting lines.

Another approach to recognition uses a relaxation method. Bunke and Allerman [81] have used this method to analyse electrical schematics. A vector with probabilities of possible interpretations is first assigned to each vertex (i.e. line segments and or conjunction of two or more segments) in the schematic. The probabilities of the interpretations at various vertices are successively changed by the relaxation process in order to achieve unique and consistent labelling. Rule-based or knowledge-based systems have also been used as recognition engines (Nagy et al. [85], Niyogi and Srihari [86], Bley [84], Bunke [82a], Huang and Tou [86], Nagasamy [89]).

2.3 Existing Diagram Interpretation Systems

Line diagrams can be classified as either "logic-dependent" or "figure-dependent" [Iwatta et al. 88]. Logic-dependent diagrams are those where the components are logically related and connected to each other. Examples include electric or logic circuit diagrams and the stereochemical representation of chemical structures. In contrast, printed circuit diagrams and maps (in general) are classified as "figure-dependent" drawings. This means that there is no logical dependency between components of such diagrams (contour lines and map features can appear almost random). Three-view mechanical drawings are grouped as the third type, as they contain both logic and figure information.

2.3.1 Circuit Diagrams and Electronic Logic Diagrams

Early work in the area of processing of electronic logic diagrams is reported by Kakumoto, et al [78]. Their system uses a "divide and synthesis" method for the
automatic recognition of large hand-drawn logic diagrams on computers with limited memory. In their approach, an image is divided into small portions and each portion is processed. Text is extracted and the box type symbols are recognized by their location and endpoints. At the end, all image portions' results are connected together to make a complete model. The system required a simplified input. Diagrams have to be drawn aligned to a grid and connecting lines must be either horizontal or vertical. There were also restrictions in regards to the size of logic symbols and characters.

Bunke [82b] has described several methods for the automatic extraction of descriptions of circuit diagrams, or of flowcharts from image data given as input. His first method is called "probabilistic relaxation". A vector of possible interpretation labels (for example "resistor terminal", "resistor corner", "connection corner" and so on) is assigned to each vertex in a diagram. The initial probabilities for different interpretations of each vertex are then successively updated. The update process checks for consistency of interpretations at adjacent vertices. For example, the label "resistor corner" for a vertex is compatible with the label "resistor terminal" at an adjacent vertex, while it is incompatible with the label "diode terminal".

The second method, described by Bunke, stores prototypes for the allowed symbols of a diagram in a decision tree. The method involves tracking lines in the diagram and matching the configuration of line segments present at a vertex with models represented in the decision tree until the whole diagram is analyzed. For example, suppose the schematic diagram under consideration is composed of diodes and resistors that are shown as a box (Fig. 2.4). A decision tree for interpretation of this part of the diagram is shown in Fig. 2.5 (thin lines identify the lines already matched while thick lines are used for those lines still to be matched).
Finally, Bunke has described the use of "attributed programmed graph grammars" (these are covered in the next chapter). This method is equivalent to transforming a graph representation of the input drawing to a graph representation of the corresponding description.

Bunke [82c] also reported a system for the automatic interpretation of lines and text in circuit diagrams. The most salient characteristic of his work is in his representation of prior knowledge about the form of symbols with a decision tree. This decision tree has
to be constructed by the user (domain expert) and is given to the system as input data. In this way the system can be easily adapted to different set of symbols. However, the construction of the decision tree is a very complicated task, particularly when there are a large number of symbols.

The primary algorithm, employed by Fukuda [82] for the understanding of logic circuit diagrams, depends on the local detection of symbols using a thinning algorithm. The algorithm is composed of four stages: Line tracking; Significant area processing; Logic symbol processing; and Starting point detection. There are numerous restrictions on the forms of input drawings. For example, the signal line must be horizontal or vertical, logic symbols must be greater than 3 mm and smaller than 2 cm squares and so forth.

Shimizu et al [82] have proposed a system for processing hand-drawn logic circuit diagrams. Their system has three stages: 1. Feature extraction, 2. Shape discrimination, and 3. Feature interpretation. The algorithm is applied to the diagram which is written using 90 symbols and 36 alphanumerics.

Lee et al (Lee [89], Lee et al. [90], Lee et al. [91]) reported a recognition system of hand-drawn symbols in schematic diagrams. Their system may be subdivided into three categories: Firstly, the presentation of a model-based scheme for recognizing similar hand-drawn symbols in schematic diagrams with AG (Attributed Graph) matching in the absence of any information concerning their pose (translation, rotation, and scale). Secondly, the characterization of the distance measuring process between the Observed AG and Model AG from a probabilistic point of view, and approximation of the conditional probability is gained by modelling the probability of some features of a component by a Gaussian distribution. Thirdly, a Model AG learning algorithm is
presented in which the structural components and their attributes are extracted to construct the model.

Matsello [91] has proposed an approach for the recognition of line drawings (especially logic circuit diagrams) using attributed graph grammars. He has described a formal model that permits the definition of classes of images represented by straight line segments. He has also proposed a fast parsing algorithm which is associated with his model.

2.3.2 Chemical Structure Diagrams

Molecular structures such as vitamins, alkaloids, antibiotics, pheromones, organometallic complexes etc, may contain a 2-D stereochemical representations (dot and wedge convention) and delocalized bonds as in donor-acceptor complexes. Molecular structures consist of two components: (a) a graph or skeleton of the structure and (b) common and special alphanumeric characters (symbols, parenthesis, charges). A molecular structure recognition system converts such an image into a basic connection table (that identifies atoms and bonds) along with supplementary data tags that encode stereochemical information etc. This recognition of chemical structures is necessary for the selective retrieval of information from papers.

One of the first significant developments in the area of chemical structure formula recognition was made by Okazaki and Tsuji [88]. Their system, "KARD" (Knowledge-based Adaptive Recognition system for Drawings), was designed as a test-bed system for a drawing recognition method with constructional knowledge of the drawings concerned. As there would be a variety of drawings, the system should be able to deal with these differences ("target variety"). Each drawing would contain different type of distortions known as "distortion verity". In order to resolve "target
variety" and "distortion variety", "KARD" adopts an objective pattern unit description of rules and a data-flow-like activation control mechanism for the rules. However, "KARD" needs further study to cover control mechanism and implementing effective top-down searches for the target pattern.

Contreras et al [90] have presented a system which supports the capture, perception, and recognition of typed and hand-printed molecular structures. Their process involves four steps: (a) scanning of molecular structures, (b) graph recognition, (c) character recognition and (d) display. The first step (scanning) is done with a HP Scanjet which is linked to a PC-AT. The digitization resolution is between 75x75 and 300x300 dpi, according to the size of the molecular structure. The digitized image is then transferred to a micro Vax II under an Ultrix operating system.

The second step (the perception of the graph) consists of several substeps. Firstly, the program does a left-to-right horizontal sweep. It starts from the top-left part of the image until it finds the first set pixel of the image. Then a counterclockwise contour search algorithm is applied until it arrives back at the first pixel. When the number of contour pixels is more than a threshold it is interpreted as the graph contour (this threshold is defined according to the resolution of the scanned image). Otherwise it is considered as a chemical symbol. In the first case, deflection of the linear trajectory of any external or internal border indicates the existence of a vertex. Two or more vertices within a defined small space indicate the point of the graph where an atom should be located.

Perception of multiple bonds and internal rings as well as perception of other molecular substructures or subgraphs is carried out through a circular inspection method. For this, a circle is centred on the middle of the space assigned to each atom. The radius of this circle is chosen to be one third of the value corresponding to a single bond length.
Unknown border pixels found in this way are used as the initial point for both a new counterclockwise contour tracing and a perception of new vertices and probable new atoms. Once the topological characterization of the graph is completed, each attribute is represented as a typical Prolog data structure. From this representation a connectivity table is constructed.

2.3.3 Engineering Drawings

Some early work on the computer processing of line drawings was carried out by Freeman[74], Jarvis[77], and others. Clement[81] and Black et al. [81] describe a technique for automatically converting engineering drawings into a set of line segments and interpreted characters. The first step of their technique involves scanning the drawing at a coarse resolution to classify it into regions containing no lines and those containing lines that need to be further analyzed. Using a track follower, simple lines are extracted and fitted with straight line segments. At the end there may be some areas of image which are not interpretable by this technique. These are flagged to be interpreted by hand. The main problem encountered during the experiment occurred at the line junctions and with small clusters or blobs.

Iwata, et al [88] have developed an automatic interpretation system that interprets "three-view drawings" (i.e. diagrams of machine parts viewed from three orthogonal directions). In implementing a practical system they established new rules on mechanical drawing standards. These standards are: only six types of line segments (object line, hidden line, dimension line, extension line, leader line, and centre line) can be used. Lines may be solid, broken, or dot-dash lines of any thickness and so forth.

After carrying out conventional preprocessings on the image of drawings and converting it to a set of vectors, they divide the vectors into two different parts called
'isolated' and 'unisolated' figures. A relatively small rectangular area enclosed by a set of interconnected vectors is called an 'isolated' figure. Classification of the isolated areas will result in the recognition of characters or "hidden" and "centre" lines, and the line segments in 'unisolated' figures are assumed to be dimension lines or leaders. Comparison of the recognised character string with data in a knowledge base is used to determine the size and the shape of the designated figures. Their system, by referring to dimensions on other views of the drawing, is able to determine deleted or undefined dimension in each view, and to find deleted or missed object/hidden lines and so solve inconsistency problems. The final stages of the interpretation process depends on user input.

Nagasamy[89] has presented techniques for the preprocessing (noise removal, void filling, image segmentation, line thinning, and contour extraction) and vectorization of scan digitized images of engineering drawings. Following vectorization, the resulting data can be interpreted using rules that define the forms allowed in engineering drawings. The interpretation process results in structural descriptions that can be transferred directly into a CAD/CAM system. In the vectorization step, in addition to straight line segments, his system also handles circles, arcs, and conic sections. The system is able to deal with very large scale drawings (C and D sizes) and blueprints. He did not cover some more difficult subtasks such as the recognition of textured lines (dashed lines, centre lines, ...), the identification of dimensions associated with contours in machine drawings, recognition of special symbols commonly used in engineering drawings and general issues dealing with missing or inconsistent information present in the input.
2.3.4 Maps and "Plats"

Madej [91] has presented an intelligent system for automatic conversion of "cadastral" maps to CAD. A cadastral map is drawn in accord with the following rules:

- The standard defines 14 different symbols. They are composed of one, two, or three small, concentric circles with diameters of range 0.7-1.3 mm.
- A given symbol may represent a parcel corner-point. Parcel borders are expressed as lines that connect symbols.
- The coordinates of the centre of a symbol should be approximated with precision ±1 pixel. His method allows precise recognition of symbols and proper interpretation of parcel borders. He has used a priori knowledge for approximation and interpretation of symbols and symbol-lines contexts.

Antoine and Lorraine [91] have also applied a priori knowledge method to understand features of French "plats" (i.e. city maps). French city maps are normalized, and detailed rules are available to extract different components such as buildings, gardens, yards, plots, blocks of plots and roads.
3. Implementing an Interpreter

3.1 Introduction

This chapter covers the design and implementation of a system that uses a "blackboard" model for the interpretation of schematic line diagrams of simple electrical circuits. This special purpose interpretation system was constructed to exercise all the low-level image analysis components that had to be developed, and to provide a base from which to explore more general interpretation schemes.

The system takes as input an "Xbm" format file containing the scanned image of an electrical circuit (for instance, a circuit representing a simple radio). The circuits used as examples mostly came from an introductory text associated with an electronics circuit kit. The raw bit map data is subjected to a number of transforms that "clean up" the image.

Typically, a scanned image is degraded with both positive and negative noise. The negative noise appears as small voids in areas that should be solid — resulting in raggedness on the edges of solid parts of an image, or as breaks in lines. The images of lines in the bit map are generally more than one pixel in width; widths tend to vary a little along the length of a line. Circles, arcs, and many oblique lines in an original
drawing appear in the bit image as adjacent sequences of bits arranged with varying orientations. The first few processing steps perform operations like the filling of voids. Solid areas must be separated from the rest of the image and then each solid area must be converted to a set of lines that represent its boundary contours. The "lines" in the bit image must be thinned to an idealized one pixel width.

Basic operations like void filling and line thinning are fairly standard. They are based on principles of morphology, as reviewed in section 3.2. The implementations carried out for this project are presented in section 3.3. to section 3.7. The methods used to separate solid regions from the rest of an image and then find their contours are novel.

After a bit image has been "cleaned up", the remaining processing steps create increasingly more abstract, high-level representations of the data represented in the bits. "Lines" are created — these are not yet straight lines defined by specific endpoints, or curves — instead they are a program's representation of contiguous runs of set pixels remaining in the cleaned up bit image. These "lines" are created by scanning an image, raster by raster, finding each set pixel and adding it to the appropriate "line" structure. The raster scan algorithm developed for this task is presented in section 3.6 (Initially, it was thought that the algorithm was novel but it was later established that there have been previous reports presenting similar algorithms).

These pixel sequences or "lines" must then be "vectorized" so as to obtain actual straight lines, arcs and so forth. Section 3.7 presents the mechanism used in the vectorization of digital line segments. The algorithms described in sections 3.3 to section 3.7 are all implemented as relatively low-level general purpose image interpretation functions. Functions such as these underlie the working of any line diagram interpreter.
Sections 3.8 and 3.9 present more limited special purpose functions designed specifically for the initial trial application of a diagram interpreter. Reviewing the images that were to be processed suggests that they could be defined in terms of primitive components such as rectangles (such as images of resistors), circles and semicircular arcs (for example images of electronic gates and signal inverters so forth), triangles (like parts of amplifiers), groups of parallel lines (such as capacitors, batteries, transformers) and connecting lines (wires). The approach taken to recognize these "primitives" from the finalized line segments was a little ad hoc as it relied on specialized data structures and hand crafted code. Details are given in section 3.8.

Several different kinds of knowledge are needed to interpret a diagram that represents an electrical circuit. One has to be able to recognize groups of parallel lines as representing batteries, or maybe capacitors, or possibly transformers. Such different interpretations are triggered by subtly different visual cues (like the uniformity in height of lines representing a capacitor as opposed to a 2::1 variation typical in the image used to represent a battery). Other information used to establish the correct interpretation of a component might come from an examination of the connections between that component and the other elements in the circuit. General rules about valid circuits can be used to check a proposed interpretation. For example, if the interpretation system finds that it is proposing an interpretation where the outputs of two "and" gates are linked by a connecting wire, then it should guess that something went wrong at an earlier stage and should try finding an alternative interpretation for at least one of the "and" gates or for the interconnecting wire.

Typically, an interpretation system will try to locate some easily identified components that will serve as starting points and will then try to find a consistent interpretation for the whole circuit as it searches for interconnections and other components. At one moment, the interpreter will be applying low level rules for recognizing a group of
parallel lines as a battery and at the next moment, the interpreter may be applying a higher level rule limiting interconnections of batteries and logic gates. Such problem solving behaviour seemed reminiscent of "blackboard architecture" expert systems that apply many types of knowledge to data represented at varying levels of abstraction on a "blackboard".

Section 3.9 presents a limited implementation of a "blackboard" system specifically designed for these interpretation problems. The weak points of the designed interpreter and methods to prevent them and/or to improve the system, are discussed in section 3.10.

### 3.2 Morphological Filters

Mathematical morphology was formalised during the mid 1960's in connection with an investigation of the relationship between the geometry of porous media and their permeabilities. These concepts later found an application in image processing for the modification of the spatial form or structure of objects in an image. The various transformations that form part of this subject are based on the following concepts [Serra 82]:

- Invariance under translation.
- Local knowledge.
- Semi continuity with respect of Hit or Miss topology.

Subsection 3.2.1 defines some terms, then in subsection 3.2.2 several different operators for modifying and combining images are described. Subsections 3.2.3 and 3.2.4 discuss how various operators can be combined to achieve results such as the filling of voids and image segmentation. Subsection 3.2.5 consists mainly of figures illustrating the effects of various operators.
3.2.1 Digital Image Definition.

Usually the domain "D" of a digital image \( f \) will be rectangular in shape and will contain a finite number of elements [Dougherty and Giardina 87]. Thus, a digital image will be represented in a manner similar to a matrix or two-dimensional array. An image will have a grey value (in our case binary level value (0, 1)) for each position within the array. In addition to the (0/1) known values, it is useful to be able to represent an undefined value ('*' in the following diagrams).

"Image operators" (that will be used to match with or transform bit images) can be represented by small matrices such as that shown in Fig. 3.1. The two subscripted integers outside the lower-right part of the matrix, in Fig. 3.1, specify the absolute location, in the XY lattice, of the upper leftmost entry in the matrix.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>*</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ f = \begin{pmatrix} 0 & * & 1 \\ 1 & 0 & * \\ 1 & 1 & 1 \end{pmatrix}_{0,2} \]

Fig. 3.1: Image with nine pixels.

3.2.2 Operations

The possible operations on images are divided into three major groups;

"arithmetic-type or binary operations",
"transformational or unary operations",
and
"morphological operations".
The first group of operations, covered in 3.2.2.1, includes image addition, multiplication, etc. Operations such as rotation, translation, etc. form a second group as described in section 3.2.2.2. The "morphological" operations, described in section 3.2.2.3, include processes such as dilation, erosion, closure etc.

3.2.2.1 Arithmetic-type Binary Operations for Images.

In following definitions \( f \) and \( g \) are the two images that are to be combined by the defined operation. All operations are carried out on the intersection of their operands domain, which is the subimage of both operand's where they intersect each other. If the operands of the operation do not intersect, there is no result.

Addition of Images:

\[
[\text{ADD}(f,g)(i,j)] = f(i,j) + g(i,j) \quad \text{if } f \text{ and } g \text{ are both defined at } (i,j).
\]

If either \( f \) or \( g \) is undefined at \( (i,j) \), \([\text{ADD}(f,g)](i,j) = *\).

Addition Algorithm for Two Images.

```c
Image* ADD( Image* f, Image* g) // Add two images f and g .
{
    if (NOT Find_Intersect(f, g)) return NULL;
    s = create new Image;
    for (each Row and Col of s) do
        if (f(Row,Col) and g(Row,Col) are Defined)
            then s(Row,Col) = f(Row,Col) + g(Row,Col);
        else s(Row,Col) = '*';
    return s;
}
```

Fig. 3.2: Addition Algorithm for Two Images.
Multiplication of two Images:

\[
[MULT(f, g)](i, j) = f(i, j) \cdot g(i, j) \text{ if } f \text{ and } g \text{ are both defined at } (i, j).
\]

If either \( f \) or \( g \) is undefined at \((i, j)\) \( [MULT(f, g)](i, j) = \ast \).

```c
Image* MULT( Image* f, Image* g) // Multiply two images f and g
{
    if ( NOT Find_Intersect(f, g)) return NULL;
    s = create new Image;
    for (each Row and Col of s) do
        if (f(Row,Col) and s(Row,Col) are Defined)
            then s(Row,Col) = f(Row,Col) * g(Row,Col);
        else s(Row,Col) = '*';
    return s;
}
```

Fig. 3.3: Multiplication Algorithm for Two Images.

Maximum Operator:

\[
[MAK(f, g)](i, j) = \max[f(i, j), g(i, j)] \text{ unless either } f(i, j) \text{ or } g(i, j) \text{ is undefined.}
\]

```c
Image* MAX( Image* f, Image* g) // Max of two images f and g
{
    if (NOT Find_Intersect(f, g)) return NULL;
    s = create new Image;
    for (each Row and Col of s) do
        if (f(Row,Col) and s(Row,Col) are Defined)
            then s(Row,Col) = max( f(Row,Col), g(Row,Col));
        else s(Row,Col) = '*';
    return s;
}
```

Fig. 3.4: Finding Maximum of Two Images.

Minimum Operator:

\[
[MJN(f, g)](i, j) = \min[f(i, j), g(i, j)] \text{ unless either } f(i, j) \text{ or } g(i, j) \text{ is undefined.}
\]

```c
Image* MIN( Image* f, Image* g) // Min of two images f and g
{
    if ( NOT Find_Intersect(f, g)) return NULL;
    s = create new Image;
    for ( each Row and Col of s ) do
        if ( f(Row,Col) and s(Row,Col) are Defined )
            then s(Row,Col) = min( f(Row,Col), g(Row,Col) );
        else s(Row,Col) = '*';
    return s;
}
```

Fig. 3.5: Finding Minimum of Two Images.
Cutting one image from another:

This Operator will cut image \( g \) off image \( f \). The operation would be done on intersection part of the two images.

\[
\text{Image}^* \text{ CUT} (\text{ Image}^* \ g, \text{ Image}^* \ f) \quad // \text{ cut image} \ g \text{ out of} \ f.
\{
\quad \text{if} (\text{ NOT Find_Intersect}(g, f)) \quad \text{return NULL;}
\quad s = \text{ create new Image;}
\quad \text{for ( each Row and Col of } s \text{ ) do}
\quad \quad \text{if ( } f(\text{Row,Col}) \text{ is Defined )}
\quad \quad \quad \text{then if ( } s(\text{Row,Col}) \text{ is Defined )}
\quad \quad \quad \quad \text{then } s(\text{Row,Col}) = f(\text{Row,Col}) \text{ AND (NOT } g(\text{Row,Col}))
\quad \quad \quad \quad \text{else } s(\text{Row,Col}) = '*'\;
\quad \quad \quad \text{else } s(\text{Row,Col}) = '*'\;
\quad \quad \text{return } s;
\}
\]

Fig. 3.6: Cutting one Image from another.

Extension Operator:

\[
[\text{EXTEND}(f,g)](i,j) = f(i,j) \text{ if } f \text{ is defined at } (i,j) \text{ otherwise } = g(i,j).
\]

Combination of operations:

Any combination of defined operations is permitted. The most useful combined operation is extension of the maximum operation, which is called "EXTMAX".

\[
\text{EXTMAX}(f,g) = \text{EXTEND}(\text{EXTEND}(\text{MAX}(f,g),f),g).
\]

3.2.2.2 Transformation Operations

In this section some useful transformation operations such as translation and rotation are defined and related algorithms are presented. For these implementations, it is assumed that the leftmost upper corner of the screen is the origin of the coordinate; and the vertical axis is downward (as is standard on most window-based graphics displays). Moreover, the operations are carried out in such a way that the result remains on the screen. Consequently, the algorithms sometimes use details of the height or width of the image.
Translation Operation:

\[ \text{TRAN}(f; i, j) = (a_{p,q})_{r+i,t+j} \quad \text{or:} \quad [\text{TRAN}(f; i, j)](u,v) = f(u-i, v-j) \]

Pay particular attention to the minus signs in the argument for \( f \).

90 degree Rotation:

\[ [\text{NINETY}(f)](i, j) = f(j, -i). \]

180 degree Rotation:

\[ [\text{NINETY2}(f)](i, j) = f(-i, -j). \]

Reflection about 135 degree Line:

\[ [\text{FLIP}(f)](i, j) = f(-j, -i) \]

Fig. 3.7: 90 Degree Rotation Algorithm.

Fig. 3.8: 180 Degree Rotation Algorithm.

Fig. 3.9: Reflection Algorithm.
3.2.2.3 Morphological Operations

When working with constant images, two operations are of particular importance. They are the minimum operation MIN and the extended maximum operation EXTMAX, which are defined in section 3.2.2.1. For constant images with constant grey value 1, these two operations play the role of intersection and union respectively. For such images, the MIN operation outputs a constant image (with grey value 1) whose domain is the intersection of the two input domains and EXTMAX outputs a constant image (with grey value 1) whose domain is the union of the two input domains. Because of these intersection and union properties, the following special notations are introduced:

\[ S \cap T = \text{MIN}(S, T) \quad \text{and} \quad S \cup T = \text{EXTMAX}(S, T). \]

\[
S = \begin{pmatrix}
1 & 1 & 0 \\
1 & 0 & 0
\end{pmatrix} \quad T = \begin{pmatrix}
0 & 1 & 1 & 1 & 0 \\
0 & 1 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 & 0
\end{pmatrix}
\]

\[
S \cup T = \begin{pmatrix}
1 & 1 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 & 0
\end{pmatrix} \quad S \cap T = \begin{pmatrix}
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0
\end{pmatrix}
\]

Fig. 3.10: Examples of MIN and EXTMAX operations

In all following definitions, \( E \) will represent a "structural element" for the image. A structural element is a constant image. It has to be selected very carefully for each operation to produce proper results. For example, \( E \) could be a 2X2 pixels image or an 1X4 pixels image etc. Examples are shown in Fig. 3.11.

\[
E = \begin{pmatrix}
1 & 0 \\
1 & 1
\end{pmatrix} \quad E = (1 \ 1 \ 1 \ 0)
\]

Fig. 3.11: Examples of structural elements
Dilation Operation:

Dilation of $S$ by $E$ is the union of all translations of the structural element $E$ on the domain of $S$.

Image* DILATE( Image* f, Image* e) //Dilate f by e.
{
    s = create new Image;
    for (each Row and Col of f) do
        if (f(Row,Col) is defined)
            for (each row and col of e) do
                if (e(row,Col) is in the range of f)
                    then s(Row,Col) = s(Row,Col) OR e(row,Col);
                else;
            else;
        return s;
}

Fig. 3.12: Dilation Algorithm.

$\text{DILATE}(S,E) = \cup \text{TRANS}(E; i,j)$ for all $(i,j)$ in domain of $S$.

$$S = \begin{pmatrix}
0 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 \\
\end{pmatrix} \quad \quad \quad E = \begin{pmatrix}
1 & 0 \\
1 & 1 \\
\end{pmatrix}
$$

$\text{DILATE}(S,E) = S \oplus E = \begin{pmatrix}
1 & 0 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 \\
\end{pmatrix}$

The positions $i,j$ become 1 for $E$.

$$S \oplus E = \begin{pmatrix}
0 & 1 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 & 0 \\
\end{pmatrix} = S$$

Fig. 3.13: Example of Dilation operation.
Erosion Operation:

\[
[\text{ERODE}(S, E)](i, j) = 1 \text{ if } \text{TRAN}(E; i, j) \uplus S = S, \text{ otherwise it is 0.}
\]

Eroding is done by matching each translation of \( E \), the structural element, with a subimage of \( S \). If they are matched, then the pixel \((i, j)\) is activated in the eroded image.

```c
Image* ERODE( Image* f, Image* e) {  //Erode f by e.
    s = create new Image;
    for (each Row and Col of f) do
        if (f(Row,Col) is defined) then
            for (each row and col of e)
                if (e(row,Col) is in the range of f and e(row,Col) == f(Row,Col))
                    then s(Row,Col) = 1;
                else s(Row,Col) = 0;
        else;
    return s;
}
```

Fig. 3.14: Erosion Algorithm.

\[
S = \begin{pmatrix}
0 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 & 0 \\
\end{pmatrix}
\]

\[
E = \begin{pmatrix}
1 & 0 \\
1 & 1 \\
\end{pmatrix}
\]

\[
\text{Erode}(S, E) = \Theta(-E) = \begin{pmatrix}
0 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 \\
\end{pmatrix}
\]

These positions become 1 for
\[
S \Theta \neg E = \Theta
\]

Fig. 3.15: Example of Eroding Operation.
Open Operation:

\[
\text{OPEN}(S,E) = V \{\text{TRAN}(E; i,j) \mid \text{TRAN}(E; i,j) \lor S = S\}
\]

From the definition it should be clear that a pixel is activated in the opened image if, and only if, it is a part of a fitted copy of \(E\), the structural element.

\[
S = \begin{pmatrix}
0 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 & 0
\end{pmatrix}
\]

\[
E = \begin{pmatrix}
1 & 0 \\
1 & 1
\end{pmatrix}
\]

\[
\text{Open}(S,E) = \text{SOE} = \begin{pmatrix}
0 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 0
\end{pmatrix}
\]

Fig. 3.16: Example of Opening Operation

Image* OPEN( Image* f, Image* e) //Open (sizing) f by e.
{
    set ISMIN;
    s = create new Image;
    for (each Row and Col of f) do
    {
        if (e lays in f AND f(Row,Col) is ON)
            then for (each row and col of e) do
                if (EXTMAX(TRAN(e), f) \neq f)
                    then reset ISMIN and break;
            if (ISMIN is set)
                then for (each row and col of e) do
                    s(Row,Col) = max(e(row,Col), f(Row,Col));
            else set ISMIN;
    }
    return s;
}

Fig. 3.17: Opening Algorithm.
Close Operation:

$$[\text{CLOSE}(S,E)](p,q) = 1 \text{ if for all } (i,j) \text{ such that }$$

$$[\text{TRAN}(E; i,j)](p,q) = 1, \text{ and } \text{TRAN}(E; i,j) \land S \neq \emptyset, \text{ otherwise it is } 0.$$  

According to the definition of closing, a pixel $\langle p,q \rangle$ appears in the closed result image if, and only if, every translation of $E$ containing $\langle p,q \rangle$ intersects the original image $S$.

$$S = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

$$E = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$

$$\text{Close}(S,E) = S \otimes E = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Fig. 3.18: Example of Closing Operation

Image* CLOSE(Image* f, Image* e) //Close f by e.

{  
s = create new Image;
  for (each Row and Col of f) do
  {
    for (each row and col of e) do
      if ( e(row,Col) is ON)
        then if ( e(row,col) == 1
                  AND NOT Intersects( TRAN(e,Col,Row), f))
          then break;     // internal loop.
        if (NOT breaked )
          then s(Row,Col) = 1;
  }
return s;
}

Fig. 3.19: Closing Algorithm.
Properties of Morphological Operations

Morphological operations satisfy the following relationships (in these relations "Comp(x)" is the complement set of x with respect to the whole image):

\[
\begin{align*}
\text{a)} & \quad \text{Dilate}(S,T) = \text{Dilate}(T,S) \quad \text{[Commutativity]} \\
\text{b)} & \quad \text{Dilate}(S,E) = \text{Comp}(\text{Erode}(\text{Comp}(S),-E)) \quad \text{[Duality]} \\
\text{c)} & \quad \text{Erode}(S,-E) = \text{Comp}(\text{Dilate}(\text{Comp}(S),E)) \quad \text{[Duality]} \\
\text{d)} & \quad \text{Comp}(\text{Open}(S,E)) = \text{Close}(\text{Comp}(S),E) \\
\text{e)} & \quad \text{Comp}(\text{Close}(S,E)) = \text{Open}(\text{Comp}(S),E)
\end{align*}
\]

3.2.3 Some Experimental Results

The results of the operations described in the previous section are illustrated here. Fig. 3.20 is the original scanned image of a schematic circuit diagram. There is little positive noise in this image, but negative noise has lead to some voids and break points.

![Original Schematic Diagram](image-url)
The effects of a "dilate" operation are shown in Fig. 3.21. Here, a 2X2 structural element was used in the operation. The result of this operation is an image that is thicker than the original, but the voids are not filled nor are breaks connected.

Fig. 3.21: Schematic Diagram Dilated by a 2X2 Structural Element.

Fig. 3.22 illustrates the effects of a close operation on the image. This has filled in some voids and reconnected some breaks of the image. However, large voids are not filled and some breaks are still left in the closed image.

Fig. 3.22: Line Diagram Closed, using a 2X2 Structural Element.
Eroding an image (such as the one in Fig. 3.20) with a 2X2 structural element results in the image shown in Fig. 3.23. This is an overall thinned version of the image with more breaks.

![Diagram](image1)

**Fig. 3.23:** Schematic Line Diagram Eroded, using a 2X2 Structural Element.

Fig. 3.24 shows the effects of the opening operation. In this case the image is not thinned, but more breaks have been created.

![Diagram](image2)

**Fig. 3.24:** Opening operation on a Schematic Diagram using a 2X2 Structural Element.
3.3 Filling In Small Voids in the Image

Using a morphological CLOSE operation, some breaks or voids are filled in for the line structures found within an image. As mentioned earlier, these voids are caused by the scanning process. The structural element $E$ used in this CLOSE operation may be selected as a 2X2 or 3X3 matrix. The choice of a 2x2 or 3x3 close operator depends on the amount of space between parts of the image.

For example if there are at least 3 empty pixels between any two parts of the image (for example between parallel lines) then the use of a 3X3 structural element would not join these parts. If parts are closer, the smaller 2X2 operator should be used (of course, if parts of an image are very close there is always a possibility that they may be joined by a close operator). Structural elements to be selected would have the rightmost upper element as 0 and the rest as 1. The origin of the structural element should be in the rightmost lower element. The result of a CLOSE operation on a sample part of a diagram is illustrated in Fig. 3.25.

![Fig. 3.25: a) Scanned image with some breaks and voids, b) Image after CLOSE operation.](image)

3.4 Contour Tracing

Many algorithms exist for tracing the contours of thick regions in a binary image (Chen and Siy [87], Legault and Suen [89] and Liow [91]). A new algorithm has been implemented based on a sequence of operations as defined by the following formula:
$S_2 = \text{CUT}(S_1, \text{TRAN}(\text{ERODE}(S_1, E_w), -w/2, -w/2) )$

Erosion of $S_1$ by $E_w$, which is a $w \times w$ structural element, will cause an intermediate $S'_1$, that is at most $w$ pixels thinner than $S_1$. Consequently, all parts of the image that were at most $w$ pixels wide are removed. Cutting the $w/2$ translated version of $S'_1$ from $S_1$ will result in traced contours of the thick regions of the image. As the image $S_2$ has to be thinned later, the thickness of the created contours in $S_2$ is not of immediate concern. The effect of these operations on the original image (Fig. 3.20) by $w=7$, is shown in Fig. 3.26.

![Fig. 3.26: Contour Tracing via Cutting Eroded Image from the image, note how the arrowhead (Fig 3.20) has now been reduced to its contour.](image)

### 3.5 Line Thinning

The algorithm used here for line thinning is sequential in nature but it can be implemented as a pipeline procedure called 'SPTA' (Naccache and Shinghal [84], Nagasamy [89], Lam et. al. [92]). Each dark point of the image is tested against the eight neighbours around it as shown in Fig. 3.27. These 8 points are known as the 8-neighbours of point $p$ and are denoted by $n_0$, $n_1$, $n_2$, $n_3$, $n_4$, $n_5$, $n_6$, and $n_7$ respectively, and collectively are denoted by $N(p)$. The number of dark pixels in $N(p)$ is denoted by $b(p)$. The pixels $n_1$, $n_2$, $n_3$, and $n_4$ are called the 4-neighbours of point...
Point $p$ is called an edge point or contour point if it has at least one white 4-neighbour. A west contour point or left edge point is a point with $n_4$ white, and an east contour point or right edge points a point with $n_0$ white; and so forth.

![Chain code definition for the eight neighbours of pixel p.](image)

Point $p$ is called a "safe point" if one of the following conditions is true:

N1: $N(p)$, the environment of point $p$, matches one of the pattern of neighbours shown in Fig. 3.28 (or, an equivalent rotated pattern of neighbours).

N2: $N(p)$ contains exactly two 4-neighbours.

For example, applying these conditions to a "left edge safe point," is equivalent the result of the following boolean expression is false:

$$n_0 \lor (n_1 \land n_2 \land n_6 \land n_7) \land (n_2 \lor n_3) \land (n_6 \lor n_5),$$

or the following boolean expression being true:

$$!n_0 \lor (!n_1 \land !n_2 \land !n_6 \land !n_7),$$

that means at least one of the following conditions should be satisfied.:

a) $n_0 = 0$;

b) $(!n_1 \land !n_2 \land !n_6 \land !n_7) = 1$, (ie. $n_1$, $n_2$, $n_6$, and $n_7$ be dark);

c) $(!n_2 \land n_3) = 1$, (ie. $n_2$ is dark and $n_3$ is white);

d) $(!n_6 \land n_5) = 1$, (ie. $n_6$ is dark and $n_5$ is white);

![Break point configurations; at least one pixel in each group marked x or y must be non zero](image)

These configurations are illustrated in Fig. 3.29. They are the cases of the configurations shown in Fig. 3.28 for the left edge point; and in each case point $p$ is a
left edge safe point and is not flagged. For example if all pixels marked 'x' in Fig. 3.29 be white, then the pixel p in case 'a' or 'b' is an isolated pixel and is not flagged to be removed; and in case 'c' or 'd' it is an end point that is not flagged. If at least one of 'x' pixels is dark, pixel p is a continuation or break point, is a safe point, and is not flagged.

\[ \begin{array}{ccc} 
  x & x & x \\
  x & p & x \\
  x & x & x \\
\end{array} \quad \begin{array}{ccc} 
  x & 0 & 0 \\
  x & p & x \\
  x & 0 & 0 \\
\end{array} \quad \begin{array}{ccc} 
  1 & 0 & x \\
  x & p & x \\
  x & x & x \\
\end{array} \quad \begin{array}{ccc} 
  x & x & x \\
  x & p & x \\
  1 & 0 & x \\
\end{array} \]

Fig. 3.29: Left edge safe point configurations.

SPTA algorithm like any other sequential thinning algorithm tries to find the pixels \( p \) that satisfy the following criteria [Lam et al. 92]:

1. \( p \) is a black pixel,
2. \( p \) is not an isolated or end point, ie. \( b(p) \geq 2 \),
3. \( p \) is a contour point,

and their deletion would not cause any break in the image. The algorithm uses two raster scans per cycle, where the first is left to right; which is defined by "j = 0" in Fig. 3.30, and the second scan is top to bottom, that is denoted by "j = 1" in Figure.

In each scan, each point \( p \) is flagged if it is not a safe point. In the first scan, left edge points are checked if they are not safe then are marked; then right edge points that are left edge safe points are checked and if not safe, are flagged. At the end of each iteration all flagged pixel are removed from the image. This cycle is repeated till no more points are flagged. Results of the thinning algorithm applied on image of Fig. 3.26, are shown in Fig. 3.31.
Thin(Image *image)
{
    repeat
        flagged_pixels = False;
        for (j = 0 and 2) do Skeletonize(image);
    until ( NOT flagged_pixels);
}

Skeletonize(Image *image)
{
    for (each pixel P of image) do
        if( P is dark and unflagged and is an edge_point(P))
            then if( is not_safe_point(P))
                then do { flag (P); flagged_pixels = True; }
                else if ( P is not flagged ))
                    then mark (P);
}

Bool edge_point(Point P)
{
    // Edge point is a point with at least one OFF 4-neighbour,
    // in the right, left, up or down of it.
    if ( j == 0)
        then return(n0\n4 is OFF) // Check right and left neighbours.
    else return (n2\n6 is OFF) // Up and down neighbours.
}

Bool not_safe_point(Point P)
{
    // Safe points are such that deletion of them do not cause a break
    // in the image.
    if ( j = 0) // Row major scan
        then if (n0\n1\n2\n6\n7\n2\n3\n6\n7\n2\n3\n6\n7)
            then return TRUE;
        else return (n4\n3\n2\n6\n5\n2\n1\n6\n7); 
    else if (n2\n3\n4\n0\n1\n4\n5\n0\n1)
            then return TRUE;
    else return ((n6\n5\n4\n0\n7\n4\n3\n0\n1));
}

Fig. 3.30: Thinning Algorithm.
3.6 Line Following

A variety of techniques for extracting lines (contiguous sequences of set, i.e. dark, pixels) from a thinned image have been reported in the literature (Black et. al. [81], Clement [81], Pavlidis [84], [86] and Nagasamy [89]). The line following algorithm used here is simple, easy to implement and quite effective. The algorithm takes a bit image as input and generates a list of "lines" as its output, each of these lines consists of a sequence of points corresponding to pixels that are set in the bit image. Thus, for the line image in Fig. 3.32 the algorithm will find five lines as shown.

The algorithm uses a raster scan approach for scanning each row (or, optionally, column) of the image from left to right (or, top to the bottom). In each row scanned, successive pixels are checked and if set, have their point positions added to one of the lines. Initially the set of lines is empty. The first point added results in the creation of a new line structure. In the example shown in Fig. 3.32 the pixel at point 'a' would be the first encountered and it would form the start of the first generated line. As successive rows are scanned, additional set pixels are encountered and their points are checked to determine whether they extend an existing line — a point extends an existing line if it is adjacent to the end point of that line. The next few pixels that would be encountered in the image as shown would all extend the line starting from point 'a'.

Fig. 3.31: Thinned Image of the Circuit Diagram.
When the pixels at point 'b' or 'c' are encountered, they have to be used to start new lines because they are not adjacent to the end point of an existing line such as 'a's line. Depending on the order in which the lines are checked, point 'd' may form part of the line starting from 'b' or that starting from 'c'. As the algorithm checks the lines in the reverse order (i.e. from last to the first), pixel 'd' is added to the line that originated at 'c'. The line starting at 'c' will continue down the branch below 'd'; while the line from 'b' will terminated. At 'e', the only line that can accept the point is that line originating at 'a'. The line from 'a' now continues along this right brach; it will eventually terminate at the point before 'f' (according to the algorithm, 'f' "belongs" to the line starting at 'c').

In the next scan row, the point just below 'e' would have to form the start of a new line. Finally the point 'g' is supposed to be a continuation of the line from 'c', because this line is the last line that has a point neighbour to this point and is not yet closed. This line would then extend the right branch to point 'h', and the down branch makes a new line. Fig. 3.33 shows the structure used to represent the lines for this line following algorithm.
At the end of the main raster scan process it is necessary to verify the line segments and check for joinable short line segments. These typically occur with "staircased lines". For example when the slope of a digital line segment is between 0 and 45 degree, the stairs of pixels are as shown in Fig. 3.34. If these are scanned horizontally, from top to bottom, the group in the first row forms one "line", those in the second row form another "line" and so forth.
Fig. 3.35 illustrates the "verification algorithm" that filters the list of line segments resulting from the raster scan by joining appropriate segments. Two segments are "joinable" if they have neighbouring endpoints. Any segment that is joined to the other would be removed from the list. In this way all fragmented line segments, such as illustrated example in Fig. 3.31 (thinned image of the diagram), would be rejoined.

```c
Line_List* Follow_Lines(Image *image)
{
    list = create(new Line_List);
    for (each dark point of the image) do
        Present_to_(list, point);
    Verify(list);  // join splitted segments.
    return list;
}

void Present_to_(Line_List *list, Point* point)
{
    for (each line L1 in the list, from last to the first) do
        if (line accepts point as its continuation)
            then append point to L1 and break the loop;
        else;
        if ( no line accepted the point )
            then create a new line and append point to it;
        else;
}

void Verify(Line_List *list)
{
    for (each line L1 in the list) do
        for (each line L2 in the list, other than L1) do
            if (Are-Joinable(L1,L2))
                then Join(L2,L1) and remove L2 from list;
            else;
}
```

Fig. 3.35: Algorithm for finding contiguous points of thinned image.
As well as dealing with "staircased" straight lines, the verification algorithm can also combine some short segments to obtain closed loops. Fig. 3.36, illustrates the case of a circle that the 'following' algorithm has split into five "lines" (pixel runs). The verifying routine will rejoin these parts; firstly, parts 2, 4, and 5 would be joined to parts 1 or 3, and finally these two parts (1 and 3) are joined together and make a complete closed loop.

![Diagram of circle with labeled parts](image)

Fig. 3.36: Different parts of a circle to be verified.

### 3.7 Line Vectorization

The "line following" process converts a thinned bitmap image of a diagram into a list of "lines" (i.e. chains of contiguous dark pixels). The next step is to extract actual straight line segments, circular arcs, and elliptic arcs from these pixel chains.

This is achieved by a "vectorization" process. The vectorization algorithm identifies "break points" on the chains of contiguous pixels so that the line segments joining these points would closely resemble an approximating curve such as would be drawn by a human interpreting the same information.
A "break point" on a curve is a point at which the curvature of the segment changes. For example in Fig. 3.32, point 'e' is an obvious break point for line segment 1. Actually, there would be several other break points on the "line" from point 'a' to point 'e'. The identification of these points enables one to get a more global picture of the curve than that obtained by considering a small subset of points at a time. This makes it easier to distinguish regions which form part of a circular arc or spline, from those which represent straight line segments.

Fig. 3.27 gave the chain code [Freeman 61] values (0..7) for movements in the 8 possible directions between adjacent pixels. The chain code for a "line" (sequence of contiguous pixels) or say a 'digital arc', extracted from the image is simply a sequence of these values. A digital arc coded with these chains is said to be straight if its chain code meets the following criteria [Freeman 70]:

1) at most two types of symbols can be present, which can differ only by unity, modulo 8; (e.g. 4,5,5,4... for a line heading about 235° or 0,7,0,0,7 for a line heading -15°)
2) one of the two symbols always occurs singly;
3) successive occurrences of the single symbol are as uniformly spaced as possible.

Rosenfeld [74] showed that a digital arc is straight if, and only if, it has the 'chord property'. A digital arc has the 'chord property' if for every two digital points c and d of arc, and for each point p: (x,y) on line segment from c to d, there is a point e: (h,k) of arc such that distance between p and e is less than unit.

The algorithm used here is based on searching for 'uneven segments' [Hung 85] on a digital curve. Hung's definition of a 'segment' is: 'A segment is a subsequence of any sequence of symbols such as a digital arc chain code'; Hung also defines the 'sum of the segment' as the sum of the values of the subsequence. Two segments are called equal if the number of their symbols is the same. He also has defined 'an uneven pair
of segments' as two equal segments with their sum differing by more than 1. Hung has proved that 'A digital arc has the chord property if and only if there is no uneven pair in its chain code'.

For example if the segments of length '1' ('CC' or 'S1' in the first row of codes in Fig. 3.37) are considered, they do not immediately identify which part of the digital curve is straight. However, using these segments, it is obvious that the last part of the example shown in Fig. 3.37 from the 15th point, is not in the direction of the first part (because its values are differing from '0' and '1' and so violate Freeman's criterion 1). So the 15th point is critical. But it is not certain that the first part of the example is straight, until the higher level segments as shown in figure have been calculated. Segments with length 2, that make the sequence 'S2', are coded as successive numbers (ie. 0 and 1) in the segments of length 1. The sequence S2 still does not determine whether the segment is straight. But similar calculation steps can be repeated. By the time sequence 'S5' is generated it is apparent that there is another break point on the curve at the 7th point. The curvature of the line changes its value around point 7. The point joining of the segments is marked as another critical (break) point. Checking the "S" sequences will continue at most as long as half length of the line segment.
void Mark_Critical_Points(Line_List *llist)
{
    for (each line segment 'L' in the llist)
    {
        CC = Chain_Code(L);
        startp = 1; // Start from the first point.
        endp = No of points on L;
        while (Found_Uneven_Segments(&CC, &startp, &endp))
        {
            mark_point(L, endp);
            startp = endp;
            endp = llist->_curl->_pno;
        }
    }
}

Bool Found_Uneven_Segments(int* CC, int* startp, int* endp)
{
    g = (*endp - *startp); // No of pixels to be checked.
    G = g / 2; // At most segments with length g/2 are checked.
    Initialize(Sj);
    j = 1; // Test for 1 length segments.
    while (True) do
    {
        if (j > G ) then return False;
        for (i = 1; i <= g-j+1; i++) do
        {
            Sj[i] += CC[j-1+i+startp-1]; // Segments with length j.
            i = 1;
            T1 = Sj[i]; // First value of symbols.
            T2 = -1; // Initial value for next.
            if (j == 1) then { Al = 1; // Number of T1 symbols.
                A2 = 0; } // Number of T2 symbols.
            while (True) // Check segments with length j
            {
                i++;
                if (i > g-j+1 ) then return False;
                if (T2 != -1 ) then break while loop;
                if (Sj[i] == T1) then (i++; continue; )
                if (Sj[i] == T1+1) then (i++; continue; )
                if (Sj[i] == T1-1) then (i++; continue; )
                i = 1;
                *endp = *startp+i-1; // There is a critical point.
                return True;
            }
            while (i <= g-j+1) // Count no of T1 and T2 symbols.
            {
                if (Sj[i] == T1)
                    if (j == 1) then A1++;
                else if (Sj[i] == T2)
                    if (j == 1) then A2++;
                else { *endp = *startp+i-1; break while loop; }
                i++;
            }
            if ((j==1) && (A1==1 || A2==1)) then return False;
            j++;
        }
    }
}

Fig. 3.38: Vectorization Algorithm.
3.8 Extraction of "Primary Elements"

All diagram interpretation systems will use multiple levels of image representation. The actual bitmap is at the lowest level, next is a model that uses "lines" (chains of pixels) extracted from a cleaned up bitmap, then the "vectorized" model that describes the image in terms of many, often short straight line segments. However, even a vectorized description is still too low-level for convenient use in an interpretation system.

An image interpretation system will look for "components" (naturally, these will be domain specific). It is rarely convenient to describe such "components" in such low-level terms as "find a line ... that touches another line at angle ... that joins to line ...". Instead, a further level of abstraction is needed. Consequently, an additional interpretive level is added.

In this level, primitive features like the short straight lines of the vectorized model are replaced by larger composites, here called primary elements. The primary elements used will be somewhat domain dependent. For the trial application involving electrical circuits, the primary elements needed included triangles, circles, rectangles, semicircular arcs etc. This section presents the mechanisms that were used to extract the various primary elements from the vectorized image.

3.8.1 Arcs

Arc fitting starts once the image has been reduced to vectorized lines. Each of the pixel chains identified earlier will now be characterized by a sequence of break points that split them into separate vectors. A few of the pixel chains will themselves represent digitized arcs. In other cases only subparts of a pixel chain, involving a few of its component vectors, will form an arc.
In the first step of fitting circular arcs to the line segments, each pixel chain is checked to see if it would be approximated by an arc with start and end points at the start and end points of the pixel chain. If a pixel chain can be approximated by a circular arc, its properties such as start and end angles, centre of the circumscribing circle and its radius, are calculated and the pixel chain is marked as an arc.

However, if a complete pixel chain does not fit initially to an arc, then the analysis can be repeated with the same pixel chain minus its first (last) vector segment. In fact, the process can be repeated recursively with successive elimination of end vectors until either an arc is identified or the segments are single vectors. The two phases of this algorithm will extract all combinations of different circular arcs.

For example, in the case shown Fig. 3.39, the algorithm results in successive vector segments being dropped from the pixel chain until the chain contains just the 4-5-6 segments. These can be recognized as forming part of a circular arc. The original segments, 0 to 4 are reanalyzed (in reverse) and another arc comprising 4-3-2-1 is isolated. Finally, the last of the vector segment (0) can be marked as a straight line segment.

![Fig. 3.39: Two Connected arcs with a straight line segment.](image)

If there are straight parts in the segment between the arcs in the line being analyzed (e.g. see Fig. 3.40), the algorithm will extract trailing or heading arcs as discussed
before, until there are two long straight parts left at the head and the end of the segment. At this stage the first part of the segment is supposed to be a straight line and is removed from the segment ( '0' to '1', or '4' to '5' segment). Then the algorithm will be repeated for the rest of the segment.

**Fig. 3.40: Two Connected arcs via a straight line segment.**

Find_Arcs(Line_List *list, int W)
{
    for (each line segment "L" in the list) do
    { // Leave short segments and those defined before.
        if (L->_pno < W || L->_type != NONE) continue; // first point of the line;
        p1 = 1; // last point of the line;
        p2 = L->_pno; // While all cases be checked.
        while (True) { if ((p2-p1) < W) // Is too short from p1 to p2?
            then break;
            if (IsArc(L, pi, p2)) // from point p1 to p2
                then
                    { Split(L, pi, p2); // Check from end to start).
                        Mark(L, ARC); // Move forward one break point from p1
                        Add the trailing segments to the list for later check;
                        break; // check next line.
                    }
                else // Is not ARC from p1 to p2.
                    { // Move back one break point from p2.
                        p2 = PrevBreakpoint(L); // Move forward one break point from p1
                        if (p1 == p2) // start to end cases checked.
                            then // Check from end to start).
                                { // Move forward one break point from p1
                                    p1 = NextBreakpoint(L);
                                    if (p1 != L->_pno) // All cases not checked yet.
                                        then p2 = L->_pno;
                                    else break; // All cases are checked.
                                }
                            }
                    }
    }
}
3.8.2 Circles

As explained in the section on line following (section 3.6), the initial line extraction process is followed by a filtering step that identifies groups of lines that might represent closed loops. Any closed loops so discovered are candidates for promotion to "circles". The "Find_Circles" routine will check all closed loops in the list of line segments, and finds those that can be matched to circles. A closed loop is considered to be a circle if all its constituent points lie within set threshold limits of points at a constant radius from the estimated centre point of the closed loop.

Find_Circles(Line_List *list)
{
    for (each closed segment in the list) do
    {
        L = Last point of the closed segment;
        M = Median point on the segment;
        C = (M + L)/2; // midpoint of the points L and M.
        radius = distance(C, M);
        Es = 0; // Error estimation.
        for (each point on the segment) do
            Es = Es + abs(radius - Distance(point, C));
        if (Es <= Threshold)
            then Make_Circle(line);
    }
}

Fig. 3.42: Circle Finding Algorithm.

Other circles (those that are not quite complete) can sometimes be found by joining arcs.

3.8.3 Triangles

Triangles are found by a simple combinatorial search through the list of straight line segments identified by the vectorizer (see algorithm in Fig. 3.43). For any given line as the starting point, the algorithm looks first for a joinable second line (i.e. one sharing an end-point with the first line), and then seeks a third line that joins both existing lines
(A Y-shaped junction would satisfy these requirements, so there is an additional check to eliminate cases where the three lines "join" at the same point.).

Find_Triangles(Line_List *list) {
    First = first line in the list;
    while (First is valid) do {
        Second = next line to the First line, in the list
        while (Second is valid) do {
            if (Are_Joinable(First, Second))
                then {
                    Third = next line to the Second line in the list;
                    while (Third is valid) do
                        if (Are_Joinable(First, Third)
                            AND Are_Joinable(Second, Third)
                            AND NOT Joined_In_One_Point(
                                First, Second, Third))
                            then Make_Triangle(First, Second, Third);
                            else Third = next line to the Third line in the list;
                }
            Second = next line to the Second line in the list;
        }
    First = next line to the First line in the list;
}
}

Fig. 3.43: Triangle Finding Algorithm.

3.9 Circuit Schematic Diagrams Interpretation

Fig. 3.44 illustrates the "blackboard" system that has been developed for interpreting diagrams of electrical circuits.

The lowest level of the blackboard holds the (cleaned up) bitmap image. The next level holds primitives like short vectors and closed loop groupings. Then there is the primary elements (or primary shapes) level, the "circuit elements" level and, finally, a "connection table" level where the circuit is represented in terms of components and connections and from where this data could in principle be loaded into some form of circuit simulator.
The various "knowledge sources" (KSs of Fig. 3.44) work between levels. The lower level knowledge sources were composed using the various routines described in previous sections.

The two highest level knowledge sources convert primary elements (triangles, circles etc) into circuit components, and then complete a connectivity analysis of these components so as to obtain the information needed to fill in a connection table representation of the circuit.

In this preliminary study, these final level knowledge sources were hard coded. The approaches used are presented in this section.
3.9.1 Perception of Circuit Diagrams Primitives

In this section the primitives (symbol) perception algorithms for circuit diagrams, are discussed. These algorithms are designed to act as a part of the "circuit elements" knowledge source which is a part of the whole blackboard model (Fig. 3.44). Each algorithm is implemented to perceive one kind of the circuit symbol.

Perception of Capacitors

Capacitors usually are shown by two short parallel line segments or by two filled narrow parallel rectangles, as shown in Fig. 3.45. The preprocessing steps (isolation and then contour tracing of solid areas) converts the filled rectangles to regular rectangles. These rectangles are found in the image by algorithms discussed in the previous section. The algorithm in Fig. 3.46, searches for two such rectangles or for two short line segments which are parallel and could make a capacitor symbol.

The descriptions of capacitors involve elements from more than one level of the blackboard (e.g. lines from the "primitives" level and rectangles from the "primary elements" level). The program makes use of a list that contains pointers to all components irrespective of their "level". Entries are removed from this list when their interpretation is finalized.

![Symbols used to show Capacitor.](image)

Fig. 3.45: Symbols used to show Capacitor.
Find_Capacitors(Line_List *list, Elem_List *elems)
{
    L1 = first line in the list of line segments;
    while (L1 is valid) do
    { if (L1 is a Rectangle OR is Horizontal OR Vertical line)
      then
      { L2 = next line to the L1;
        while (L2 is valid) do
        { if (L2 is same as L1 AND they are near by a threshold)
            then if ( there is a line(L3) connected to L1
                    AND there is a line(L4) connected to L2)
                then
                { Append(Capacitor(L1, L2, L3, L4), elems);
                    break this loop; }
        } }}
    L1 = next line to L1;
}

Fig. 3.46: Perception Capacitors Algorithm.

Perception of Amplifiers (OP-Amps)

Amplifiers are represented by a triangle with three incident lines; one line attaches to a vertex, the other two to the side opposite that vertex. Two cases of an Amplifier are shown in Fig. 3.47. Algorithm 3.48, tries to find amplifiers on the circuit diagram and insert them in the list of found symbols.

Find_Amplifiers(Line_List *list, Elem_List *elems)
{
    for (each L1 in the list) do
    if (L1 is a Triangle with a Horizontal OR Vertical side)
    then
    if ( there are two lines(L2, L3) connected to H/V side
        AND there is a line(L4) connected to opposite corner)
    then
    Append(Amplifier(L1, L2, L3, L4), elems);
}

Fig. 3.48: Amplifier Perception Algorithm.
Perception of Resistors

In the circuit diagrams in the data set used for input, resistors are actually shown by a number of different symbols, as illustrated in Fig. 3.49. Algorithm 3.50 searches the blackboard to find block resistors or zigzag one, and append them to the list of elements found in the diagram.

![Symbols used for Resistors.](image)

Find_Resistors(Line_List *list, Elem_List *elems)
{
    Find_Block_Resistors(list, elems);
    Find_Zigzag_Resistors(list, elems);
}

Find_Block_Resistors(Line_List *list, Elem_List *elems);
{
    for (each L1 in the list) do
        if (L1 is a Rectangle with valid length and width)
            then if( there are (L2, L3) connected to L1 oppositely)
            then append(Resistor(L1, L2, L3), elems);
}

Fig. 3.49: Symbols used for Resistors.

Fig. 3.50: Resistor Perception Algorithm (cont.).
Find_Zigzag_Resistors(Line_List *list, Elem_List *elems);
{
    Make an empty list (Rlist) to store found proper primitives;
    L = first Diagonal line in the list;
    if (there is no L) then return;
    add(L to the Rlist);
    Mark L in the list;
    L = Unmarked diagonal line of the list, next to L;
    if (there is no more Unmarked L) then return;
    while (True) do
    {
        add(L to the Rlist); // L is joinable to Rlist members.
        Mark L in the list;
        if (number of Rlist members is less than 5)
        {
            L = first member of the Rlist;
            L = Unmarked diagonal line of the list, next to L;
            if (there is no more Unmarked L)
                then return;
            else continue while loop;
        } else
            // at least five elements are found.
            for (each two contiguous Horizontal/Vertical lines
                L1 and L2 in the list)
            do
            {
                if (L1 is joinable with a member of Rlist
                    and L2 is joinable with another one)
                    then
                    { append(Resistor(Rlist, L1, L2), elems);
                        empty(Rlist);
                        break for loop;
                    }
            }
            if (Rlist is empty) // One resistor is found.
                then
                { L = First Unmarked diagonal line in the list;
                    if (there is no more Unmarked L) then return;
                    add(L to the Rlist);
                    Mark L in the list;
                }
            else
            { L = first member of the Rlist;
                L = Unmarked diagonal line of the list, next to L;
                if (there is no more Unmarked L) then return;
            }
    }
}

Fig. 3.50: Resistor Perception Algorithm.
3.9.2 Connection Table

The final step in the interpretation process is the construction of a connection table representing the circuit. The algorithm shown in Fig. 3.51 finds all connection points on the diagram. It tries to find all straight line segments to connecting directly or indirectly to each circuit component.

```
Find_Links(Line_List *list, Element_List *elems)
{
    make an empty list (Links) for storing found links;
    for (each element in the elems list)
        do
            for (number of element's links)
                do // eg. two for resistors.
                    while (True) do
                        { //eg two for resistors.
                            for (each line L1 in the list)
                                do
                                    if (L1 connects to the element) then break;
                                if (NOT breaked for loop)
                                    then send an ERROR message and break while;
                            for (each line L2 in the list)
                                do
                                    if (L2 connects L1 and is not part of an element)
                                        then add(L2, Links(L1));
                            for (each line L3 in the Links(L1))
                                do
                                    for (each line L4 in the list)
                                        do
                                            if (L4 connects to L3 and not found before)
                                                then add(L4, Links(L1));
                                }
                            }
    }
}
```

Fig. 3.51: Algorithm to find connections.

3.10 Discussion

The main aim of this initial phase of the research was to develop new algorithms for the various image preprocessing operations required and to implement efficient versions of new and some existing algorithms. This work was accomplished successfully and lead to a suite of routines that could be used for image clean up and analysis.
The effectiveness of these preprocessing routines was evaluated empirically in preliminary attempts to build an interpreter. As well as testing the image analysis routines, this primitive interpreter provided experience in dealing with problems arising with real data.

The interpreter was limited in aim. It was designed only for the special purpose of interpreting simple circuit diagrams such as the example shown in Fig 3.52.

The preprocessing routines, as described earlier in this chapter, could convert this image to a "thinned", "cleaned up" image (Fig. 3.53). This was further analyzed to obtain lines, rectangles, triangles etc that could be interpreted as the components and interconnecting wires as shown in Fig 3.54.
Fig. 3.54: Elements found in the diagrams and their connecting lines.

The final stage of interpretation lead to a connection table representation of the circuit, summarized in part in Fig. 3.55.

![Connection Table](image)

Although the interpreter did work successfully on a number of simple circuit diagrams, it was limited in many respects. For example, the program would fail if a circuit contained any element apart from the limited set of target symbols (pre.-amps, resistors, etc) that had interpretation rules coded. Additional symbols could be accommodated by reprogramming, but this was highly inconvenient because it involved coding in C and defining complex relations over primitive elements such as lines and arcs.
Further, minor distortions in an image (typically arising during the scanning process) could cause changes in the shape of the original components. These changes could then result in differences in the sets of lines and arcs extracted from the image and consequently could cause the interpreter to fail. For example (as shown in Fig. 3.56), if the line segment from point 'a', through 'e', is vectorized as a circular arc, the interpreter would not perceive the "rectangle" and so would not identify the symbol as a resistor.

![Diagram](image)

Fig. 3.56: Distortion makes unexpected results.

It is futile to try to avoid such problems by defining additional routines for recognizing "distorted symbols". ("A distorted rectangle is a box with three sides and a the fourth side partially closed by an arc.") There is no limit on the range of possible distortions and so no limit to the number of different special rules that one would require if one attempted this approach.

Of course, the fundamental problem with this interpreter was that it was completely hard programmed and domain specific; consequently, it could not be used for any other kind of line diagrams.

These limitations of the simple system were expected. It provides experience that leads forward to the more general system described in the following chapters.
4. The General Interpreter

4.1 - Introduction

This chapter presents a generalized line diagram interpreter. Very few such systems have been reported in the literature. Filipski and Flandrena [92] have given details of a prototype for a commercial system. The system combines a scanner, a PC workstation, and a specially designed hardware recognition module. This hardware recognition module is a multiprocessor engine running software to transform pixel representations of engineering drawings into CAD files. After standard preprocessing (contour tracing, thinning, text separation, vectorization), the context processing phase attempts to construct groups of objects, such as text strings from characters and symbols from vectors. Their work focuses primarily on text recognition, however they have used a blackboard model and a LISP like language to define rules for symbol recognition via a matching algorithm. Matching rules often fail because of distortions in the images. The system allows for interactive editing of rules. When failures occur, the operator may intervene and define new rules that deal with the distorted data.
Pasternak and Neumann [93] have developed a system for interpreting engineering drawings, schematic diagrams, and maps. They have devised a special purpose language for describing operations on graphics primitives and have shown that their system is capable of handling many kinds of line diagrams, such as those which occur in engineering drawings.

In section 4.2, structures used for designing the line diagram interpreter are introduced. Section 4.3 will illustrate the overall system structure. Section 4.4 contains a discussion of the algorithms used for finding and matching defined templates with the primitives found from the diagram.

4.2 - Definitions

Shapes of components in line diagrams can be described in different ways. One may use geometrical feature representation techniques — describing shapes in terms of line segments and singular points such as branch points and crossings [Cox et al. 82]. Alternatively, one may use quasi-topological [Nishida and Mori 92, 93] descriptions — defining shapes by loops, their convexity/concavity, and connectivity. Nishida and Mori [93] have defined the following criteria for a good definition of primitives and shape description methods for character recognition:

1: Primitives must be simple enough to be computed easily [Pavlidis 72].
2: Algebraic structure (the way primitives are sequenced) is introduced to higher level (global feature) as well as lower level, representation.
3: Systematic rules are presented to integrate local features into global description, and the type of global feature described by the description scheme should be clear.
4: Any continuous curves can be represented by the description scheme. Few methods satisfy these four criteria [Pavlidis 77, 80].

5: Geometrical and quasi-topological feature representation are integrated [Nishida and Mori 92].

6: The shape description is invariant to scaling and translation operations.

As this work is concerned with the interpretation of line diagrams, criterion 6 has to be extended to include invariance with respect to rotation operations. For example a schematic diagram symbol can be drafted in different directions, so the shape description schema should be invariant to rotation.

4.2.1 - Attributed Relational Graphs and Structures

One way to describe shapes is to use attributed relational graphs. An attributed relational graph (ARG) is an attributed and weighted graph denoting by:

\[ \text{AG} = (p, a, r), \]

where \( p = \{1, 2, ..., m\} \) represents a set of m nodes (primitives); \( a = \{a_i | i \in p\} \) is a set of attributes (unary relations [Li 92]) for each node in \( p \), in which \( a_i = [a_{i1}, a_{i2}, ..., a_{ik}] \) is a vector consisting of \( k_1 \) different types of attributes; \( r = \{ r(i,j) | (i,j) \in p^2 \} \) is a set of relations, defined over \( p^2 = p \times p \) in which \( r(i,j) = [r_{11}(i,j), r_{22}(i,j), ..., r_{k2}(i,j)] \) is a vector consisting of \( k_2 \) different type of relations. Thus every node in an ARG has its own attributes and relations with every other node in the graph. The order of an ARG is two, as the first level are the primitives and their attributes, and the second level are couples of the related primitives.

An attributed relational structure (ARS) of order \( n \) (defined in \( n \) levels) as an extension to an ARG, is a \( n+1 \)-tuple denoted by:
\[ g(p, r_1, r_2, ..., r_n) \]

In this notation \( p = \{1, 2, ..., m\} \) represents a set of \( m \) primitives;

\[
 r_n = \{ r_n(i_1, i_2, ..., i_n) \mid r_n(i_1, i_2, ..., i_n) \in p_n \}
\]

is a set of \( n \)-ary relations defined over \( p^n \) in which:

\[
 r_n(i_1, i_2, ..., i_n) = [r^1_n(i_1, i_2, ..., i_n), r^2_n(i_1, i_2, ..., i_n), ..., r^k_n(i_1, i_2, ..., i_n)]
\]

is a vector consisting of \( k_n \) different types of \( n \)-ary relations.

In the rest of this section, \textit{Primitives, Relations, Correlations} (interrelation of simple relations) are introduced.

\textit{Primitives} are the straight line segments, circular arcs, and circles that can be extracted from an image in the early stages of processing.

\textit{Relations}, in this first stage, defines the constraints on pairs of primitives. For example two line segments may be required to be parallel, or to make a "T" junction with each other, and so on. At higher levels of the system, \textit{relations} can be defined between more complicated elements of the line diagram.

An \textit{Correlation} defines an additional constraint attribute on primitives already restricted by \textit{relation} constraints. For example, if "L1 C L2" defines that line segment "L1" is connected to "L2", and "L1 C L3" defines a connection between "L1" and "L3"; a \textit{correlation} constraint would be something that, for example, specifies that the connecting point of "L1" and "L2" is the same (or is not the same) as the connecting point of line segment "L1" and "L3".
4.2.2 - Primitives

It is assumed that any line diagram will be capable of being described in terms of the following *primitives*: a) digital straight line segments, b) circular arcs, and c) circles.

A straight line segment has a finite set of attributes; including at least the end-points (p1 and p2 in Fig. 4.1) of the line segment and type tag that characterizes a line as being "horizontal", "vertical", or "oblique". The system actually represents lines as instances of a C++ class. In addition to the data members, several member functions are defined.

One member function for the line class returns the slope of a line segment. The slopes of "horizontal" and "vertical" tagged lines are, by definition, zero and "maxint" respectively.

The slope of an "oblique" digital line is calculated in an unusual way. The "non standard" calculation is necessary to accommodate digital lines. Consider for example the digital line segment L shown in Fig. 4.2 as a sequence of black squares. It can be seen that there is no exact slope for it. Instead, two slopes (e1, e2) are defined with these slopes providing a bound on the real slope of the line. Krishnaswamy and Kim [87] have proved that: e1 < slope of L < e2, where e1 is the slope of the line through pixels such as those shown as b and c in Figure 4.2 and e2 is the slope of the line.

![Fig. 4.1: Straight line segment and its attributes.](image)
through pixels $a$ and $d$. Krishnaswamy and Kim provide an algorithm for finding the appropriate pixel points.

Another member function of the class line is the "on" function. A point is called "on" a line if it lies between the end points somewhere on the line segment; and is at least a minimum distance from an end point (this distance is determined by a threshold value "w" which relates to the line thickness of the original image). Those points of a line segment that satisfy the "on" function, are called the "interior points" of the line segment. Another member function of class line is the "length" function which returns the distance between two endpoints.

To provide the capability of defining the direction of a line segment, one of the end points of the line segment is defined as "start " and the other as "end" point. This definition makes it possible to define proper relations among the line segments as needed to describe any particular object combined of these lines. Different cases of the orientation of the "start" and "end" point of line segments are illustrated in Fig. 4.3.
Fig. 4.3: Defined "start" and "end" points for the straight line segments.

Circles found in the image of the line drawings are defined by their centres, O(xc, yc); and their radii R (Fig. 4.4). Class circle has a member function "tangent" which returns the tangent line at a given point "p(x,y)" on the circumference of the circle. This tangent line is used for calculating the "angle of the connection" in all those relations that require a contact between circles and other lines, arcs or circles. The member function "on" is also applicable for circles. A point "p (x,y)" is called "on" a circle "C(O,R)" if "d" is the distance of "p" from "O" and R-w < d < R+w.

Fig. 4.4: Circle primitive and its attributes.

Circular arcs are defined in terms of their centre, radius, start point angle, and arc angle (or "length"). The starting angle may be positive or negative, for example in Fig. 4.5, the start angle could be given as -45° or 315°. The arc length is negative if the direction of the arc is supposed to be clockwise; otherwise it is positive. In Fig. 4.5, if the start of the arc is supposed to be "p1" then the arc length would be positive. In addition, the end points of the arc are stored. They may be needed in later checks for connections.
with other primitive graphic elements. One of these two end points is marked as "start" and the other as "end" point, according to the arc chord (see Fig 4.3). The member function "on" is also defined for circular arcs. A point is "on" the arc if it is a point of the arc, and is at least \( w \) pixels far from each end point of the arc.

![Circular arc and its attributes.](image)

Preprocessing frequently results in a large number of very short segments. These may actually be parts of arcs, or parts of straight lines (that have been "staircased" by digitization etc.). Rather than being tagged as horizontal, vertical, or oblique these very short segments are termed "line". They are sometimes useful when working with cursive script writings. With cursive script it is often necessary to define the various forms of accent marks etc. that can be placed around a character. In a typical digitized image, such marks appear as tiny line segments in essentially arbitrary directions. While some of these short segments are significant in their own right; most tend to be noise or are parts of other elements that have somehow been fragmented by image scanning and preprocessing steps.

**Primitives** are defined by the structure shown in Fig. 4.6. Some graphic composites (such as squares, schematic diagrams element, different boxes in flowcharts, etc) are also defined using this simple structure. **Type** defines what kind of primitive the element is. An element's type may be found from the image by the preprocessing
routines. Where the same "Elem" structures are used for more elaborate composites (e.g. decision boxes, squares etc) the type will be assigned when matching a combination of primitives according to the user defined templates. These types all have to be given a name, such as resistor, or decision box, etc.

```c
struct Elem
{
    short type; // "horizontal", "oblique", "line", "arc",
    // "circle", "square", ...
    short lb; // Lower bound range on length of line, arc,
    // or radius of circle,
    short ub; // Upper bound. If lb or ub is defined "0",
    // it is not significant.
    short loc; // Is element over or under a baseline?
}
```

Fig. 4.6 : Structure for declaration elements of a diagram.

The `lb` and `ub` properties define a range of allowable lengths (in pixels) for a line, or an arc (in degrees), and or a circle's radius (in pixels). The value "0" indicates that there is no limit on that primitive. The `loc` property is often used to define the location of a primitive relative to the baseline of text when carrying out character recognition. It will have the value `over = 1` or `under = 2`, depending on whether the primitive (or any more complicated element) is located over the baseline or under it. In other situations, this field is left as 0.

The following examples illustrate the role of these properties. The first example defines a horizontal line whose length should not be less than 10 pixels, and it is not important where it is located in regards to the baseline: "H 10 0 0". A second example is "C 6 10 1", this defines a circle with a radius length between 6 to 10 pixels, located above the baseline. The third example specifies a requirement for a 90° arc in length (with 5° tolerance) which starts from 10°: "A 10 100 0".

82
4.2.3 - Relations

A relation defines some constraint on the placing of two primitive components \( P1 \) and \( P2 \). Higher levels in the system use relations defining constraints on the location of pairs of composite structures, and constraints combining primitives and partially assembled composite structures. Fig. 4.7 shows the structure used to encode relations. The type property specifies the kind of constraint; for example "parallel" or "above". The next two properties (\( elem1 \) and \( elem2 \)) are the identifiers of elements involved in this relation. Each relation, has two other properties — lower bound: \( lb \) and upper bound: \( ub \). These are used to characterize limitations such as a range for the angle between two connected line segments. They may also be used to indicate the way that two primitives connect. For example "H" as the \( lb \) and "T" as \( ub \) mean that the \( elem1 \) is related by its "start" point to the "end" point of the \( elem2 \), some examples are given later in this section.

```c
struct Rels {
    short type; // "parallel", "contacts", "tjunc", "includes", // "above", "close",...
    short elem1; // Identifier of first related element by this // relation.
    short elem2; // Other element participated in the relation.
    short lb; // Lower bound of the defined range.
    short ub; // Upper bound of the range.
        // (not significant if lb or ub is "0").
}
```

Fig 4.7: Structure to declare the relation between two elements.

Some examples of the defined relations follow. In all examples it is assumed there are two defined primitives with "L 10 0 0" structure, which means that the system is looking for any line segment that is at least 10 pixels in length, and whose location is not restricted. The relation "1 2 P 0 0" specifies a "parallel" relationship between two line segments. The relation "1 2 T H 0" describes a tee-junction relation between two line segments, where the first makes a tee-junction with the second at its "start" point.
Currently a set of 17 relations is used to cover most of the common cases needed to describe line diagrams such as flowcharts, circuit diagrams, chemical structure diagrams, engineering drawings and so forth. These defined relations are described in the rest of this section.

**Parallel Relation:**

Two primitives are defined "parallel" if one of the following conditions apply, and in all cases the \( lb \) and \( ub \) (see Fig. 4.7 Rels structure) should be defined as "0":

a) Two line segments are "parallel" if both are horizontal or vertical; or both are oblique with nearly the same slopes (Fig. 4.8a).

b) A line segment is "parallel" to an arc, if it is "parallel" to the tangent line on the middle point of the arc, and it lies outside any circle circumscribing the arc (Fig. 4.8b).

c) A line is "parallel" to a circle if it (or its continuation) does not contact the circle (Fig. 4.8c).

![Fig. 4.8: Cases of "parallel" relation with line segment.](image)

d) An arc is "parallel" to a line, if the tangent line at the midpoint of the arc is "parallel" to the line segment, and the line segment is not outside the circle of the arc (Fig. 4.9a).

e) Two arcs are "parallel" if their centres are the same; their start angle is almost the same and they are nearly same length (Fig. 4.9b).
f) An arc is "parallel" to a circle if the centre of the circle and the circle circumscribing the arc are the same (Fig. 4.9 c, d).

![Fig. 4.9: Arc "parallel" cases to other primitives.]

---

g) A circle is "parallel" to a line if the line is "parallel" to it, and similarly for a circle "parallel" to an arc (Fig. 4.8c, Fig. 4.9 c, d).

h) Two circles are "parallel" if their centres are nearly the same (Fig. 4.10a). If there is no connection point between two circles it does not mean that they are "parallel" (Fig. 4.10 b, c).

![Fig. 4.10: "Parallel" relation between circles.]

a) Two circles are assumed parallel; b, c) Not parallel in these two cases.

Angle Between Primitives:

Many relations involve two primitives that cross; often it is necessary to specify constraints on the crossing "angle". The "angle" between two primitives may be constrained to a range from "a1" to "a2". The crossing point could be "on" or "off" the primitives.
In general, two primitives are in "contact" if they have neighbouring (or common) endpoints. The definition of "contact" in the case of a closed loop, such as a circle, is slightly different. The definitions which follow cover all cases that have been considered:

a) Two line segments are in "contact" if either of their end points are neighbours (Fig. 4.11a). In this case the connection angle is calculated by the lines slopes.

b) A line segments and an arc "contact" if either one of their end points are neighbours. The connection angle is calculated by the slope of line and the slope of the tangent line on the arc at the connection point (Fig. 4.11b).

c) In the case of a line "contacting" a circle, it is necessary that one endpoint of the line be "on" the circle. The connection angle is computed as in the previous case (Fig. 4.11c).

d) An arc is in contact with a line (Fig. 4.11a) or with another arc (Fig. 4.12a), if they have a common endpoint.

e) The "Contact" relation of an arc and a circle is like the "contact" relation of the line and the circle (Fig. 4.10b).
f) Theoretically, two circles are in "contact" relation if they connect each other at just one point. However this definition is not applicable with digitized image data, because there may be more than one common point between two contacting digital circles. The "contact" relation of two circles is checked by their radii (R1 and R2), and the distance of the centres (d). If \( d = R_1 + R_2 \) (Fig. 4.13a) or \( d = |R_1 - R_2| \) (Fig. 4.13b) then the contact relation is satisfied. The calculation is done using a threshold tolerance on the match of distance and radii.

If the first primitive is a line or an arc, \( lb \) can be represented by "H" this means that the first primitive is in "contact" with the other one at its "start" point. Setting the \( lb \) as "T" would relate the first primitive to the other by its "end" point. These conditions would apply for the second primitive too by using \( ub \), if it is a line or an arc.

Bicontact Relation:

Two arcs, or a line and an arc are in "bicontact" relation; if both end points are in contact (Fig. 4.14). Cases like Fig. 4.14 "b" and "c" should be defined by more relations, if it is desired.

A line segment or an arc is in "bicontact" relation with a circle if both of its end points are "on" the circle (Fig. 4.15 a, b). If two circles intersect, they have a "bicontact" relation (Fig. 4.15 c). The \( lb \) and \( ub \) are not used in this relation.
Tjunc Relation:

a) A line segment makes "tjunc" with another line or arc if it has an end-point that is "on" the second one (Fig. 4.16a, b, c). Difference between two cases shown in Fig. 4.16"b" and "c", is carried out by another relation which defines wether or not the line segment is located outside of the arc's circumscribing circle.
b) An arc makes a "tjunction" with a line or another arc, if one of its end-points is "on" the line or arc (Fig. 4.17). Like the previous case for line; cases "b" and "c" are distinguished by an extra relation to identify whether "p1" is included in "p2" or not.

![Fig. 4.17: "Tjunc" relation of arc with line or arc.](image)

c) A circle is connected by a line segment or an arc as shown in Fig. 4.18, defines the "tjunc" relation between line or arc with the circle.

![Fig. 4.18: Line and arc in "tjunc" relation with circle.](image)

d) If one of the primitives is line or arc, then its possible to specify whether the junction be near the start or end point (by setting the $ub$ or $lb$ value).

Cross Relation:

a) Two line segments, two arcs, or a line segment and an arc are "crossed" if one of their interior points — i.e. points that are "on" them — is in common (Fig. 4.19).
b) Two circles are "crossed" if they connect at two points (Fig. 4.15c).

![Crossing angle](image)

Fig. 4.19: "Cross" relation cases of lines and arcs.

c) An arc or a line "crosses" a circle, if two of its interior points are in common with the circle (Fig. 4.20)

![Crossing angle](image)

Fig. 4.20: "Crossed" circle with line or arc.

Above Relation:

A primitive is "above" another one if it is entirely located above the horizontal line drawn from the highest point of that primitive (Fig. 4.21). If primitive p1 is "above" p2; primitive p2 is defined to be "below" p1, so there is no need for a separate "below" relation.

![Above relation](image)

Fig. 4.21: "Above" relation between two primitives.
Not Above Relation:

One primitive is "not above" another if there is at least one point of it below the horizontal upper bound of the first one (Fig. 4.22). This relation can be useful when looking for a component that is located near another one. In the figure, none of the first primitives, "p1", are "not below" the second one "p2".

![Not above relation between two primitives.](image)

Fig. 4.22: "Not above" relation between two primitives.

Right Relation:

A primitive is at the "right" hand side of another (Fig. 4.23), if it is completely at the right hand side of the vertical line drawn from the rightmost point of that primitive. If the primitive "p1" is located "right" of "p2"; the primitive "p2" is on the "left" hand side of "p1".

![Some cases of "right" relation.](image)

Fig. 4.23: Some cases of "right" relation.
Not Right Relation

The "not right" relation is defined in the same way as the "not above" relation (Fig. 4.24).

Fig. 4.24: "p1" primitives are "not right" of the "p2".

Continues Relation:

A line segment "continues" another one, if they are collinear even though separated (Fig. 4.25a). Two arcs are "continuous" if their circumscribing circle is the same (Fig. 4.25b). A line and an arc are "continuous" if the line is collinear with the chord of the arc (Fig. 4.25c). The "continues" relation is useful in dealing with images that use dashed lines.

Include Relation:

A primitive or any other constraint is "included" in a circle if it lies entirely inside the circle (Fig. 4.26). A primitive may be both "included" within a circle and be in "contact" with a circle (as shown by examples "a" and "b" in Fig. 4.26).
A primitive like a line segment, an arc, or a circle is "included" in an arc if it lies entirely inside the circumscribing circle of the arc (Fig. 4.27).

Close Relation:

Two lines and/or two arcs are "close" to each other if the distance ("d" in Fig. 4.28) of their middle points is in the described range (a percent of the first or second primitive). For example "p1 p2 X 5 8" means that the middle point of the primitive "p1" is near the middle point of the primitive "p2" (near suggests that the middle point is within 0.5 to 0.8 of p1's length).
Suppose the circle radius is "R" and the distance of the circle centre from the middle point for the arc or line is "D"; then a line segment or an arc is "close" to a circle if "d = D - R" is in the defined range as discussed in the previous example (Fig. 4.29).

Greater Relation

Line segments are compared according to their length in pixels; arcs are compared by their length (arc angles) in degrees. However, if an arc is compared with other primitives; the length of its chord is considered. When a circle is to be compared with another primitive, the length of its diameter is calculated and used for comparison. The "greater" relation actually incorporates a range of ratios (often needed for expressing ideas like "up to twice as big" etc). For example "\texttt{p1 \ p2 \ G \ 15 \ 20}" means that primitive "\texttt{p1}" is at least one and half times greater than primitive "\texttt{p2}" (15 tenth); and at most two times (20 tenth) of its length. If either of the range limits is given as zero, it means that range constraint is not considered.

Equal Relation

Two line segments are "equal" if their length in pixels are almost the same (a threshold has to be considered for all calculations in order to accommodate distortions in the scanned image). Two arcs are "equal" if their length in degrees are nearly the same, and two circles are "equal" if their radii are almost the same.
4.2.4 Correlations

In some cases it is necessary to define more than one relation between the same pairs of primitives. For example, as shown in Fig. 4.30 line segment "L4" has three other primitives related to it. "L3", "L5" and "L6" have made "contact" with it. Suppose three relations are declared to define the triangle shaped by three line segments "L3", "L4", and "L5": R1: "4 3 C 0 0", R2: "4 5 C 0 0", and R3: "3 5 C 0 0". These relations do not adequately distinguish between the triangle arrangement for lines L3, L4, and L5 from the Y-junction relationship that exists among L4, L5 and L6.

![Fig. 4.30: Ambiguity in relation definition.](image)

To avoid the ambiguity inherent in such cases, it is necessary to declare a higher level relation that specifies how two relations may themselves be related. Such a higher order relation is called correlation.

**Correlations** are defined by structures, as shown in Fig. 4.31. Each correlation will have a type (essentially an enumeration with values "same", and "different"). For example, if a correlation is specifying an interrelation between two "contact" relations; "same" means the connection points used for those two relations should be the same point, and the connection points should be different if the correlation value is "different".

```c
struct Cors
{
    short rel1;  // Number of relation as first parameter.
    short rel2;  // Number of second one.
    short type;  // "same" ("S"), or "different" ("D")
}
```

[Fig. 4.31: Structure for declaring correlation of relations.]
4.2.5 Templates

A Template is a schematic definition of a grouping of primitives or any other predefined templates, which may be found on a line diagram. As the template definition is "recursive"; it allows templates to be defined hierarchically. A template has a type to indicate its name for the user; a set of primitives with appropriate attributes (e.g. line not less than 10 pixels), a set of relations between these primitives, and a set of correlations between relations to avoid mismatching and solve any ambiguities among the relations.

Templates have an additional Connections data field. Many templates (e.g. AND-gates, batteries, resistors etc) need to have interconnecting links that will appear as long straight lines in an image. The Connections count and list allow such requirements to be specified in a template. Each template is represented as an instance of a C struct:

```c
struct Template
{
  short type; // "triangle", "square", "hexagon", "resistor", // "transistor" ...
  short elno; // Number of elements making this template.
  Elem *elms;// Array of the element identifiers.
  short rlno; // Number of relations defined between elements.
  Rels *rels;// Array containing relation identifiers.
  short crno; // Number of correlations among relations.
  Cors *cors;// Array of the correlations.
  short lnkno; // Number of template connectors.
  Lnks *lnks;// Array of the connectors.
}
```

Fig. 4.32: Template structure.
Connections

A template may have a number of connectors. The structure used for the connection definition is shown in Fig. 4.33. Connecting lines can be specified as being "inputs" or "outputs" (this does not affect the matching process, instead the information is used when verifying overall connectivity constraints at the highest level of the interpretation process). Examples of templates with connectors are Fig. 4.30 with "input" links (see Fig. 4.30 "L1" and "L2"), and "output" links (Fig. 4.30 "L6"), or none (Fig. 4.37 "P0").

```
struct Llnks {
    short eno;  // The number of element that is defined as link.
    short type; // type = {N, I, 0}. "I" means input link,
                 // "O" output, and "N" none.
}
```

Fig. 4.33: Connection structure.

The next two examples illustrate definitions of simple templates. The first defines a template for a triangle with each side being not less than 9 pixels in length (this might for example represent a 'pre-amp' in some form of circuit diagram).

Example template no. 1

```
// Type or name of the template
Triangle

// Defined three line segments with at least 9 pixels in length
// and no matter where they are.
3 L 9 0 0    L 9 0 0    L 9 0 0

// Each element makes corner with the others.
// "0 1 C 0 0" means L0 and L1 are connected in one endpoint.
3 0 1 C 0 0    0 2 C 0 0    1 2 C 0 0

// "0 1 D" means the connection points in relation 0
// and 1 are not the same.
3 0 1 D    0 2 D    1 2 D

// No connector is defined.
0
```

Fig. 4.34: Example of a triangle template.
As explained above; if no correlations are defined among the relations, there will be some misinterpretations (Fig. 4.35), and three line segment passing through a point could be interpreted as a triangle.

![Fig. 4.35: Two interpreted instances of triangle template. a) with correlations. b) without correlations.](image)

Example template no. 2

The second template defines one kind of resistor representation found in schematic diagrams. It is supposed that the resistor is drawn as a rectangle (Fig. 33a). As it is not certain that the lines of the image are preprocessed properly, angles of the junctions have to be given some tolerance in their required matches.

```
// Type or name of the template
Resistor

// Defined six line segments with at least 9 pixels in length and no matter what type they are.
6 L 9 0 0 L 9 0 0 L 9 0 0 L 9 0 0 L 9 0 0 L 9 0 0
// "0 1 C 0 0" means L0 and L1 are connected in one endpoint.
// "0 1 A 85 95" says the angle between L0 and L1 is about 85° to 95°. "T" stands for Tjunction and "G" for "Greater" relation.
13 0 1 T 0 0 2 1 G 15 0 1 2 C 0 0 2 3 C 0 0
3 4 C 0 0 4 1 C 0 0 5 3 T 0 0
0 1 A 85 95 1 2 A 85 95 2 3 A 85 95
3 4 A 85 95 4 1 A 85 95 5 3 A 85 95
// "0 1 D" means the connection directions in relation 0 and 1 are not the same.
2 2 3 D 4 5 D
// There is two connectors, which are elements "0" and "5". "N" after "0" and "5" means the connectors neither are "input" nor "output".
2 0 N 5 N
```

Fig. 4.36: Example of a resistor template.
4.3 - System Overall Structure

The objective of this phase of the work was the development of a general system for the interpretation of line diagrams (Fig. 4.38) that could take various low-level preprocessing components that had been developed for the more limited special purpose interpretation system described in chapter 3. The implemented system offers a general approach for interpretation of line diagrams based on attributed relational structures (ARS) that describe components and their interrelations. The term templates is used for the attributed relational structures employed by this system. Each template, as defined before, represents a standard component as used in a particular problem domain.

In the first step the scanned image should be segmented. There are several algorithms reported in the literature for segmentation of documents (Wahl et al. [82], Nagy and Seth [84], Nagy et al. [85], Wang and Srihari [89], Baird et al. [92], Nagy et al. [92], Pavlidis and Zhou [92], O'Gorman [93]). The algorithm used here scans the image repeatedly checking for blank raster lines, in either horizontal or vertical directions. Any such blank line will separate segments. Segments separated by horizontal blank areas (=paragraphs or lines of text etc) are resegmented using a scan for blank vertical raster lines (this would separate columns of paragraphs, or, later, individual
characters). Whenever a segment is split into two parts, each part becomes a candidate for further analysis.

The actual process uses separate thresholds to control the segmentation height and width. The horizontal segmentation threshold "H" defines the minimum amount of the blank space (i.e. number of blank horizontal raster lines) between horizontally separated segments. The vertical threshold "W" controls the amount of blank space (blank vertical raster lines) between occupied areas that must exist for these to be split into separate segments.
The user is able to assign different values for the H and W parameters and so control the segmentation process. If "H" or "W" are assigned with a value more than 1; then there should be at least "W" blank columns or "H" blank rows between two parts of the image to be segmented, otherwise it is left as a connected segment. Figures 4.40 and 4.41 illustrate the segmentation of the document image shown in figure 4.39, with different values for "W" and "H".

Figure 4.40 shows the segmented image with "W=1" and "H=1", and Figure 4.41 illustrates the effects of the segmentation with "W=9" and "H=5", and as can be seen in this case, the paragraphs of the text are segmented from each other.
Fig. 4.40 Segmented document page with "W=1" and "H=1".

Fig. 4.41 Segmented document page with "W=9" and "H=5".
Using various heuristics, segments that represent structural diagrams are separated from the text regions. One major difference between the text and graphics parts of the image is the density of the black pixels. The graphics parts usually have a low density of pixels (<0.1). The diagram regions or segments are cleaned up with noise filters and other conventional image processing operations, such as contour tracing, thinning, etc. Then diagram is converted to a collection of lines, arcs and circles by their geometrical properties, which are called *primitives*, by the extraction and grouping of lines. These processing stages use the algorithms that were presented in the previous chapter.

In most domains that use line diagrams, there are specific groupings or sets of lines and arcs that are schematically meaningful; for example, a combination of parallel lines and complex arcs that represent a transformer. A major step in the processing involves the recognition of groups of lines that can be reliably matched with one of these domain specific groupings. These groupings are defined by *templates* that are originally specified by a domain expert and stored in a dictionary (Fig. 4.38).

Defined templates and extracted primitives from the image are delivered to the template matcher. This part of the system works in a manner analogous to a forward chaining expert system. The matcher’s "working memory" initialized with the primitives, and it will try repeatedly to match the parts of the diagram image (components, primitives), with a confidence level to templates, till no more matchings can be found. When templates are matched, newly found matches for template are added to the working memory (the new component replace the matched primitives, which are then removed from working memory). Subsequently, components or a component and a remaining primitive may be combined when matches are made with more elaborate templates.
4.4 - Matching

"Matching", here, is not an algorithm for matching bitmaps or any other colormap images as in the image processing area. "Matching" is the process of finding instances of a defined template among the primitives extracted from the image of the line diagram.

The template matcher works with dictionaries of templates as defined by domain experts. When the program starts, a user-selected dictionary is opened, and the system loads the descriptions of the templates that it is to seek.

Initially, "exact matching" is attempted. For a template to be "matched" exactly, all the components that it specifies must be found among the set of primitives found in the image and these selected primitives must satisfy all the constraints defined by the relations and correlations given in that template.

It is common for distortions in the original scanned image to be propagated through the sequence of preprocessing steps and eventually cause "incorrect" vectorization. For example, an arc in the original drawing may end up being represented by several straight line segments in the final vectorized image; or it is possible for a straight line segment to be vectorized as an arc. In most scanned images, a significant fraction of the components fail to be successfully identified by the exact matching approach. For example, as shown in Fig. 4.42, where an "AND" gate has not vectorized properly, consequently a template defined as a combination of three straight lines and an arc will fail to match.

![Fig. 4.42: Arc shape parts of an AND gate is vectorized as a set of straight line segments.](image)
Consequently, there is a second phase that employs "inexact matching". In this phase, the template matcher tries to find a proper combination of the primitives that could substitute for a primitive which is not exactly matched. For example, if all the elements are not matched for some template, the matcher tries to find an alternative element(s) to be replaced by the unmatched one. If the unmatched element is a line segment, the template matcher will try to find an alternative such as a circular arc that has attributes (location, length, etc) similar to those that would have been present in a matched line. If such an arc can be found, it can be substituted for the missing line element. As shown in Fig. 4.42 sequences of short straight line segments can sometimes be used to substitute for an unmatched arc.

The template matching step uses a hierarchical approach. Initially simple templates, such as triangles, rectangles, etc, are matched. Later, more complicated templates that are defined as combinations of primitives and other templates would be considered for matching.

There are no restrictions on the order of elements that make up a template, nor are there any restrictions on the order of the constraint relations that the domain expert may define. Consequently, there is no generally optimal order for matching templates.

The template matcher (Fig 4.43) simply starts matching with the first constraint defined in the template, and tries to find element(s) that match. All possible elements are checked to find out if they are accurate instances for this relation. Each element that is unsuitable is marked not to be checked again. In each successive stage of matching a template, the matcher selects the next unsatisfied constraint relation, and tries to find matches for its elements. Some of the later relation constraints will be satisfied by data elements already matched in earlier relations. In other cases the matching has to be
expanded to include additional primitives. The matching algorithm is shown in Fig. 4.43.

```plaintext
Match-Template(List of Templates LT, List of Constraints LC)
{
  for (each template "T" in LT) do
  {
    // Set the status of constraint as unmatched.
    for (each constraint "C" in LC)
      do C->_match = UNMATCHED;
    // Initialize the holding matched constraints array.
    for (each element of TMatch)
      do element = UNMATCHED;
    Match(T, LC, 0);
  }
}

Match(Template T, List of Constraints LC, int index)
{
  if (index is reached to the number of elements in T)
    then // all constraints are matched.
    {
      create a new constraint "C" by TMatch elements;
      if (!ExistIn(LC, C)) // it is a new case of template.
        then
          {
            append(C, LC);
            ShowMatchedTemplate(T, LC);
          }
        else; // it has found before, in other way.
    }
  else

Fig. 4.43: Matching Algorithm (cont.).
```
i = 1;
for (each Constraint C in the list of constraints) do
{
    // Must have the same type as defined.
    if (NOT RightType(C, T, index)) then continue the for loop;
    // Figure element must not have been reserved.
    if (C is CLAIMED) then continue the for loop;
    // Must not be matched yet with current template.
    if (C is MATCHED) then continue the for loop;
    // Must be in range of index'th T elem., if defined any.
    if (is NOT InRange(C, T, index)) then continue the for loop;
    // Check constraints
    checked = 1;//To indicate if the relation checked or not.
    for (each relation "re" of T) do
    {
        el = first element of re; e2 = second element of re;
        if (e1 == index) then {a = e1; b = e2; order = 1;}
        else
            if (e2 == index) then {a = e2; b = e1; order = 0;}
            else continue the for loop;
        if (other = TMatch[b] == UNMATCHED) then continue;
        // Current element should be in particular relation
        // to a previously matched element.
        LC->locate(other);
        Constraint *cnst2 = LC->_cur->item;
        if (order == 1)
            then checked = Matching(cnst , cnst2, re);
        else checked = Matching(cnst2, cnst , re);
        if (!checked) then break the for loop;
    }
    if (!checked) then continue;

Fig. 4.43: Matching Algorithm (cont.).
// Check if satisfied relations are correlated properly?
correlated = 1;
for (each correlation "c" of the template T) do
{ rl = first relation of c; r2 = second relation of c;
  if (one element of relations rl or r2 is not MATCHED)
    then continue for next correlation;
  c11 = constraint matched with first element of rl;
  c12 = constraint matched with second element of rl;
  c21 = constraint matched with first element of r2;
  c22 = constraint matched with second element of r2;
  // check if they are in right correlation.
  if (AreCorrelated(c11, c12, c21, c22, T))
    then continue for next correlation;
  else break;
}
if (!correlated) then continue;
TMatch[index] = i;
cnst->_match = index;
Match(T, LC, index + 1);
  TMatch[index] = UNMATCHED; cnst->_match = UNMATCHED;
}

Fig. 4.43: Matching Algorithm.

Quite often, a composite element that is to be matched will have some symmetry. In such cases, one can find multiple matchings. For example, suppose that the template matcher has to perceive the template defined by Fig. 4.44 (which represents an amplifier symbol like that in Fig. 4.45).

This template has a horizontal axis of symmetry and it is possible to match top/bottom elements in more than one way. As shown in Fig. 4.44, if the line segment "p1" is considered as the first primitive sought, then the primitives set \{p1, p2, p3, p4, p5, p6\} would constitute the matching of the sides. However, the set \{p1, p2, p3, p5, p4, p6\} is an equally good matching of this symbol. The template matcher would find the second matching when starting with the line segment p2. When an instance of a
template is found, it is checked against previous matches for that template and only inserted in the working memory if it involves a different set of primitive elements.

```
// Type or name of the template
Amplifier
// Defined six line segments with at least 9 pixels in length
// and no matter what type they are.
6 L 9 0 0 L 9 0 0 L 9 0 0 L 9 0 0 L 9 0 0 L 9 0 0
// "0 2 T 0 0" means p0 makes "Tjunc" with p2, and "1 2 A 85 95"
// forces the connection angle be almost right.
// "2 3 C 0 0" means p2 contacts p3.
8 0 2 T 0 0 1 2 T 0 0 2 3 C 0 0
2 4 C 0 0 5 4 C 0 0 5 3 C 0 0
0 2 A 85 95 1 2 A 85 95
// The relations should be such that the direction of the Tjuncs
// and the connection points of p3 and p4 with p5 be the same.
// The connection points of p2 with p3 and p4 must be different.
3 0 1 S 2 3 D 4 5 S
// There are 3 connectors, 2 as input ("I"), and 1 output ("O").
3 0 I 1 I 5 O
```

![Fig. 4.44 A defined template for amplifier.](image)

![Fig. 4.45: An amplifier that would be interpreted in different directions.](image)

Each time two relations of a template are completed (i.e. the right instances of their primitives are found in the image); any correlation between these two relations is
checked. If the required correlation constraint is not satisfied, then a backtrack search is done to find other possible cases of the relations and their elements.

Next, interconnections are found among instances of the templates. The templates defined by a domain expert can have some of their component lines (arcs etc) defined as "connecting lines". The Connection finder will try to find lines in the original image that join the "connecting lines" of individual matched templates. Proposed connections are checked further in those cases where a set of rules can be provided to validate interconnections. Some apparent connections may be rejected by such rules; e.g a proposed connection joining the inputs of two "OR" gates, or two batteries connected in a wrong way (Fig. 4.46) would each be rejected as a "false connection". If a false connection is found, the system will assume that there is a mismatching in the template matching process, and will try again to find the right components. In this way the system tries to find the best possible matching.

![Diagram](image)

**Fig. 4.46: Example of wrongly matched templates.**
5. Experimental results

5.1 Introduction

The application of templates and the template interpreter with different data including schematic circuit diagrams, chemical formulas, flowcharts, and cursive script writings (for example Arabic scripts) has been implemented and tested. Although all data presented is different in appearance and application, it could all be successfully interpreted.

Different data sets were tested on actual scanned images from original sources such as books and journal articles. All samples were scanned by a Macintosh "Onescanner" using the Macintosh Ofoto standard software package to acquire a bit map image which would then be converted to Xbm bitmap form on Unix.

For each application domain, a number of appropriate templates were defined. They identified the symbols or structures on the image of the diagram that would have to be matched. In section 5.2 the results of the interpretation system on some schematic circuit diagram samples are shown. Section 5.3 illustrates the interpretation of flowcharts. The system was also applied to chemical structure diagrams; original diagrams and the results of their interpretation are presented in section 5.4. Finally, the recognition of Arabic characters is examined, the results are illustrated in section 5.5.
5.2 Circuit Diagrams Interpretation

5.2.1 Symbols and Templates

The symbols illustrated in Fig. 5.1 form the basic repertoire for the interpretation of circuit diagrams. Template(s) were defined to describe each of these symbols and were combined in a "circuits library". Obviously, the set is not complete; other components may be found in circuit diagrams. However, this set is sufficient for many cases including all the circuits in this thesis's case studies. Additional symbols and templates could be defined if needed.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Shape</th>
<th>Symbol</th>
<th>Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>Earphone</td>
<td></td>
<td>Transformer</td>
<td></td>
</tr>
<tr>
<td>Capacitor</td>
<td></td>
<td>Morse key</td>
<td></td>
</tr>
<tr>
<td>Transistor</td>
<td></td>
<td>Battery</td>
<td></td>
</tr>
<tr>
<td>Resistor</td>
<td></td>
<td>OP-Amp</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Diode</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 5.1: Symbols considered for schematic diagrams interpretation.

Templates corresponding to these symbols are maintained in a small "database". As noted in previous chapters, image distortion before or after preprocessings is common. The adaptive matching schemes described in Chapter 4 attempt to deal with certain types of distortion (e.g. an arc where a linear segment was expected). But in order to accommodate other distortions (and to simplify handling of matching of symbols at different orientations), some symbols in the database have multiple templates.
For example, an appropriate template for a resistor symbol could be the template defined in Fig. 5.2. Template number one in the figure, considers a resistor drawn horizontally, template number two is for matching resistors that are drawn vertically, and finally template number three defines a resistor symbol in any direction.

// Template No. 1 Defining a horizontal resistor symbol.
RESISTOR

// Defined six line segments with at least 9 pixels in length.
// "H" means "Horizontal" and "V" means "Vertical" line.
6 H900 V900 H900 V900 H900 H900

// "T" stands for "Tjunction", "G" for "Greater", and "C" for
// connection relation.
7 01T00 21G150 12C00 23C00
   34C00 41C00 53T00

// "2 3 D" means that the connection direction in relations 2 and
// 3 are not the same.
2 23D 45D

// There are two connection elements, number 0 and 5. Number 0
// supposed as "input", and number 5 as "output".
2 0 I 5 0

// Template No. 2 Defining a vertical resistor symbol.
RESISTOR

6 V900 H900 V900 H900 V900 V900

7 01T00 21G150 12C00 23C00
   34C00 41C00 53T00

2 23D 45D
2 0 I 5 0

// Template No. 3 Defining a general template for resistors.
RESISTOR

// Defined six line segments no matter what type they are.
6 L900 L900 L900 L900 L900 L900

13 01T00 21G150 12C00 23C00
   34C00 41C00 53T00
   01A8595 12A8595 23A8595
   34A8595 41A8595 53A8595

2 23D 45D
2 0 I 5 0

Fig. 5.2: Example of a resistor templates.
5.2.2 Interpreted Example

Fig. 5.3 illustrates a typical circuit diagram from the data set used to test the program. The image contains a variety of different types of symbols including "Transistor", "Resistor" and so on.

The sample document shown in Fig. 5.3, is segmented in order to separate the text parts from the diagram part using the segmentation algorithm discussed in the previous chapter. The segmentation routine calculates some properties of each segment, such as the length and width of the minimum rectangle surrounding the segment and the ratio of black pixels to white pixels and so forth. Such simple heuristics work well and it is easy to segment the image and separate those parts that contain diagram(s); these parts are then selected by the system for further processing.

Next, text annotations are removed from the diagram. As previously discussed, the text separation algorithm, again using simple heuristics, finds and removes the characters. Then, the outline contour of the selected segment that contains the line diagram is detected and extracted. The extracted line diagram is then processed further for analysis and interpretation.
diagram is traced (the user is prompted to enter the control thresholds that are needed i.e. the parameter W as described in the definition of the algorithm). Fig. 5.4 shows the line diagram segment of the document with its text parts removed and the contour of the filled areas traced.

In the next step of the preprocessing the image is thinned by the thinning routine. The thinned image the sample is shown in Fig. 5.5.

Following the sequence of the black pixels and vectorizing these sequences, will result in a bunch of straight line segments, circular arcs, and probably digital circles. These primitives are entered into the "working memory" of the template matcher. The file containing templates is given as input to the program. Templates from the file are loaded and the program begins its matching process.

The template matcher repeatedly attempts to match primitives and adds instantiated templates to the "working memory", as it tries to find every possible instance of the templates. When all matchings for templates are complete, the matched primitives are
removed from the working memory. As was discussed before, the template matcher in each step selects a template from the database and tries to find an instance for it by matching each of its elements. Every structure identified is inserted into a list of found symbols. A template can define a number of connection points or lines and the system will then search for all lines that link these connection points to each other. Fig. 5.6 presents an illustration of the symbols and connections found in the example image.

![Line Diagram Interpreter](image1)

**Fig. 5.5:** Thinned image of the diagram segment.

![Interpreted Image & Connection Table](image2)

**Fig. 5.6:** Found symbols of the diagram with their links marked.
The line diagram interpretation system produces a table with the located symbols, giving details of their names, positions and the number of their connecting lines. This "connection table" also contains information identifying each of the connecting lines. The connection table of the interpreted line diagram example is shown in Fig. 5.7. This tabular data is in a form that could be used as input to some CAD database systems.

<table>
<thead>
<tr>
<th>Object</th>
<th>Location</th>
<th>Connection Lines</th>
<th>List of Connection Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>No</td>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>Capacitor</td>
<td>1</td>
<td>27</td>
<td>100</td>
</tr>
<tr>
<td>Capacitor</td>
<td>2</td>
<td>85</td>
<td>100</td>
</tr>
<tr>
<td>Capacitor</td>
<td>3</td>
<td>141</td>
<td>100</td>
</tr>
<tr>
<td>Resistor</td>
<td>1</td>
<td>56</td>
<td>152</td>
</tr>
<tr>
<td>Resistor</td>
<td>2</td>
<td>214</td>
<td>102</td>
</tr>
<tr>
<td>Resistor</td>
<td>3</td>
<td>391</td>
<td>102</td>
</tr>
<tr>
<td>Resistor</td>
<td>4</td>
<td>115</td>
<td>151</td>
</tr>
<tr>
<td>Transistor</td>
<td>1</td>
<td>309</td>
<td>99</td>
</tr>
</tbody>
</table>

Fig. 5.7: Thinned image of the diagram segment.

The next two figures show the results of the interpretation system as applied to the diagram part of the manual page shown in Fig. 4.39. The text was removed and all preprocessing was carried out on the diagram. Fig. 5.8 illustrates the redrawn diagram with the located symbols, and the links between these symbols were marked by different numbers. Fig. 5.9 is the table containing found symbols, their location, connecting line numbers, and a list of the connection points which defines what symbols are connected to each other. Figures 5.10 — 5.14 illustrate further examples of circuit analysis.
Fig. 5.8: Interpreted diagram part of the Fig. 4.39.

Fig. 5.9: Connection table of interpreted diagram (Fig. 5.8).
NOTE: In all the circuit diagrams, wires that cross without being joined are shown like this:

Fig. 5.10: Another circuit diagram to be interpreted.

Fig. 5.11: Annotated text is removed from Fig. 5.10.
Fig. 5.12: Thinned image of circuit diagram of Fig. 5.11.

Fig. 5.13: Rebuilt circuit.
### 5.2.3 Problems

For good quality image data (that only involve symbols from the database), the performance is satisfactory; though in most cases one or two symbols (i.e. 5-10%) are not recognized. A limited amount of human post processing appears unavoidable. As image quality deteriorates, performance falls off. For example Fig. 5.15 illustrates the preprocessed image of a document, where all symbols except the transistor and one of the resistors are found by the system.

![Fig. 5.15: A circuit diagram that contains 8 symbols.](image)
Fig. 5.16: Incomplete interpreted diagram of Fig. 5.15 (6 out of 8 symbols).

Fig. 5.17 is a less satisfactory example, because a part of the battery symbol has almost vanished during the preprocessings and the diode symbol is vectorized as a little circle by the vectorizer, the recognition rate is only 60% or less.

Fig. 5.17: More incomplete interpreted circuit diagram (3 out of 5 symbols).
5.3 Flowchart Interpretation

5.3.1 Symbols and Templates

The symbols that are normally used in flowcharts are shown in Fig. 5.18. There are defined templates for each of these symbols. These templates are stored in a database to be used by the system whenever a flowchart is included in a document. Some of these templates are shown in Fig 5.19 and Fig 5.20. The "cross page connection symbol" that is common in flowcharts was not included in the symbol set; any such symbols in a flowchart being processed are treated simply as connections.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Shape</th>
<th>Symbol</th>
<th>Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td></td>
<td>Subroutine</td>
<td></td>
</tr>
<tr>
<td>I/O Box</td>
<td></td>
<td>Process</td>
<td></td>
</tr>
<tr>
<td>Decision Box</td>
<td></td>
<td>Stop</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 5.18: Flowchart Symbols.

Template number 1 defines a flowchart "start box" symbol. It is composed of five elements: two circular arcs, two horizontal lines and a vertical line segment serving as the connecting line. The expected limits of the start and end angles of the arcs are supposed to be from 90 to 270 degrees for the left side arc and from 270 to 90 degrees for the right side. Two horizontal line segments are supposed to be almost equal in length and are connected to the arcs. The vertical connection line segment has a "T junction" relation with one of the horizontal lines and is defined as lying under it. Two arcs components of the start symbols should be defined in such a way to avoid misinterpretation with another symbol, as illustrated in the Fig. 5.20.
The second template (Fig. 5.21) defines a "decision box". A "decision box" comprises of seven elements, four diagonal and almost equal line segments, and three other line segments (two vertical and one horizontal in this case) that play the connecting line role for the template. A decision box may be drawn with two connecting lines horizontal and the third vertical. It isn't necessary to define this arrangement with a second template. The template used for the decision box does not specify directions for the connection lines but works instead in terms of restrictions on angles, for example from 100° to 160°. This approach also makes the matching process less sensitive to distortion.
// Template No. 2 Defining a decision box.
// Type or name of the template

DECISION

// There is defined 7 elements. Four diagonals as the body of symbol, and the others as connecting lines.
7 D 9 0 0 D 9 0 0 D 9 0 0 V 9 0 0 V 9 0 0 H 9 0 0

// Diagonals are connected to each other successively.
10 0 1 C 0 0 1 2 C 0 0 2 3 C 0 0 3 0 C 0 0
4 0 C 0 0 4 1 C 0 0 5 2 C 0 0 5 3 C 0 0
6 1 C 0 0 6 2 C 0 0

// Connection points of the first four elements are different.
// and the connection points of each connecting line are the same.
5 0 1 D 2 3 D 4 5 S 6 7 S 8 9 S

// Connecting lines are number 4 (input), 5 and 6 (output).
3 4 I 5 0 6 0

// Template No. 3 defines a general template for decision symbol.

DECISION

// 7 elements are defined. Four diagonals as the body of symbol, and the others as connection lines.
7 D 9 0 0 D 9 0 0 D 9 0 0 L 9 0 0 L 9 0 0 L 9 0 0

// Diagonals are connected to each other successively and make a "diamond". The last three lines are connected to the corners of the diamond.
16 0 1 C 0 0 1 2 C 0 0 2 3 C 0 0 3 0 C 0 0
4 0 C 0 0 4 1 C 0 0 5 2 C 0 0 5 3 C 0 0
6 1 C 0 0 6 2 C 0 0
4 0 A 100 170 4 1 A 100 170 5 2 A 100 170 5 3 A 100 170
6 1 A 100 170 6 2 A 100 170

// Connection points of the first four elements are different,
// and the connection points of each connecting line are the same.
5 0 1 D 2 3 D 4 5 S 6 7 S 8 9 S

// Connecting lines are number 4 (input), 5, and 6 (output).
3 4 I 5 0 6 0

Fig. 5.21: Some templates defined for "decision box".

5.3.2 Interpreted Example

An example of one of the flowcharts, successfully interpreted by the system, is shown in Fig. 5.22. The standard preprocessing steps are carried out on the image. The
cleaned up and thinned image is shown in Fig. 5.23. The recognised elements and their connecting lines are shown in Fig. 5.24, finally a summary table of found symbols and the links between them is illustrated in Fig. 5.25.

Fig. 5.22: Sample document containing a flowchart diagram.
Fig. 5.23: A cleaned up, and thinned image of the diagram.
Fig. 5.24: Found symbols from the flowchart image and their links.
5.3.3 Problems

Symbols that are "subsets" of other symbols present problems for interpretation systems. Such problems were mainly encountered with the flowchart examples (but could occur more widely). For example, if the template of the "processing box" is to be matched and there is a subroutine symbol on the image, this symbol will be matched as a "processing box".

The simplest way to avoid problems of conflicting matches is to arrange the templates in the database so that the matching of the "superset" template is attempted before the matching of the "subset" template. Thus, a symbol for the "subroutine" should be matched first. Later matchings of the "process box" template will not attempt to match those elements that have been (reliably) interpreted as parts of a subroutine symbol.
5.4. Chemical Structure Diagram Interpretation

5.4.1 Symbols and Templates

Chemical structure diagrams follow a number of conventions. The greater part of a structure will be built up with carbon atoms with a few nitrogens, oxygens, or "functional groups" attached to the skeleton. A chemical structural diagram usually appears as a sequence of lines (bonds) with some (named) atoms.

Chemists rarely name all the atoms — carbons (and any attached hydrogen atoms) are often left implicit. A join point of two lines has an implicit -CH2-, a join point of three lines has an implicit -CH< etc. Only the "heteroatoms", functional groups and some peripheral -CH3 groups amongst others are explicitly named. The names appear as character strings, possibly with subscripts and characters in different sizes. However, there are only a small set of functional groups. It is possible (at least in principle) to define elaborate templates for each of the possible character strings.

Apart from the basic skeletal structure made up of bonds, and the character strings for the functional groups, a structure diagram may also contain some "stereochemical information". This information is a code used by chemists to indicate the three dimensional structure of the molecule as shown in the two dimensional diagram. There are a variety of "stereochemical tags" that can appear in structure diagrams; the most common are triangular shaped wedges — either solid or made up out of many short lines.

The template approach has been adapted to handle chemical diagrams. This was a prototype study and no attempt was made to include all the parts that would be needed to handle complex structures. Some of the symbols used are shown in Fig 5.26 and some corresponding template definitions are given in Fig 5.27.
Template No. 1 defines a (pi-bonded) "Carbon" symbol. Note that the character "C" does not appear in the image. There are three main straight line segments that are connected to each other in one point and the fourth line segment is close and parallel to one of them. This symbol would match a part of a structural diagram that represents one end of a double bond system e.g. a carbon in an olefin (>C=C<) or a ketone (O=C<).

Template no. 2 (Fig. 5.27) is for the perception of the (pi-bonded) "CH" parts of the molecules. The "CH's" are shown in the chemical formulas by three line segments with two of them being connected at one of their endpoints. The third one is very close and parallel to one of them and almost equal in length. This symbol would appear in diagrams of structures with aldehydes (O=CH-), other olefins (>C=CH- and -CH=CH-) and would also match parts of the aromatic systems. Each corner of the hexagon representing "C₆H₆" could be matched by this template.

Template no. 3 (Fig. 5.27) defines the symbol "C₆". This corresponds to the "aromatic benzene ring" which is a common component of many structures. Sometimes these
components are drawn showing an alternating sequence of double and single bonds. Sometimes they are drawn with hexagonal frames containing circles (but both represent the same thing). The form with alternating double and single bonds may be matched by repeated matchings of templates 1 and 2. As mentioned before, the combination of six carbons are usually shown by a hexagon that has three line segments parallel to its sides, or a circle in place of these three line segments. The template specifies that there may be as many as six substituents on such a system.

// Template No. 1. Defining carbon symbol.
CARBON
// Defined four line segments.
// "L" means any sort of straight line segment.
4 L 0 0 0 L 0 0 0 L 0 0 0 L 0 0 0
// First three lines are connected in one point. Fourth one is parallel ("P"), close ("H"), and almost equal ("E") to one of them. The closeness is defined at most two tenth of its length.
5 0 1 C 0 0 0 2 C 0 0 0 0 3 P 0 0 0 0 3 H 0 2 0 3 E 0 0
// Connection point of the first three elements is the same.
// "S" means the connection point of L0 and L1, is identical to the connection point of L0 and L2.
1 0 1 S
// In this template all elements are connection lines..
4 0 N 1 N 2 N 3 N

// Template No. 2. Defining "CH" symbol.
CH
// Defined three line segments.
// "L" means any sort of straight line segment.
3 L 0 0 0 L 0 0 0 L 0 0 0
// First two lines are connected in one point. Third line is parallel("P"), and close("H") to one of them.
// The closeness is defined at most one tenth of its length.
3 0 1 C 0 0 0 2 P 0 0 0 0 2 H 0 1
// In this template all elements are connection lines..
3 0 N 1 N 2 N

// Template No. 3. Defining a "C6" structure.
C6
// There are defined 2 elements, one "hexagon", and one "circle".
2 H 0 0 0 C 0 0 0
// The circle is included ("I") in the hexagon.
1 1 0 I 0 0
// There should be six connecting elements, but as it would differ, they are not defined ("-1" indicates this matter).
6 -1

Fig. 5.27: Template examples defined for chemical structures interpretation.
5.4.2 Interpreted Example

A sample document page including chemical structural diagrams is shown in Fig. 5.28.

Utilizing the information thus obtained, several derivatives of I were synthesized and subsequently tested. The following information is added to a ChemBase database dedicated to the project: structures, melting points, and LD₅₀ and CE values (lethal dosage and carrageenin-induced paw edema primary testing information). ChemBase automatically calculates the formula and molecular weight when the structure is drawn.

Data for the first set of compounds, consisting of twenty-five 2-aryl-derivatives, are shown in Figure 2. This ChemBase spreadsheet format shows compounds and associated data in the order they were entered. By viewing multiple entries, comparison of information is simple. ChemBase also provides the ability to sort a list of compounds in ascending or descending order by data fields. This feature is illustrated in Figure 3 in which the compounds are organized by descending order to rank them according to activity, allowing investigation of structure-activity correlations.

Inspection of the structures for the entries in the table shows that antiinflammatory activity is increased by strong electron-converting groups in the para position of the aryl substituent, while electron-withdrawing substituents show much less activity (Scheme II).

![Chemical Structures](image:Chem100.xbm)

Fig. 5.28: Sample document page containing chemical structures.
Firstly, the image of the document is segmented and one of the segments that contains a chemical structure diagram is selected (Fig. 5.29) and preprocessed (Fig. 5.30).

![Chemical structure diagram](image1)

**Fig. 5.29:** A chemical structure diagram selected from Fig. 5.28.

![Preprocessed chemical diagram](image2)

**Fig. 5.30:** Preprocessed sample chemical diagram.
In the next interpretation step, a database of chemical templates is loaded by the interpreter and matching elements are marked on the position where they are found and the chemical structure is rebuilt (Fig. 5.31).

Atoms and their bonds are listed in a table (Fig. 5.32).

Fig. 5.31: Rebuilt chemical structure.

Fig. 5.32: Connection table of found atoms and their bonds.
5.4.3 Problems

The most serious problem with the interpretation of chemical structures is character recognition. The rate of character recognition via the templates is not as high as in a conventional OCR system, so it is possible to omit some parts of the formula. This weakness in the recognition rate occurs due to the distortions caused by the different steps of preprocessing and by the inherent difficulty of defining proper templates. To overcome this problem, the best solution might be to use a conventional OCR system for the recognition of the textual parts of the chemical formula. This necessitates breaking the segment containing chemical structure into two or more parts, some with text and one with just the lines (bonds) and images such as stereochemical symbols. As discussed previously, the system is able to carry out this function and separate the text component for OCR.

As in the case of the flowcharts, the problem of template inclusion (that is one template being a superset or subset of another) arises. For example template no. 2, as defined in Fig. 5.27, is a subset of template no. 1. One possible way to prevent any misinterpretation caused by this problem, is to arrange the templates in the database in such a way that any template which is a subset of another, appears after its superset.

5.5. Cursive Script Writing Interpretation

5.5.1 Symbols and Templates

Arabic/Persian writing has been selected as an example of OCR to demonstrate the applicability of the defined templates. Arabic, like other languages, is written in different fonts, such as "Naskh", "Kufi", "Nastaligh" and so forth. The most popular font for printing books and papers is "Naskh". The Arabic character set has 28 characters and 10 digits. If 4 extra characters, that are defined in the Persian character
set, are added to these characters, the interpretation system would be able to recognize Persian script.

The most important feature of Arabic/Persian writing is that the characters are not written separately, as are Latin based and other scripting systems. Consequently, the characters must first be separated, then recognized. Using templates, it is not too hard to find the individual characters in the text and recognize them.

Each character in the Arabic/Persian character set, has different cases (corresponding to the upper and lower cases in the Latin character set). Some of them have just one case, some have two cases and others, four. Work on the computerized display and printing of Arabic/Persian writing has helped to identify standards for the cases of each character. One of these standards defines four cases for every character. The first case of each character is to be employed when the character is used as the first letter of a word (i.e. the rightmost one!). The second character case is used when the character is neither the first nor the last letter of a word. The third case is used when a character is the last character of a word and finally, the fourth case is defined when the character is written in isolation. The cases for some of the characters are shown in Fig. 5.33.

There are several groups of characters that differ solely in respect to some dot-like accent marks. For example, the "Be" group includes four characters named: "Be", "Pe", "Te", and "Se". "Pe" is similar to the "Be" but has three dots under it instead one. "Te" has two dots over it, and "Se" also has three dots over it. The "Jim" group forms another example. This group has four members called: "Jim", "Che", "Ha", and "Khe". Apart from the accent dots, these are identical in shape. "Che" has three dots instead of one dot, "Ha" has no dots, and "Khe" has one dot over it.
Some characters in the Arabic/Persian character set, as shown in Fig. 5.33, have just two different cases. One of these is the "Alef" or "A" character. For this character, the first and fourth cases are the same, and the second case is the same as the third. Others are the "Daal" group, "Re" group and "Vaav". These characters have fewer cases because they never make contact with characters written after them (that means to the left of them).

Fig. 5.34 shows some of the templates defined for recognition of the Arabic/Persian characters. Template no. 1 is defined for the "Alef" character in its second and third cases. There should be two short line segments above the baseline of the text. These two line segments, the first of which is horizontal and the second vertical, are connected at one of their endpoints. There is a further restriction, in that the first line segment should be at the right hand side of the second.
Template no. 2 is defined for the recognition of the "Be" character in its second case. There should be four elements. First is a horizontal line that connects the character "Be" to the previous one (i.e. on its right hand). Second is a short vertical line segment connected to the left end of the first and to the right end of the fourth element, and which should be located above the baseline. The third element is a very short line segment, below the baseline, close to the second element. The fourth, final, element is a horizontal line segment connected to the vertical one.

// Template No. 1 defines the character "Alef" in second case.  
// Type or name of the template

ALEF

// Defined two lines, one horizontal "H" and one vertical "V".  
// Both lines are located over the baseline of the text ("1").  
2 H 3 0 1 V 3 0 1

// First line is located on the right hand side ("R") of next one,  
// and connects it. The vertical line is at least two times of  
// the other one ("G").  
3 0 1 C 0 0 0 1 R 0 0 1 0 G 2 0 3 0

// No correlations.  
0

// Element one is the connecting line.  
1 0 I

// Template No. 2 defines the character "Be" in second case.  
// Type or name of the template

BE

// Defined four lines. Two horizontal "H", one vertical "V",  
// and a very short line segment. Lines 1, 2, and 4 are located  
// over the baseline (1), and the third one is under it (2).  
4 H 3 0 1 V 3 0 1 L 1 3 2 H 3 0 1

// First line is located on the right hand side ("R") of next one,  
// The vertical line is at most equal to the first one.  
7 0 1 C 0 0 0 1 R 0 0 1 0 G 5 1 0  
1 3 C 0 0 1 3 R 0 0 1 3 G 5 1 0 2 3 H 0 0

// Connection points on the relations 0 and 3 are one point.  
1 0 3 S

// First and the fourth Elements are connection lines.  
2 0 I 3 0

Fig. 5.34: Some templates for Arabic/Persian characters recognition.
5.5.2 Interpreted Example

Fig. 5.35 shows a sample document of Arabic/Persian text. This image is preprocessed and thinned by the system and the resultant image is shown in Fig. 5.36. Using a small database of defined templates for Arabic/Persian character recognition, several characters are recognized and marked on the image (Fig. 5.36 to Fig. 5.38). On Fig. 5.36 all cases of the character "Alef" are recognized and marked. Note that an extra character ("Laam") has also been recognized. This would be removed later when the cases of the character "Laam" are interpreted.

Fig. 5.35: Sample document of Arabic/Persian text.

Fig. 5.36: Preprocessed image of the Fig. 5.35, and all "Alef" characters marked.
5.5.3 Problems

The rate of character recognition is only about 60% for characters such as those shown in the preceding figures. The inclusion of more complex characters, such as "He", would lead to a further lowering of the recognition rate. Inadequate resolution of the scanned image is one factor leading to these low rates. It occurs because features such as accents are frequently reduced to single pixels which are then removed as noise.
The preprocessing algorithms, described in chapter 3, are not attuned to the handling of cursive characters. The filling of voids, closing of gaps and thinning steps can remove features that distinguish characters.

These limitations are what one would expect for a system which was originally designed to handle technical diagrams. These tests with characters served more as a test of the completeness of the scheme for defining complex features using templates than for character recognition.
6. Conclusion

In this thesis a general approach for the interpretation of different types of line diagrams, such as circuit diagrams, flowcharts, chemical structure diagrams has been developed. A system of Primitives, Relations, and Correlations has also been developed. They present definitions of expected forms for individual parts of a line diagram — these definitions take the form of "templates" that can be matched with the line diagram. A graph matcher mechanism has been implemented that can perceive all possible instances of each "template".

The goals achieved in this thesis are, in brief:

- A number of preprocessing algorithms (Morphological Filters, Thinning, Vectorization and so on) have been revised and implemented.
- Some novel algorithms (for example, Contour Tracing) have been designed and implemented.
- The Primitive, Relation, Correlation system has been developed for the definition of any component of line diagrams. As part of this development a minimum set of relations has been formulated to cover most cases in the different line diagrams under consideration. The image interpretation
system includes implementations of algorithms for checking each of the different relations.

- Algorithms for matching defined "templates" with the image have been implemented.
- Based on the "templates" idea, a system for interpretation of line diagrams has been implemented and tested with several different type of image data.

6.1 Future Work

The most difficult part of these experiments proved to be the definition of the templates. This is a very time consuming and complicated task. This task could be made simpler with the use of an interactive graphics editor program. Rather than designing a template for any particular part of the line diagram, one could use the graphics editor to select the various primitives from a menu or palette and these can then be combined with relations and correlations taken from other menus. The graphics editor program would translate the designed symbols to the structures used by the interpretation system. The development of such an editor should be relatively simple.

A substantially more ambitious method of creating the templates would be to start with scanned images of ideal prototypical components. The system would have to identify all possible primitives and relations. The domain expert could then select those primitives and relations that were significant. These could be used to define matching structures. In this way, the system could be trained using real data.

Defined templates should be arranged so that the "superset" templates appear in the template database before any "subset" template. The ordering of templates is currently
carried out by hand, but it might be better to leave this to the graphical editor system or another system designed for template definition for automatic reordering.

Matching the defined templates with the line diagram image is carried out in two stages. In the first stage, matching is "exact" — all elements must be found, all relations (and correlations) must be satisfied. The second stage tries to complete the interpretation process for those parts of an image that could only be partially matched. In this second stage, the matcher is permitted to perform substitutions. For example it can substitute lines by arcs or vice versa. However, these two steps sometimes fail to find all possible instances of the templates. The development of an "inexact" matching mechanism is a consideration best left for the future. If the "training" approach is attempted for the template definition task, the need for "inexact" matching might be reduced, because all the samples used to define the templates would have had to pass the same preprocessing steps as the image.

Attempts have been made to match textual annotations embedded in the line diagrams. This text matching has been carried out using defined templates. For example, the chemical structure interpretation system has templates for the character strings for common heteroatoms and substituent groups. The character recognition rate achievable with these templates is usually not more than 80%. It should be possible to achieve higher recognition rates by isolating the parts of the image containing text strings and passing these to a conventional OCR system.

The template matching algorithm searches its complete working memory for possible primitives as it extends each partial matching. For example, if it has one line primitive selected and needs a touching arc then it considers all arcs in the working memory. If the diagram is large, this results in numerous checks against remote primitives. The matching process should be further refined to group spatially related primitives. One
possibility might be to use a "quadtree" or "octtree" type of data structure to encode the position of primitives in the image. Then, when matching a template, consideration could be more easily restricted to plausible neighbouring primitives. This extension would dramatically improve performance for some applications such as the recognition of characters in a page of text.
BIBLIOGRAPHY


[Freeman 70] H. Freeman, "Boundary encoding and processing", in Picture Processing and Psychopictorics, B. S. Lipkin and A. Rosenfeld (Eds.), New York, Academic, pp. 241-266, 1970.


