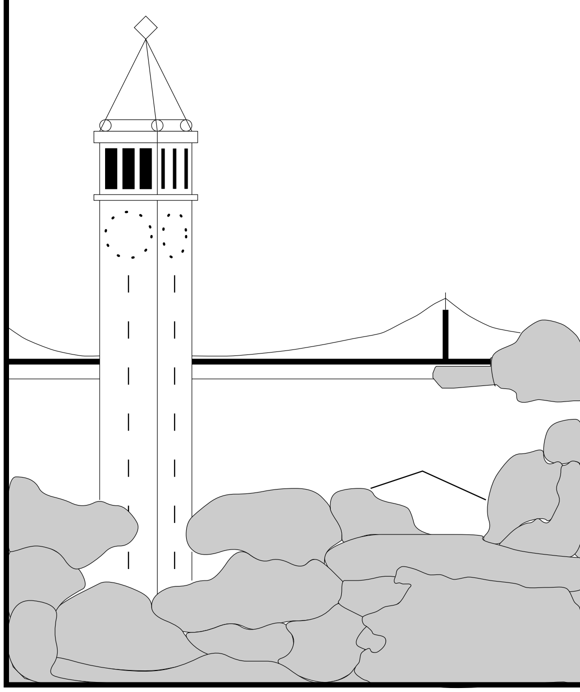
Turbo Recognition: An Approach to Decoding Page Layout

Taku Andrew Tokuyasu



Report No. UCB/CSD-2-1172

January 2002

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Turbo Recognition: An Approach to Decoding Page Layout

by

Taku Andrew Tokuyasu

B.A. (Pomona College) 1984
Ph.D. (Princeton University) 1990
M.S. (University of California, Berkeley) 1995

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION
of the
UNIVERSITY of CALIFORNIA, BERKELEY

Committee in charge:

Professor Richard A. Fateman, Chair
Professor Jitendra Malik
Professor Ray Larson

Fall 2001

This research was funded as part of the NSF/NASA/DARPA Digital Library Initiative, under National Science Foundation grant number CA 98-17353

Turbo Recognition: An Approach to Decoding Page Layout

Copyright 2001
by
Taku Andrew Tokuyasu

Abstract

Turbo Recognition: An Approach to Decoding Page Layout

by

Taku Andrew Tokuyasu
Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Richard A. Fateman, Chair

Turbo recognition (TR) is an approach to layout analysis of scanned document images inspired by turbo decoding from communication theory. The TR algorithm is based on a generative model of image production in which two regular grammars simultaneously describe structure in horizontal and vertical directions. The TR model thus embodies non-local constraints while retaining many of the features of local statistical methods. This grammatical basis allows TR to be quickly retargeted to new domains. While TR, like turbo decoding, is not guaranteed to recover the statistically optimal solution, we present experimental evidence of its ability to produce near-optimal results for a non-trivial synthetic problem. We explore the expressiveness of TR for describing abstract structure in two dimensions, and develop a hierarchy of grammars of increasing complexity. We demonstrate the application of the TR framework to the analysis of simple text documents. We discuss how TR can be applied to the analysis of composite documents and images corrupted with extreme amounts of noise, and show how it can be applied to problems such as the layout analysis of journal article title pages.

Professor Richard A. Fateman
Dissertation Committee Chair

To my sister, Setsu

Contents

Acknowledgements

It is a great pleasure to thank my advisor, Richard Fateman, for guiding me through my graduate career at U. C. Berkeley. His suggestions constantly opened up new perspectives that I, narrowly focussed on my research, failed to consider. He has always been generous with his time and wisdom, and especially so after I began the task of writing this thesis. I am also grateful to him for showing by example that it is possible to maintain some semblance of sanity in what can be an intensely hype-driven discipline.

I am deeply indebted to Phil Chou for generously sharing his idea that a new development in communication theory called turbo codes could form the basis of an approach to layout analysis. I had been aware for some time of the importance of Document Image Decoding (DID) for optical character recognition, but the analysis within DID of even two-column documents presented significant difficulties, which I had been struggling to remedy with stochastic context-free grammars. Phil's idea brilliantly bridged the gap between the complexity of two dimensions and the efficiency of regular grammars. I am grateful to him for patiently clarifying numerous points about DID.

I thank the other members of my committee (in addition to Richard Fateman), Jitendra Malik and Ray Larson, for reading my thesis and providing feedback. Robert Wilensky generously provided me with financial support for many years, and helped place this work in a more general context, where scanned document images are just another electronic document type, albeit a particularly inert one without further analysis.

Michael Jordan's lucid Statistics 242 course influenced my adoption of a graphical model approach to TR. I thank him, Yair Weiss and Andrew Ng for several discussions on graphical models and turbo codes. I thank Henry Baird, Kris Popat, Tom Breuel, Dan Bloomberg, and Dan Greene at Xerox PARC for conversations and moral support. A document image analysis course taught by Henry Baird and Richard Fateman helped clarify my thinking on the topic. I thank Henry Baird and Dan Bloomberg at Xerox PARC, and Jon Hull at Ricoh California Research Center, for hosting me during summer internships.

My erstwhile officemates, who have included ByungHoon Kang, Tony Morosco, Hao Chen, Tracy Riggs, and Shendong Zhao, have been consistently helpful and of good humor. Jeff Anderson-Lee, Ginger Ogle, Joyce Gross, Howard Foster, Tom Phelps, Loretta Willis, and other members of the Digital Library Project at Berkeley have given me the benefit of their varied experiences, and provided important support on software and hardware issues. Rich Martin, Drew Roselli, Sara McMains, Phil Liao, Brent Chun, Chad Yoshikawa, Melody Ivory, Tina Wong, Bernd Pfrommer, Richie Vuduc, Nikunj Oza, S. P. Rahul, and many others helped me survive and enjoy the computer science graduate experience. I thank International House Berkeley for providing a stimulating residential environment, while freeing me from having to wash dishes and supplying a fast T-1 network connection.

This thesis would not have happened were it not for Gary Kopec. He generously showed me how document image analysis could be placed within a framework of rigor, elegance, power and depth, and thereby influenced my entire way of thinking about research, as well as the course of my graduate career. I am very grateful that I was able to work with him through the auspices of the Berkeley Digital Library Project. I can only hope that this thesis may help contribute to the further application of DID-based ideas to document image analysis.

Preface

Turbo recognition (TR) is an approach to layout analysis of scanned document images inspired by turbo decoding from communication theory. The TR algorithm is based on a generative model of image production in which two finite-state grammars simultaneously describe structure in horizontal and vertical directions. The TR model thus is able to embody non-local constraints while retaining many of the features of local statistical methods.

TR can be viewed as theoretical and practical progress in the the Document Image Decoding (DID) approach to document image analysis (DIA). Previous work on DID layout analysis [1] was based on stochastic context-free grammars (SCFGs). TR's basis in regular grammars makes it considerably more efficient than SCFGs on the subclass of layouts to which it is applicable.

The key contributions of this thesis include:

- The development of a decoding basis for layout analysis that is 1) statistical, 2) nearly optimal, 3) efficient, 4) expressive in two dimensions, 5) quickly retargetable.
- Experiments which demonstrate the near-optimal behavior of the TR algorithm.
- The exploration of the space of layout structures that can be described using TR grammars.
- The exploration of the use of TR for both physical segmentation and logical labeling on scanned document images.
- The development of a freely available implementation to demonstrate and further explore these ideas in practice.

In the following chapters, we introduce our problem domain, introduce the foundations of TR, and discuss its application. In more detail, this thesis is organized as follows.

- Chapter 1: Introduction to page layout analysis (PLA). For readers who may be unfamiliar with this topic, we describe PLA in some detail, placing it in the context of the overall DIA task. We review previous work on PLA and other work related to TR.
- Chapters 2 through 5: Foundations of the TR model. We begin by describing the basics of DID [2], a general framework for document image analysis which forms the background to our work. We describe the communication view of TR, which

frames the problem of page layout analysis as a decoding task. We discuss how the underlying technologies for TR, finite-state transducers and graphical models, are applied to form a statistical model of two-dimensional image generation, and consider the TR decoding process from several points of view. We then describe some details of our current methodology, including our Java implementation of the TR decoding algorithm.

- Chapters 6 through 10: Further exploration of TR. We turn to an investigation of the optimality of the decoding results produced by TR. We explore the space of structures that TR can describe, developing a hierarchy of grammars of increasing complexity. We apply this to the analysis of simple text documents. We demonstrate the ability of TR to handle levels of noise beyond the reach of alternative methods, and apply it to the analysis of composite documents. We illustrate the application of TR to logical layout analysis. We end with a summary and conclusions, and provide some suggestions for future work.

- Appendices A and B: Finite state machines and graphical models. We provide an overview of some background material, to introduce our notation and for general reference.

We note here some aspects of the system used to perform this research and write this thesis.

- Software: emacs, Sun JDK 1.2 and 1.3, IrfanView, Cygwin, MikTeX/AucTeX, xfig, Adobe PhotoDeluxe, F-Secure SSH, Hummingbird Exceed.

- Hardware: Dell Dimension 4100 pc (900 MHz, 256 MB RAM, Windows 2000), Fujitsu C-6320 notebook pc (333 MHz, 64MB RAM, Windows 98), Epson 1240U scanner.

I am particularly indebted to the Open Source and freeware developers whose freely available software made this work possible.

Chapter 1

Introduction to Page Layout Analysis

Research in document image analysis (DIA) is concerned with the development of techniques which enable computers to "read." This has traditionally been viewed as a subset of the general effort to enable computers to "see." The idea of digitally processing images in general was given strong impetus in the 1960s by the need to analyze images produced by the US space program. The range of techniques that were viable at that time was strongly constrained by limited computing resources. Since then, the character of computing has changed considerably, with powerful processors and large memories readily available. The scope of DIA has also broadened considerably. While the standard input type for DIA remains the single sheet document page image from a flatbed scanner, input material now can come from a variety of devices, such as hand scanners (e.g. the HP Capshare [3]), pen scanners (e.g. the C-Pen [4]), copiers [5], video cameras [6], fax machines, and book scanners [7]. The source documents take many different forms, such as business letters, journal and magazine articles, books, bank checks, postal mail pieces, forms, and tables. Even electronic documents such as email [8, 9] have become the subject of DIA research. The document content can be machine printed or handwritten, and in addition to standard text (which itself has a rich characterization), it can include more exotic content such as mathematics, graphics, and logos. DIA has evolved from its early digital image processing days and can now count a variety of influences and fellow disciplines, including speech recognition, computer vision (the analysis of images of natural scenes), video, image compression, communication theory, and linguistics. From a broader perspective, DIA is concerned with the entire spectrum of information[1] from low level details of the document instance to high level semantic concerns within the purview of natural language processing [11].

DIA has long been considered one of the simplest artificial vision tasks, no doubt because text documents are human artifacts constructed specifically for the purpose of

[1]The set of tasks in this research area can be broadly characterized as *document analysis, document classification, and document understanding* (see for example Malerba et al. [10]). These are close equivalents of what we will call physical layout analysis, page classification, and logical layout analysis in the context of page images.

communication. In practice, DIA has proven to be much harder than anticipated[2]. This can be seen to have several causes: 1) noisy images; 2) a huge number of layouts, fonts, etc.; 3) a large variety of conceivable applications, each with different performance metrics, 4) tradeoffs between efficiency, generality, availability and use of prior knowledge, and the quality of retrieved (extracted) information. It is unlikely that a single general paradigm will "solve" DIA. Rather, different methods can be used, depending on the application and the available resources.

A burgeoning area where DIA techniques can be of considerable value is digital libraries [12, 13]. These often contain large collections of scanned paper images, which require processing if they are to be more than just static images, or even retrieved in the first place. In the Berkeley Digital Library Project [14], scanned images constitute one possible layer in a novel document type called Multivalent Documents (MVD) [15]. DIA can "enliven" static document images, to enable actions such as cutting and pasting, reformatting, and searching. High quality DIA results can be critically important for applications such as information retrieval and data mining. The desire to mesh with such applications also help define what the targets of DIA research should be.

A goal espoused since the advent of electronic publishing is the seamless integration of paper and electronic documents. This requires that printing (from electronic to paper media) and recognition (from paper to electronic form) be both fast and precise, so that the two media are essentially interchangeable. This dream has yet to be realized in general, but having the dream itself may help shape research and even the form that documents take in the future.

1.1 Overview

Document image analysis[3] has traditionally been divided into two separate tasks, optical character recognition (OCR) and page layout analysis (PLA) [16]. In brief, OCR is concerned with transcribing isolated glyphs or text lines into their corresponding symbolic form, and PLA is concerned with analyzing the overall structure of the page image without necessarily knowing what the characters are on the page. PLA is typically considered to be a preprocessor for OCR, to find where the text zones are. Nevertheless, one can envision situations where OCR and PLA are done independently, jointly, one before the other in either order, applied iteratively, etc.

For example, OCR can be done largely in the absence of layout analysis when the document is a simple succession of text lines. Similarly, PLA can be performed to categorize documents without doing OCR. The availability of fast, general-purpose OCR engines (e.g. Textbridge [17]) suggests workflows where OCR is done first (where some aspects of layout analysis are done implicitly by the OCR engine), and then application-specific layout analysis is performed on the output [18].

[2]This is partially a matter of definition. There are many circumstances, such as recognition of clean single-font text, where present systems perform quite well. Barcode reading is a related task where control over the source domain enables outstanding performance.

[3]Here we consider documents to be composed largely of machine-printed text. Such documents can include non-textual content, such as halftones, and graphics, and come in various types, such as articles, letters, tables, forms, etc.

OCR is the more mature field of the two. It is sometimes considered to be a solved problem for all practical purposes. Much of this maturity can be ascribed to its characterization as a statistical classification task. A recent book [19] illustrates how current systems are nevertheless still brittle, sometimes surprisingly so. Innovative research with applications to OCR (e.g. Belongie et al. [20]) continues to appear. We also note that OCR is strongly dependent upon the performance of other tasks such as PLA.

PLA remains a less well-developed area than OCR. From a recognition (as opposed to authoring) standpoint, page layout is apparent only from the cumulative effect of many lower level primitives, such as characters or foreground pixels. Simple combinatorics suggests that page layout can exhibit great variety. It is also at the page layout level that DIA truly reflects the two-dimensional nature (and hence complexity) of the underlying page image.

PLA itself is often separated into parts, which we will refer to generically as physical (structural, geometric) and logical (functional, syntactic) layout analysis[4]. Physical layout analysis is concerned with analyzing the structure of a given page image, construed as a two-dimensional array of pixels, without recourse to meta-information such as the type of document the image represents. Its aims can include estimating page level properties such as global noise or skew, finding zones within the page such as halftone images, text columns, text lines, and individual words, or estimating local properties such as the texture class of individual pixels.

Logical layout analysis applies prior knowledge about a document to construe the logical function of various parts in the page image. Some typical tasks include labeling text zones in a journal article title page as the "title," "author," "abstract," or "body text," identifying the cells of a table according to data type, and identifying the address block on an envelope given that it is a US mail piece. Read order determination is another significant logical layout analysis task. The distinction between physical and logical analysis is again not hard and fast. For example, prior knowledge about logical structure can constrain the way physical segmentation is performed.

As an example of the layout analysis task, consider the page image in Figure 1.1. Despite the fact that we cannot decipher what is written on the page[5], we still can have much to say about its contents. One thing we notice from the outset is that the image is *two*-dimensional. This is an obvious, yet often inadequately dealt with, feature of the image. The arrangement of black (foreground) pixels on the page is not random, but shows clear structure. Pixels that we identify as belonging to text and belonging to a halftone image are segregated into different regions of the page[6]. The text itself is highly organized into parallel linear structures (text lines), reflecting the requirements of human legibility and centuries of typographic convention. These in turn are organized into two columns. The

[4]Other names for related tasks include page segmentation and page classification.

[5]This statement itself is worth pondering further. Approaches which have demonstrated some OCR capability even on documents that are illegible include word shapes [21, 22, 23] and DID [2, 6]. It is probably true, though, that this particular example (Figure 1.1)is at too low a resolution for any technique to have much success

[6]A strict classification of this sort is not always readily made. Text can appear within graphics such as maps or diagrams, and text itself can take on the character of graphics, as in highly decorated capital letters in illuminated manuscripts.

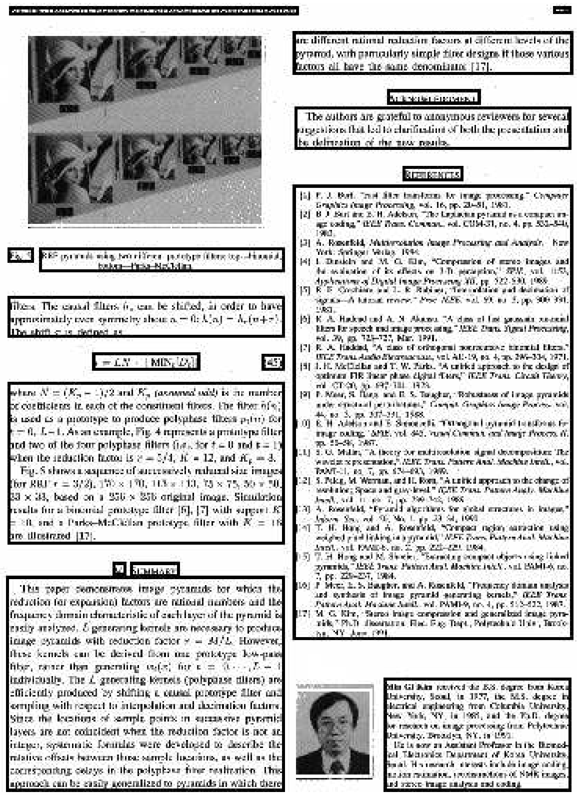


Figure 1.1: Example of the page layout analysis task.

borders of the image are straight and orthogonal. Additional regular features, such as text line spacing, word spacing, font size, etc., can also be discerned (and estimated). These are features of the physical layout of the page. The logical layout of the page is comprised of regions with functional roles such as the bibliography, figure caption, page number, photo of the author, etc. An intermediate step might be to *classify* the document according to type, such as business letter, article from a specific journal, etc. Given the class of the document, it might then be possible to assign logical meaning to physical elements that have been identified at an earlier stage. Alternatively, the classification itself might be sufficient for some kinds of retrieval tasks.

1.2 Technical review

Here we review selected previous work in page layout analysis. The literature is vast, and we focus on a few typical works. This selection is necessarily incomplete, for which we apologize in advance. Reviews and article collections which cover some of the topics that we touch upon include O'Gorman and Kasturi [16], Cattoni et al. [24], Tang et al. [25], Bunke and Wang [26], and Baird et al. [27]. The current state of the art is represented in a series of workshops on "document layout analysis and its interpretation" (DLIA) [28, 29], with proceedings available online.

Desiderata for a robust page layout analysis system include the ability to handle image skew, noise, arbitrary layouts, mixed content (text, graphics, halftones), and grayscale or color images. Most systems in practice focus on a few of these dimensions while ignoring the others. TR, for example, focuses on dealing with noise and Manhattan layouts, and can potentially handle an arbitrary number of content types[7].

1.2.1 Run-length smoothing

Early work on page segmentation was based on relatively straightforward processing of the foreground pixels in the image, a stream of research that continues to this day. A driving factor in the design of such methods is the desire for efficiency in both space and time. A prominent example is the method of run-length smoothing, which can be applied to binary[8] images. Each row (or column) of pixels is first represented as a list of intervals ("runs") of black pixels, and then neighboring runs are merged together ("smoothed") if the space between them is less than some pre-specified threshold. This allows the physical segmentation of a page image into regions (such as text columns) by a judicious choice of the threshold values. Examples include Wahl et al. [30] and Fletcher and Kasturi [31]. Such methods work well on clean images with well-defined distance scales which separate the different types of content. However, due to their critical dependence on fixed thresholds, they are generally brittle against noise.

1.2.2 Docstrum

O'Gorman [32] defines a "docstrum" representation to summarize the spatial relationship between connected components and their k-nearest neighbors. This allows the extraction of important layout parameters such as skew angle and the spacing between words and between text lines. This information can then be applied to text line and text block segmentation. The dependence on connected components implies a rather high sensitivity to noise, since noise can merge or split components, and add new ones. The general philosophy of accumulating local evidence to support making global decisions is echoed in the TR formalism, though in a rather different guise.

1.2.3 Whitespace

Baird et al. [33] concentrate on analyzing the structure of the white space background between regions, instead of recognizing the foreground regions per se. By sorting maximal white rectangles by size, this approach implements a "global-to-local" strategy, which utilizes as much information as possible before making segmentation decisions. Other work includes Pavlidis and Zhou [34], Antonacopoulos and Ritchings [35, 36], and Kise and Yanagida [37]. Baird [38] includes a review of work as of 1994.

[7]Each content type would correspond to a different input symbol (see later chapters). At present, TR can process text and halftones. In general, a preprocessing step would probably be required which yields initial probabilities of each content type at each pixel.

[8]Other terms for such images where the pixels are either black or white include bitonal and bilevel.

1.2.4 Area Voronoi diagram

Kise et al. [39] propose a segmentation method based on the area Voronoi diagram, a generalization of the usual "point" Voronoi diagram to extended objects such as connected components. Judicious pruning of the derived edges segments the page image in a manner reminiscent of background analysis (see above), and can be applied to skewed page images and non-Manhattan layouts. This method has shown state-of-the-art performance in a recent comparison of layout analysis methods by Mao and Kanungo [40]. One weakness is again the dependence on connected components, which makes it vulnerable to noise.

1.2.5 XY cuts

XY trees were proposed by Nagy and Seth [41] as a data structure for describing a large class of layouts, such as those used for technical journal articles. They proposed creating the tree by recursively segmenting the page image based on alternating horizontal and vertical projection profiles on extracted subregions. The 2D analysis problem is thus reduced to a series of 1D ones. This structure was exploited in later work [42], in which ordinary 1D string grammars control the analysis process, thus allowing the integration of physical segmentation with logical labeling.

The XY cut approach, with its emphasis on analysis of grammatical structure in orthogonal directions, is similar in spirit to TR. It is however not explicitly based on a stochastic imaging model. The segmentation decision is based on thresholding of projection profiles, which can be brittle. By eliminating information in one dimension, projection profiles wash out local information which could be useful in making segmentation decisions (as we shall see). TR also can analyze some Manhattan layouts which cannot be segmented using XY cuts (see Figure 1.2 for an example). On the other hand, the XY cut procedure can proceed to an arbitrary depth of recursion, while TR is limited to at most a fixed depth of recursion.

1.2.6 Textures

Humans can easily recognize and distinguish text regions from other types of page content such as halftones, graphics, and the background, even from far away or while squinting. This suggests that these various content types have differing characteristic *textures*. Taking advantage of such texture differences for general image segmentation tasks is a standard topic in computer vision, with a voluminous literature. As an aside, the concept of a text texture is reinforced by the possibility of *synthesizing* such textures, as proposed by Efros and Leung [43].

In the context of DIA, three texture classes are typically targeted: text, halftones, and graphics. A sampling of work in this area includes Dunn et al. [44]), who use Gabor filters, Jain and Zhong [45], who train neural network masks, and Suen et al. [46], who define a modified fractal signature feature.

A large number of related techniques from the image processing and computer vision communities have been applied to layout analysis. For example, wavelets and related transform techniques are quite popular (some references are given in the next section). Mor-

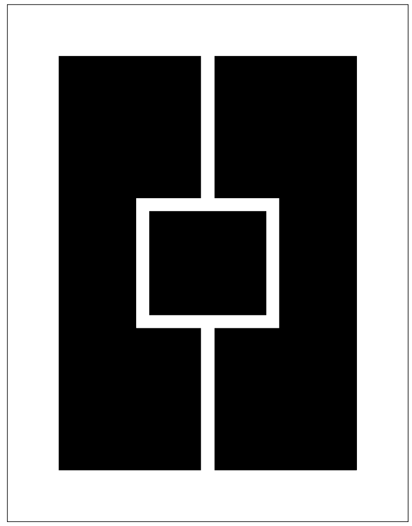


Figure 1.2: Example of a layout which cannot be decomposed using XY cuts.

phological transform methods have in recent years become standard [47] and are possibly used in some commercial OCR engines [17].

1.2.7 Statistical model-based approaches

One popular texture segmentation approach uses Markov Random Fields (MRFs). This is a generalization of a Markov chain to a two-dimensional grid [48]. Each node on the grid represents a pixel (or possibly a block of pixels) in the image. In a one-dimensional Markov chain, the probability of a node having a given value is dependent only on the value of its predecessor in the chain. Similarly, in an MRF, the value of a pixel depends only on the values of its neighbors. This neighborhood is defined by the connectivity of the grid. In order to make probabilistic inference tractable, the 1D "predecessor" relationship is often (rather arbitrarily) generalized in 2D to "above and to the left," to form what is known as a *causal* MRF. MRF methods are in common use for image segmentation and classification [49, 50]. Li et al. [51] apply a form of causal MRF known as a 2-D HMM to document image segmentation.

The probabilistic dependence of each node in an MRF on the state of its neighbors is a simple attempt to incorporate local context. The need to incorporate more context beyond a fixed-size neighborhood has given rise to various "multi-scale" approaches [52, 53]. Li et al. [54] has further references to the general computer vision literature on this topic, which has close ties to wavelet analysis.

A rather different approach to introducing statistical models into DIA has been developed by Haralick and coworkers [55]. Given the knowledge (or assumption) that document layout can be described hierarchically as a tree-like structure, which starts with the entire page image at the root and descends through substructures such as columns and text lines down to words and characters, they develop a probabilistic approach designed to fit the observed data to this expected structure. The observed data consists of the sizes and the locations of connected components (bounding boxes) in the page image, and structures such as text lines and text blocks are constructed bottom up. An iterative relaxation-like method is then used to explore the space of partitions of the glyphs and their functional labels, with the goal of maximizing the probability of a set of measurements, given the partition. This method can perform simultaneous physical and logical layout analysis and, amongst other things, process images with moderate amounts of skew.

We also mention here the work on pseudo-2d HMMs [23], which, while not addressed to layout analysis per se, has some resemblance to TR, in being a Markov model-based approach to analyzing structure in two dimensions. A pseudo-2d HMM has a factorized structure, in which a series of HMMs describing horizontal pixel patterns are stacked together vertically, roughly speaking (the reverse situation, where a series of vertical HMMs are connected together horizontally, is also a possibility). This forms an elastic shape matcher, which is useful for identifying keywords in noisy text, for example. A pseudo-2d HMM cannot capture non-local correlations, however, such as the fact that the top and bottom row of a rectangle have the same length, which is something that is easily embodied in a TR model.

We note in passing the paper by Freeman et al.[56] as an example of recent research that treats image analysis problems in terms of graphical models (see Appendix B), in this case as an undirected (two-dimensional) Markov network. We will find it useful to employ related graphical model techniques in deriving the TR decoding equations.

1.2.8 Encoding, acquiring knowledge, groundtruthing

An important issue in logical layout analysis is how to incorporate document knowledge. The ability to tag zones with metadata such as title, author, abstract, sender, addressee, etc., clearly requires some form of domain-specific knowledge. Languages and data formats for encoding such knowledge include ODA [57] and SGML [58]. A number of formats, such as DAFS [59] and XDOC [60], are aimed specifically at DIA tasks. The possibility of interactively developing layout models in terms of stylesheets encoded in XML [61] has been explored by Spitz [62]. A system which utilizes a database of decision trees and learned rules for document analysis, understanding, and classification is described by Esposito et al. [63]. Lee and Kanungo [64] have developed a flexible document layout data format encoded in XML to support the visualization and editing groundtruth data. A related toolkit for performance evaluation is described by Mao and Kanungo [40]. Commercial products and formats such as Adobe Capture [65] and DjVu [66] allow the incorporation of OCR data (and possibly rudimentary zone information) in documents meant for distribution on the World Wide Web.

1.2.9 Manual correction

Modern consumer-oriented commercial OCR systems provide a facility for performing manual zoning, which can be used to augment or substitute for automatic methods. This is often sufficient for casual users who need to process just a few pages. In the comprehensive document analysis system proposed in the early 80's by Wahl et al. [67], manual editing of scanned documents was an integral part of the design. In large service bureau contexts devoted to the processing of forms, checks, postal mail pieces, etc., OCR is often just one part of an extended workflow involving many human operators [68]. Here it is again possible to rely upon human feedback, although this may simply involve processing the pieces that the OCR system rejects. Recent research has generally avoided incorporating the user explicitly into the system. Interactive feedback has been used for training and retargeting of model-based recognition systems, such as those of Spitz [69] and Shamilian et al. [70]. In general, research systems have a wide variety of different expectations of their users. The extent to which DIA systems can rely on the user expertise and make the feedback process easier are largely unsettled issues within the DIA community.

1.2.10 Applications of layout analysis

Here we mention some applications of layout analysis, both physical and logical. A sampling includes extraction of metadata from journal article title pages (Kim et al [18]), document classification by genre [71], labeling of forms and office documents [72], finding zones in bank checks (Kornai and Connell [73]), postal address block location (Jain and Chen [74]), table detection and understanding (Hu et al [71], Baird et al [70]), newspaper segmentation and understanding [75, 76, 77], and document image compression [78, 79, 80, 81, 6].

The development of systems which can automatically convert printed documents into marked up electronic documents has generated much interest in the context of digital libraries [82, 83]. A system which uses document structure analysis to support logical queries directly on document images stored in digital libraries is described by Niyogi and Srihari [84]. The distinction between paper and electronic documents is being blurred by efforts to "enliven" static page images with functionality such as cut and paste, search, clickable hyperlinks, annotations, etc. [15, 85]

1.2.11 DID

Document image decoding is unique as a framework in that it spans the entire range of DIA. It is unusual in that it confounds many of the above method classifications. This is because it takes a different approach than most other methods. We discuss it in more detail in the next chapter.

TR is an approach within the general DID framework, which is statistical, structural/syntactic, local (it inspects local evidence directly, as opposed to, e.g., taking projection profiles) and global (it seeks a globally optimal estimate). It can be top-down (in the sense of requiring high-level constraints on, e.g., the number of columns), and bottom-up (both in terms of the decoding process, which as mentioned previously proceeds from the pixel level, and in the sense that less restrictive TR grammars can be designed to segment word boxes, for example). At present, TR is primarily a physical and logical layout analysis technique. In principle, it can be integrated with OCR to make a single system optimized from top to bottom (without hard decisions made in between). It naturally transcends many of the previous categorizations of layout analysis techniques, as will be elucidated in the following chapters.

Chapter 2

Document Image Decoding

Document Image Decoding (DID) is a general framework for document image analysis proposed by Kopec and Chou in 1994 [2]. DID forms the basis for the work in the following chapters, and here we introduce its technical and historical background for reference purposes and to describe the context in which TR arose.

2.1 The DID framework

DID is based on the theory of communication, and we briefly mention some elements this theory here in order to clarify this connection. Modern communication theory is founded on the work of Shannon [86], who showed that the problem of digital communication over a noisy channel can be divided, without sacrificing optimality, into two parts, source coding and channel coding [87]. Source coding deals with encoding an analog or digital signal into a finite-length string of symbols. The symbol alphabet is often taken to be binary (without loss of generality). Data compression, for example, is naturally framed within this aspect of communication theory. Channel coding is concerned with transmission of a sequence of bits over a noisy channel. Typically some sort of redundancy is added, in order to make the transmission process more robust against noise. This is the subject of error-control coding, which comes in two main flavors, block coding and convolutional coding. DID, especially in its Markov model formulation, is most closely related to convolutional coding, and inherits from it a dynamic programming decoding approach for the purpose of recognition.[1] DID is also closely related to hidden Markov model approaches to speech recognition [88, 89].

The communication theory view of DID is shown in Figure 2.1. This is intended as a general framework for modeling the generation, transmission, and subsequent recognition of document images. The first three modules in Figure 2.1 constitute a stochastic model of image generation. The source selects an *input* message U from a set of candidates according to some prior probability distribution. Under the control of this message, the encoder formats two-dimensional *output* image X. This image is degraded upon transmission through the channel, resulting in an *observed* image Y. The objective of the decoder is to

[1] In typical document applications, there is no formal error control coding step per se, of course. Product bar codes are an interesting exception.

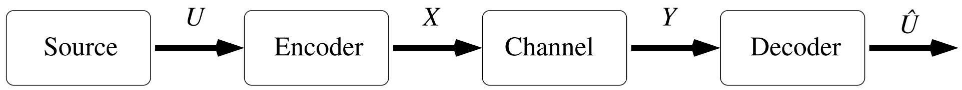


Figure 2.1: Communication theory view of DID.

recover the message U, given the observed data Y and prior knowledge (i.e., parameters) embodied in the generative model. The minimum probability of error[2] is achieved when the decoder returns the maximum a posteriori (MAP) message $\hat{U} = \arg\max_U P(U|Y)$. In this manner, document recognition is rephrased as the optimal decoding of a signal that has been generated and transmitted in a particular way.

This framework is meant to abstract the essential features of some real world process. For example, the input message U, the output image X, and the observed image Y could represent a TeX file, the resulting dvi file, and the tiff file produced after printing and scanning, respectively.

More generally, the input message U can be viewed as an ascii string, which could include embedded logical tags (as in XML). The pixels in the output image X assume values from a binary alphabet [3] $\{0', 1'\}$ in the most straightforward case, but larger alphabets are also common [91]. These output image symbols in general label different pixel *types*, which could designate any number of attributes, such as color, texture class, or susceptibility to noise. In most work on DID to date, the observed image Y has been binary, although research has recently begun on treating grayscale images [6].

The source and the encoder are jointly described by a Markov source, or equivalently, a stochastic finite state machine (FSM) with labeled arcs[4]. The FSM arcs are attributed with a transition probability, an ascii transcription (for the input message) a character template (for the output image), and a two-dimensional displacement (denoting where the next character should be placed). The stochastic evolution of the machine induces a probability distribution over the space of machine histories (of a given length), or alternatively, paths in a trellis (see e.g. Rabiner and Juang [88] or Manning and Schütze [92]). This in turn induces a joint probability distribution over input messages and output images via the labels. The channel in its simplest (and most commonly used) form is a stochastic mapping from output symbols to observed symbols, on a per-pixel basis (bit-flip noise).

The overall imaging process can be visualized as a cursor (or "turtle" [93]) which moves from point to point on the page, starting from the top-left corner of the page and ending in the bottom-right corner, meanwhile laying down templates on the page. The resulting output image is then passed through the channel to yield the actual observed image. We refer to the original paper for details [2].

The decoder uses a form of the Viterbi algorithm (dynamic programming) to find the MAP estimate $\hat{U}$. The general form of this algorithm is briefly described in Appendix A. We again refer to the original paper [2] for the specific algorithm used by DID. Elaborations

[2]This can also be phrased as "minimum risk under a 0-1 loss function." For details, see Duda and Hart [90].

[3]We use primes on symbols to signify that they are from the output image pixel alphabet.

[4]A brief overview of finite state machines is given in Appendix A.

of the generative model and optimizations of the decoding algorithm have been developed by Kam and Kopec [94], Kopec and Lomelin [95], Minka et al. [96], and Popat et al. [97], amongst others.

This framework plays several important roles. First, it organizes DIA processes in a modular fashion, enabling researchers to concentrate their modeling efforts on each part separately. By incorporating an explicit (stochastic) model of the document production process in the decoder, it becomes possible to define the notion of an *optimal* decoder. The basis in communication theory implies that we can immediately take advantage of the decades of research in this area. It also means that we can often apply innovations in communication theory to DIA with little modification. The framework is general enough that essentially any DIA task of interest can be placed within it. Much work remains to be done, however, to supply methods that can fully exploit the power of this framework. This thesis can viewed as further exploration of this idea.

It is worth pointing out that, despite the emphasis in DID on carefully crafted models of the imaging process, we are never concerned with actually generating images in this manner. We are only concerned with the inverse process, i.e. recognition. In addition, experience with DID (and related experience from speech recognition) shows that, while the connection of such generative models to reality is often tenuous at best, recognition performance is nevertheless robust against many details of their specification.

There are several aspects of the DIA task that make DID more than just a straightforward application of communication theory. Chief among these is the fact that the observed signal that the decoder receives occupies *two* dimensions instead of one. This potentially changes the character of the problem considerably from traditional one-dimensional decoding problems such as speech recognition (where the time dimension provides a natural linear ordering to the data). In general, it has proven to be difficult to make the jump from one to two dimensions, both in terms of forming models that are expressive enough, and in developing decoding algorithms that are efficient enough to be used in practice. We briefly discuss current DID approaches to layout below.

Some of the most thorough demonstrations of the effectiveness of the DID framework were a set of recognition experiments [82, 98] done in conjunction with the Berkeley Digital Library Project [14]. DID processing of one document in particular, a compilation of California state water district acts [99], exhibited a reduction in OCR error rate by a factor of ten, relative to a commercial OCR engine. These examples also demonstrate the ability of DID to perform logical analysis, which adds considerably to the value of the electronic version of the document beyond that which is provided by the ascii OCR stream alone.

A number of objections have been raised regarding the use of DID for DIA applications. Chief amongst these has been the criticism that DID is too slow to be of practical use. There are a number of aspects to this issue. One is that DID research has until relatively recently been focussed on simply establishing how to base DIA on a communication theory framework and examining some of the benefits that thereby accrue, not raw speed per se. In terms of computational complexity, DID text line decoding is linear in the number of image pixels, and hence in the theoretical sense, DID is quite efficient. In addition, this criticism is now largely invalid, at least in the realm of OCR. Optimizations have closed the gap between DID and commercial systems to roughly a factor of five or less [96], depending on the size of the character template library.

Another criticism leveled against DID is that it requires too much prior knowledge. This issue partially reflects differences in opinion on how a DIA system should be designed. The emphasis in commercial systems has largely been on developing general purpose engines which can recognize arbitrary document images. In practice, this is an extremely difficult task, and constraints of some kind, e.g. as to language, levels of noise, document type, and desired output have to be added in order to maintain a reasonable level of performance. When faced with a specialized task (e.g. retargetable high accuracy table reading [70]), unusual formatting (e.g. mathematical expressions [100, 101, 102, 103]), or new low-level primitives (e.g. printed music [104]), general purpose OCR engines can fail completely, with little recourse for adapting their performance to the data. DID advocates the development of custom recognizers tailored to the document and task at hand. It is possible, for example, that DID recognizers will be eventually generated by tools akin to *lex* and *yacc* [105]. The hope is that DID decoders can be specified and retargeted efficiently by users with relatively little knowledge of DIA and DID technology. This mode of operation is particularly well suited to batch processing of large sets of similar (and possibly unusual) documents which require conversion to high-value electronic form. In what follows, we try to be explicit about the prior knowledge that is required in order for our programs to work, and leave it to the reader to decide whether it is overly burdensome.

2.2 DID models of layout

DID has traditionally been focussed on performing OCR, as broadly construed. In pursuit of this task, DID has developed a variety of models of page layout which form an important background for the development of TR. We review this work briefly here.

As described in the previous section, DID source models of text describe the placement of character templates within the confines of a page. Each of these templates has a finite region of support where the pixels are nonzero (foreground). At present, the only hard constraint on the form of these models is the requirement that separate templates have disjoint support (i.e. have no overlapping foreground pixels) when placed on the page. This *template disjointness constraint* is perhaps easily misconstrued, so we elaborate on it a bit further. The key point is that the constraint only applies to the *generative* model, not to the glyphs that appear in a given observed image. A related point is that template pixel *overlap* is distinct from pixel *connectedness* in an observed image (it is not clear that pixel overlap makes any sense in the observed image). Indeed, images in which glyphs are connected to each other, either by design (as in cursive scripts) or due to noise, are some of the most interesting targets for decoding by DID methods. Similarly, bounding boxes of adjacent characters are allowed to overlap. The disjointness constraint means only that in the decoding *output* of DID, foreground pixels from neighboring glyphs will not overwrite each other. Theoretically, this constraint stems from the straightforward requirement that the observed evidence (in the form of foreground pixels) not be double-counted in evaluating likelihoods [2, 95].

2.2.1 Text lines

DID source models can be classified according to the layout complexity of the template placement in the model. Probably the simplest DID model is that for a single text line, which consists of putting down character templates one after another along a baseline, in accordance with a Postscript-like sidebearing model of character positioning [2]. DID currently does not incorporate a model of baseline skew, and it is customary to rely on external pre-processing to remove large amounts of skew prior to decoding[5]. A vertical jitter of ± 1 or more pixels around the baseline is often allowed in scoring templates during text line decoding.

2.2.2 Text block

The obvious generalization, a text block consisting of several text lines, can be described as a vertical sequence of text lines, each of which is a horizontal sequence of characters. Text block decoding consists for the most part of repetitions of the text line decoding algorithm, with the addition of a "carriage return" at the end of each horizontal pass to prevent the same pixels from being decoded twice. A search over all possible paths through the model is in principle still required to ensure optimality, which means every row of the image must be tested as a possible text baseline. A variation which allows the decoding process to be sped up considerably without loss of optimality is described in Kam and Kopec [107]. This takes advantage of the factorization into horizontal and vertical image structure to form *separable* models of text blocks. Together with the introduction of *admissible* heuristics, this allows the formulation of the *iterated complete path* (ICP) algorithm [108], which reduces the amount of processing required by applying text line decoding only along rows where there is likely to be a text line, in a manner that rigorously maintains optimality. These DID text block ("text column") models represent one approach to dealing with two dimensions in a "$1 + 1$" dimensional fashion.

2.2.3 Other text-like structures

DID models of a single text block, as developed by Kam and Kopec [94], set a standard for handling text layout within DID. Text images with more complex layouts, such as multiple columns, have as a practical matter been analyzed by first segmenting them into blocks by other means, prior to text block decoding (for instance, see the yellow pages example discussed in Kopec and Chou [2]). While DID clearly has the ability to analyze multiple column text directly by constructing more elaborate models, this approach quickly becomes cumbersome as the complexity of the layout increases. We note, though, that even within the context of a simple text column, sophisticated DID models can be developed in order to perform logical analysis, as shown again by the yellow pages example.

The advanced structured documents [98] included as part of the Berkeley Digital Library Project [14] demonstrate some of the sophisticated analysis possibilities inherent in the DID approach. As described by Kopec [106], these results were obtained using a

[5]Some of the best OCR performance exhibited thus far by DID resulted from the use of a third order polynomial fit to the base "line" [106].

combination of hand-crafted routines for layout analysis and automatically generated line decoders for text transcription. This was intended as a provisional methodology, while awaiting the development of DID technology that could efficiently decode common text layouts. This division of labor between scripts and line decoders had distinct advantages. It allowed Kopec to concentrate on the design of line decoders, which due to their simplicity is a relatively easy task. The use of scripts, meanwhile, allowed the addition of functionality such as cross referencing (hyperlinking), which would be extremely cumbersome to analyze within DID. Overall, these examples reinforce the idea that very high quality OCR is key to enabling advanced functionality from scanned document images.

Kopec was not clearly not satisfied with this state of affairs, however. He died in 1998 before being able to follow up on these ideas. This dissertation can be considered an outgrowth of his desire to expand the repertoire of DID-based methods for decoding page layout. Before turning to this work, however, we first briefly describe a different basis for DID which in fact generalizes all of the DID framework described so far.

2.2.4 Stochastic context-free grammars

The DID models presented up to this point have been based on stochastic finite state machines (i.e., a form of hidden Markov model). As mentioned earlier, image production can be viewed as resulting from the action of a cursor moving along a single path, down and across the page. The associated decoding algorithms retain much of the character of one-dimensional methods. While this allows for a surprisingly broad treatment of two-dimensions (e.g. printed music) [109], such an approach in general is quite cumbersome for describing the spatial relationship of *regions* in two dimensions.

Chou and Kopec [1], building on earlier work by Chou [100], established a more general framework that is based on stochastic context-free (instead of regular) grammars (SCFGs). This constitutes the most general formulation of DID at present. The source and encoder are now jointly described by a stochastic context-free attribute grammar. This generates a parse tree at random (which in the case of regular grammars means that a stochastic finite state machine generates a path at random). The tree is deterministically "walked" or annotated to produce a message string, and then is deterministically walked again to produce an encoding. Each nonterminal symbol can be thought of as denoting a subimage, which upon translation and rescaling is placed on the page. The terminal symbols at the leaves are character glyphs with the usual font metric information as attributes. Each rule of the grammar is attributed with functions describing the coordinate transformation between symbols on the left and right sides of the rule. Noise can in fact be incorporated within this generative process. The imaging model is completed by the addition of a channel, which can be a bit-flip noise as before.

Due to the coordinate functions which adorn each grammatical rule, this framework is quite general, essentially equivalent to many document formatters [1]. This SCFG formulation is likely to be sufficiently powerful to handle any layout analysis task of interest within the context of DID.

Decoding algorithms within this framework are derived from methods for parsing SCFGs, which in turn are closely related to standard methods for parsing CFGs. Background material on regular and context-free grammars can be found in Hopcroft and Ullman

[110], Grune and Jacobs [111], and Révész [112], and the stochastic versions are described by Charniak [113], and Manning and Schütze [92]. In brief, chart-like parsing methods which have a dynamic programming basis are typically used. The use of probabilities in addition provides a natural means of dealing with possible ambiguity of the parse.

The extra power offered by this SCFG basis for DID has been demonstrated chiefly in the realm of recognition of mathematical expressions [100, 114]. The main drawback of these methods is their computational complexity, which grows as $N^{5/2}$, where N is the number of image pixels[6]. Their utility for general page layout analysis tasks has thus been limited. The possibility of using heuristics as in the work by Kam and Kopec [94] has yet to be explored. One of our key contributions in this thesis is to provide a flexible layout analysis method within the DID framework that nevertheless has linear time complexity.

[6]The exponent varies with the number of attributes, so $N^{5/2}$ is in fact a lower bound.

Chapter 3

Introduction to Turbo Recognition

Turbo recognition (TR) is a communication theory based approach to page layout analysis in the tradition of Document Image Decoding (DID), which was described in the preceding chapter. As in the most common implementations of DID, TR is based on stochastic finite state machines (Markov sources). TR differs from this prior work in focussing on page layout rather than character recognition. As described below, this entails a slight modification of the DID communication channel framework, and the character of the decoding algorithm changes from being a single pass algorithm to an iterative one. Among its prominent features, TR is

- Efficient (linear complexity)
- Nearly optimal
- Rapidly retargetable

The linear time complexity is achieved at the cost of some loss in generality, relative for example to two-dimensional stochastic context free grammars. Nevertheless, TR is applicable to many common layouts, and it is hoped that its efficiency and relative ease of use may spur the further development of such DID-based methods for layout analysis.

In this chapter, we introduce turbo recognition from several points of view. We begin with a brief discussion of turbo decoding, a recent development in communication theory, and describe its relation to TR. We review the communication system view of TR, before turning to a description of the image generation process in more detail. We then describe TR decoding in general terms, both operationally and algorithmically. We illustrate the decoding process by tracing through a run of TR on a simple example. The decoding algorithm is most easily derived using the message-passing formalism of graphical models (Appendix B), as we describe in the next chapter.

3.1 Relation to turbo codes

Turbo recognition is closely related to turbo decoding[1], and we briefly describe turbo codes here. The connection appears most explicitly in the decoding algorithms,

[1] TR can in fact be viewed as a highly specialized form of turbo decoding.

which we discuss in Chapter 4.

Turbo coding for the communication problem was discovered in 1993 by Berrou et al. [115]. Compared to state-of-the-art convolutional codes, turbo codes can achieve a far lower bit error rate on channels with a given signal-to-noise ratio (10^{-5} vs. 10^{-2}, at 1.7 dB), or conversely, reliable coding on channels with a far worse signal-to-noise ratio (bit error rate 10^{-5} at 1.7 dB — within 0.5 dB of the Shannon limit of 1.2 dB — vs. 4.0 dB). Turbo codes consist of two parallel convolutional codes (Frey [116], Heegard and Wicker [117]). The first convolutional code encodes the bit sequence as usual, while the second encodes a permutation of the original sequence. In this way, error patterns which are difficult for one of the codes to correct may be easy for the other code to correct. For example, burst errors with respect to one code appear as isolated errors to the other.

In TR, we apply this insight and the general methodology of turbo coding to the recognition of layout structure in document images. Transposition of the image, horizontal to vertical, plays the role of the permutation in turbo coding. To make the analogy explicit, consider two ways of ordering image pixels into a one-dimensional sequence. The first is row-wise, with the pixels ordered left-to-right within a row, starting from the top row and ending with the bottom row. The second is column-wise, with the pixels ordered from top-to-bottom within a column, starting with the left-hand column and ending with right-hand column. These two sequences provide "orthogonal" views of the source data, in close analogy to turbo codes. This enhances the noise robustness of TR, since information in the horizontal direction can be brought to bear on the interpretation of structure in the vertical direction, and vice versa. The precise manner in which this feedback occurs will be described further below.

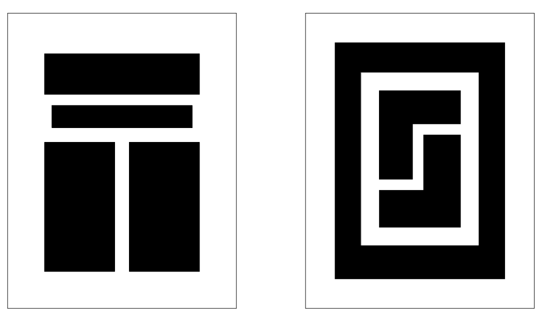


Figure 3.1: Examples of Manhattan layouts.

Layout structure in TR is described by two grammars, corresponding to the two convolutional encoders in turbo coding. Since these grammars act independently in orthogonal directions, TR is targeted principally at Manhattan layouts (see Figures 3.1 and 1.2), although TR may be useful in analyzing more general layouts as well. As a result, a typical application of TR requires that the input ("observed") image is approximately aligned on a rectangular grid. We take the point of view that it is easier to deskew an image than it is to decode detailed structure within it. For the time being, existing techniques should be sufficient to accomplish the necessary alignment.

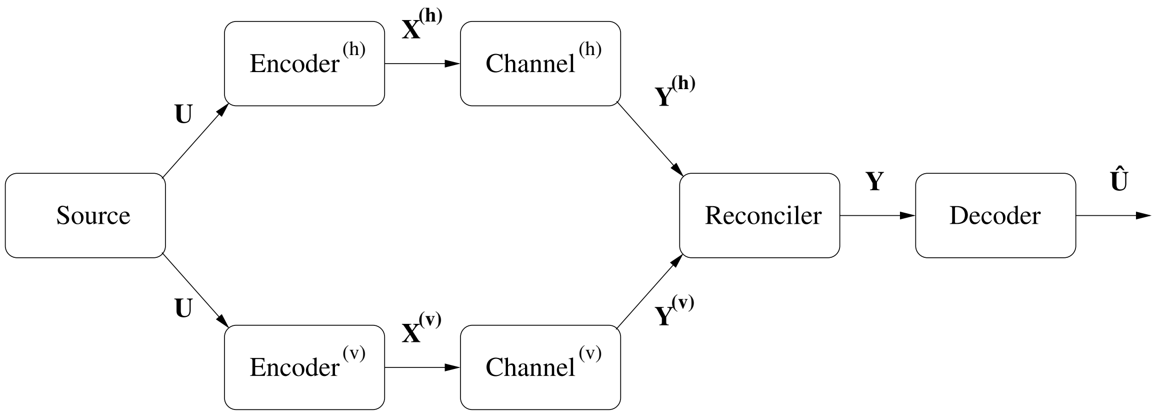


Figure 3.2: A communication system view of turbo recognition.

3.2 A communication system view of TR

The communication system view of TR is shown in Figure 3.2. This can be viewed as a specialization of the general DID framework given in the previous chapter (Figure 2.1), targeted to a particular class of page layouts. As in DID, all of the elements in this figure except for the decoder amount to a generative model for images. A single source message U, viewed now as a two-dimensional field, is encoded into *two* ideal images $X^{(h)}$ and $X^{(v)}$ (where the superscripts h and v stand for "horizontal" and "vertical," respectively). These images are transmitted through separate noise channels, resulting in two corrupted images $Y^{(h)}$ and $Y^{(v)}$. The reconciler passes $Y^{(h)}$ and $Y^{(v)}$ to the decoder as a single image Y if and only if they are identical. Otherwise the reconciler passes a null image to the decoder. The objective of the decoder is to recover the message field U, given the observed data Y and prior knowledge (parameters) embodied in the generative model. As before, the minimum probability of error is achieved when the decoder returns the field $\hat{U}$ that maximizes the posterior, $\hat{U} = \arg\max_U P(U|Y)$. We will call U the *input* field, X (generically) the *output* field, and Y the *observed* field.

The redundancy implied by encoding the U image field twice, horizontally and vertically, is in fact a hallmark of turbo coding. If we eliminated the reconciler, and instead transmitted U, $X^{(h)}$, and $X^{(v)}$ through a single noisy channel (resulting in the receipt of three images-worth of data at the decoder input[2]), then the TR image generation scheme would be a simple form of turbo coding.

Of course, what the decoder receives in practice is a single observed image, not three images, and this motivates the introduction of the reconciler. As a consequence, the generative scheme is very inefficient, since in the presence of noise, it is quite unlikely that $Y^{(h)}$ and $Y^{(v)}$ will be identical. Fortunately, our concern is not in generating images. From

[2]In a typical turbo code, $X^{(h)}$ and $X^{(v)}$ would be "punctured" for efficiency reasons, reducing the total amount of data to two times the original amount, instead of three. The convolutional encoders would also be selected for their error-correcting properties, rather than their ability to describe image layout. Further details can be found in Frey [116].

the decoding point of view, we are only interested in the population of images that succeed in being produced. The reconciler is a mathematical artifice which is necessary for us to construct a joint probability model for U, $X^{(h)}$, $X^{(v)}$, and Y. It does not appear explicitly in the decoding algorithm itself.[3]

The decoding algorithms used by TR are derived from graphical models [118], as applied in particular to turbo codes [116]. For those readers familiar with the HMM literature, decoding consists of applications of the forward-backward algorithm, once per row and once per column, iterated to convergence. As in the case of turbo codes, while TR is not guaranteed to recover the optimal result, [4] TR produces results that are nearly statistically optimal. These points will be described in detail in following chapters.

We now consider the decoding problem in more detail to gain some intuition into the problem. Note that a valid input message U must now satisfy (i.e. be accepted by) two transducers instead of the usual one. Let L^H (L^V) denote the set of images whose rows (columns) all drive the horizontal (vertical) transducer into its accepting state(s). The prior distribution on U, $P(U|L^H, L^V)$, can then be considered as a distribution $P(U)$ restricted to the intersection of L^H and L^V.

The decoding problem is to find the message image $\hat{U}$ maximizing the posterior distribution $P(U|Y, L^H, L^V)$. This is a hard problem in general, but if Y is not the null image (which of course is always the case in practice) then we have effectively observed Y^H and Y^V (since $Y^H = Y^V = Y$), so that $Y^H \to X^H \to U \to X^V \to Y^V$ is a Markov chain. Then the problem is to find the message image $\hat{U}$ maximizing the posterior distribution $P(U|Y^H, Y^V, L^H, L^V)$ in

$$
\begin{aligned}
&P(U|Y^H, Y^V, L^H, L^V)P(Y^V, L^V|Y^H, L^H) \\
&\quad = P(U, Y^V, L^V|Y^H, L^H) \\
&\quad = P(Y^V, L^V|U)P(U|Y^H, L^H) \\
&\quad = \prod_j P(Y_j^V, L_j^V|U_j) \\
&\qquad \times P(U_j|U_1, \ldots, U_{j-1}, Y^H, L^H) \\
&\quad = \prod_j P(Y_j^V, L_j^V|U_j) \\
&\qquad \times \prod_i P(U_{i,j}|U_{i,1}, \ldots, U_{i,j-1}, Y_i^H, L_i^H) \\
&\quad \approx \prod_j P(Y_j^V, L_j^V|U_j) \\
&\qquad \times \prod_i P(U_{i,j}|Y_i^H, L_i^H).
\end{aligned} \tag{3.1}
$$

Here, j is a column index, U_j is the jth column of the image U, L_j^V is the event that U_j drives the vertical transducer into an accepting state, and Y_j^V is the jth column of the

[3] A similar modeling philosophy applies in the use of HMMs for speech recognition. Few speech researchers would suggest that Markov models are faithful representations of human speakers, and yet such models have proven to be adequate for many recognition tasks.

[4] See Weiss and Freeman [119] for theoretical progress on this point.

observed image Y^V. Likewise, i is a row index, U_i is the ith row of the image U, L_i^H is the event that U_i drives the horizontal transducer into an accepting state, and Y_i^H is the ith row of the observed image Y^H. Finally, $U_{i,j}$ is the (i,j)th pixel of the image U. Now, (3.1) is maximized over U by maximizing

$$P(Y_j^V, L_j^V | U_j) \prod_i P(U_{i,j} | Y_i^H, L_i^H)$$

independently for each column U_j of U. For this purpose, the product distribution $\prod_i P(U_{i,j}|Y_i^H, L_i^H)$ provides a prior on the column U_j. Each factor in this product is the marginal posterior distribution $P(U_{i,j}|Y_i^H, L_i^H)$, which can be found by the forward-backward algorithm through the horizontal trellis for row i.

The approximation in (3.1) is exact when $U_{i,j}$ is conditionally independent of $U_{i,1}, \ldots, U_{i,j-1}$ given Y_i^H, L_i^H. This will be true if paths through the trellis beginning $U_{i,1}, \ldots, U_{i,j-1}$ have probability one, i.e., if one path takes all the probability. Thus the approximation will be good if the posterior distribution of U given Y^H, L^H is sharply peaked. Intuitively, by feeding in the posterior distribution of the horizontal decoding as the prior distribution of the vertical decoding, the posterior distribution becomes more peaked. This leads to the iterative algorithm, in which the posterior distribution of the vertical decoding is then fed in as the prior distribution of the horizontal decoding, whereupon the process is repeated until convergence.

As mentioned previously, the convergence properties of the turbo decoding algorithm itself have yet to be fully understood. The empirical success of turbo codes encourages us to believe that this framework achieves high performance in the image recognition context as well.

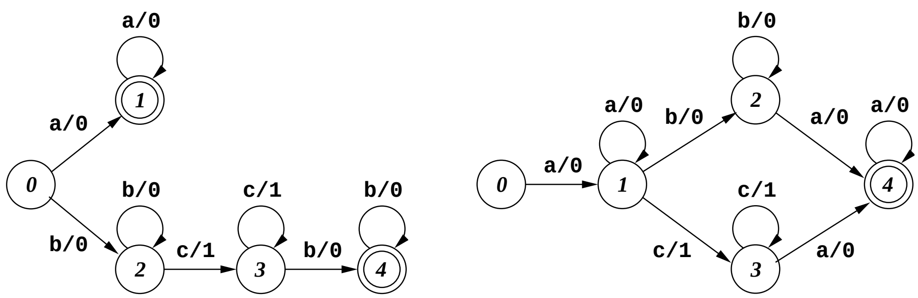
3.3 TR image generation in detail

We now turn to a more detailed description of the encoder and channel which form the core of the TR generative model. These components will be combined to form a unified probabilistic model in Chapter 4.

The encoders are modeled as finite-state transducers (FSTs)[5], which can be thought of as translating a one-dimensional sequence of symbols in some input alphabet to a corresponding sequence in an output alphabet. A more detailed treatment of FSTs is given in Appendix A. For example, the horizontal FST in Figure 3.3a accepts input sequences, construed as coming from an image row, that are described by the regular expression $a^+ \mid b^+c^+b^+$, and it emits corresponding output sequences $0^+ \mid 0^+1^+0^+$. Similarly, the vertical FST in Figure 3.3b accepts input sequences $a^+(\ b^+|c^+)\ a^+$, construed as coming from an image column, and it also emits output sequences $0^+ \mid 0^+1^+0^+$.

We now imagine having one horizontal FST on each image row, and similarly one vertical FST on each image column. The only two-dimensional input *images* (thought of as generalizing one-dimensional *sequences*) that are consistent with both collections of FSTs are those having the layout shown in Figure 3.4. If we interpret the output symbol "1"

[5] Non-finite state machines, such as recursive transition networks, could also be used in principle, although a different decoding algorithm would be required.



(a) (b)

Figure 3.3: Finite state transducers: (a) horizontal; (b) vertical. The start state is labeled 0 in both instances, and accepting states are marked with double circles. Transitions are labeled with an input and output symbol in the format "input/output".

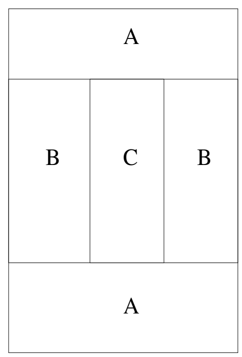


Figure 3.4: Sample layout described by the FSTs in Figure 3.3. All of the pixels in a given rectangular region are meant to be labeled by the symbol in its center.

as "black" and the symbol "0" as "white," we see that the FSTs translate such an input image into a single black rectangle on a white background. Note that while a and b are both mapped to the output symbol 0, they are not redundant. If, for example, all instances of b were replaced by a in the above grammars, then an amorphous shape, and not just rectangles, would be considered grammatical[6].

To summarize, the horizontal encoder in the communication system view of TR (Figure 3.2) is a collection of identical horizontal FSTs, one per row. Similarly, the vertical encoder is a collection of identical vertical FSTs, one per column. The two sets of FSTs taken together define the class of layouts that can be generated (and conversely, decoded) by the given TR model.

Figure 3.5 illustrates explicitly how the 2D message field U is encoded into the noise-free output image X[7]. Each horizontal FST converts a row of the input field U into

[6]Such simple "blob" grammars in fact exhibit some nice properties, as described in Sections 7.1.4 and 8.6.

[7]Strictly speaking, we should be referring to *two* images $X^{(h)}$ and $X^{(v)}$, and these two images in fact need not be identical (only $Y^{(h)}$ and $Y^{(v)}$ must be identical). For pedagogical purposes, it is simpler to just treat $X^{(h)}$ and $X^{(v)}$ as identical for now.

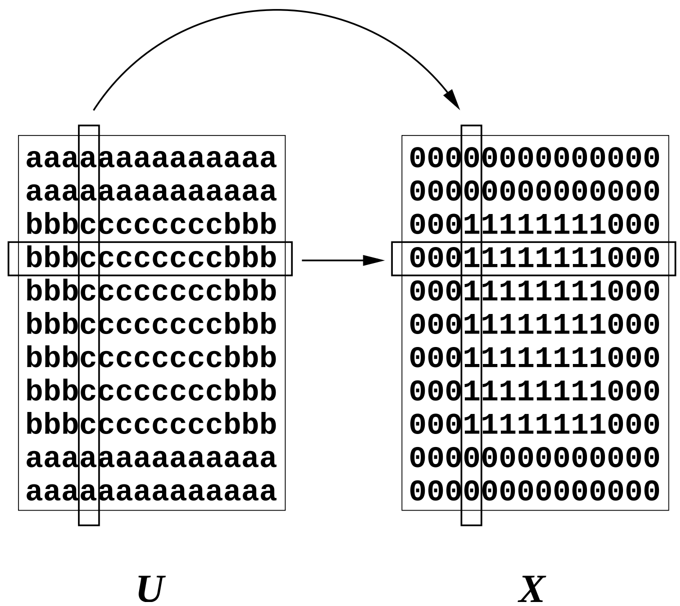
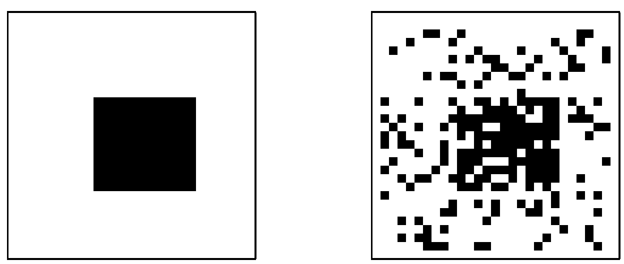


Figure 3.5: The two-dimensional input message field U is encoded by the horizontal and vertical FSTs row-by-row and column-by-column into separate copies of the output image X.



(a) (b)

Figure 3.6: (a) Source image with one rectangle; (b) a sample noisy version.

a corresponding row of X, and each vertical FST converts a column of the input field U into a column of X. The encoding redundancy is again clear. Note that each FST is acting independently of the others. The reconciler (two steps later in the generation process) is the only part of the process that enforces consistency of the horizontal and vertical outputs. The decoding algorithm deals with this independence through an iterative process, as we shall see.

Image encoding is thus largely a deterministic process. A stochastic element is introduced into encoding only when two or more outgoing edges from a given state are labeled with the same input symbol. This is described more fully in Chapter 4.

As in DID, the main stochastic element in the TR model is the channel (or horizontal and vertical channels, to be more precise). We follow typical DID practice and take the channel to be simple independent bit-flip noise, in which each output image pixel is

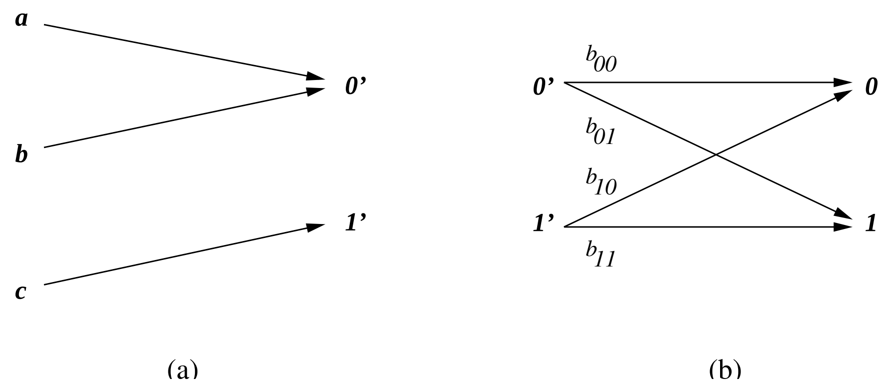


Figure 3.7: (a) Deterministic relation between U and X; (b) Noise channel between X and Y.

stochastically mapped to an observed image pixel. The effect of the channel is illustrated in Figure 3.6. The channel is specified completely by the conditional probability distribution $P(Y|X)$, which factorizes in this case into a product over pixels. For example, suppose both X and Y are binary images, and the noise process is a simple symmetric bit-flip channel. Then the conditional probability of Y given the ideal image X is

$$P(Y|X) = \prod_i (y_i(1-\pi)^{x_i}\pi^{(1-x_i)} + (1-y_i)\pi^{x_i}(1-\pi)^{(1-x_i)})$$

where π is the bit-flip probability (i.e. $\pi = P(1|0) = P(0|1)$), x_i and y_i take on the values 0 or 1, and i is an index which runs over all pixels in the image.

In general, the channel can be specified by an $n \times m$ array $b_{kl} \equiv P(Y = l|X = k)$, where n is the size of the output alphabet, m is the size of the observed alphabet, and (in a slight abuse of notation) l indexes the observed alphabet, and k indexes the output alphabet. b thus satisfies the constraint $\sum_l b_{kl} = 1$. At present, we set b manually, based roughly on the density of black pixels we expect to see in the image.

The channel adds considerably to the expressive power of TR models. We will in fact exploit the capabilities of the channel to model *clean* images of text in Chapter 8.

We can summarize the encoding and channel transmission steps with Figure 3.7. The input image U is encoded in a largely deterministic manner into the output image X (Figure 3.7a). X is then subjected to a stochastic noise process (Figure 3.7b), to appear as the observed image Y. Given such a generative framework, a key question that remains is its expressiveness in describing layout. We consider this in detail in Chapter 7 and following chapters.

3.4 TR in the context of DID

Here we briefly attempt to place TR within the larger context of DID. DID is meant to be a framework for DIA tasks in general. In principle, massive DID models could be developed that incorporate knowledge of the physical and logical processes underlying the formation of page images, at the level of pixels, characters, words, lines, pages, documents, natural language, etc. At the same time, specialized DID models could be developed to handle small snippets of such a massive scheme, such as the analysis of image skew[8].

TR is intermediate between the massive and the specialized, and aims to be a reasonably general purpose layout analysis method. It is targeted mainly at analyzing structure on the scale of the full page image, although the length scale of interest in TR can vary with image resolution, amongst other things. At the low end, TR models could be extended by incorporating DID character models, and at the other end, TR models could themselves be embedded in higher-level models of document structure, where the grammatical basis of TR may be useful in this regard.

As mentioned earlier in Section 2.2.4, powerful DID layout analysis techniques have been developed based on stochastic context free grammars [1]. These have been applied primarily to recognition of mathematical expressions [100, 114], which provides a particularly challenging 2D layout analysis problem. Related work has also been done by the author [120] using two-dimensional rectangular grammars [121]. The time complexity of these methods is relatively high (at least $O(N^{5/2})$), preventing them from seeing much practical use. Similar sentiments about the difficulty of using context-free (and context-sensitive) grammars for layout analysis have been expressed by Conway [122]. The linear time complexity of TR makes it considerably more attractive as a layout analysis method.

In previous applications of DID, such as the character recognition of text lines, the source message has been conceived of as a linear sequence of symbols. The description in terms of parse trees of more complex structures such as mathematical expressions is arguably also naturally linearizable. TR focuses on decoding two-dimensional regions ("shapes"), and the source message in this case is much more naturally thought of as a two-dimensional field instead of a one-dimensional sequence.

On a related note, previous work in DID concentrated on decoding to the level of characters. There was no direct way to extract higher level physical and logical concepts such as "text column" and "title zone." Such structural concepts emerged only as a side effect of decoding a page at the character level (cf. the yellow pages example in the original DID paper [2]). While TR is in a sense lower level than such previous work, in that the symbols that make up the source message are at the pixel level, an important contribution of TR is to provide a way of introducing language appropriate to the page layout analysis task into the DID framework, through its imaging model based on two regular grammars.

3.5 TR operationally

In this section we summarize TR operationally by describing the input and output of a typical TR decoding run. The inputs are the prior probability of the input field U, the finite state transducers (grammars), the noise channel, and the observed image. The output typically is the estimate $\hat{U}$ represented as a 2D image. We now go through each of these in turn.

[8]The work of Kam and Kopec [94] on baseline finding points to some of the possibilities in this direction.

```
% One column horizontal grammar
% 0 | 121, where 2 is the only printing symbol.
%
NTRANSITIONS 8
NINSYMBOLS 3
NOUTSYMBOLS 2
FROM S0 TO A1 IN 0 OUT 0 PROB 1.0
FROM A1 TO A1 IN 0 OUT 0 PROB 1.0
FROM S0 TO B1 IN 1 OUT 0 PROB 1.0
FROM B1 TO B1 IN 1 OUT 0 PROB 1.0
FROM B1 TO B2 IN 2 OUT 1 PROB 1.0
FROM B2 TO B2 IN 2 OUT 1 PROB 1.0
FROM B2 TO B3 IN 1 OUT 0 PROB 1.0
FROM B3 TO B3 IN 1 OUT 0 PROB 1.0
START S0
FINAL A1 B3
```

Figure 3.8: File representation of the FST corresponding to the standard one-column grammar, $0^+|1^+2^+1^+$.

```
% One row vertical grammar
% 010 | 020, where 2 is the only printing symbol.
%
NTRANSITIONS 12
NINSYMBOLS 3
NOUTSYMBOLS 2
FROM S0 TO A1 IN 0 OUT 0 PROB 0.5
FROM A1 TO A1 IN 0 OUT 0 PROB 1.0
FROM A1 TO A2 IN 1 OUT 0 PROB 1.0
FROM A2 TO A2 IN 1 OUT 0 PROB 1.0
FROM A2 TO A3 IN 0 OUT 0 PROB 1.0
FROM A3 TO A3 IN 0 OUT 0 PROB 1.0
FROM S0 TO B1 IN 0 OUT 0 PROB 0.5
FROM B1 TO B1 IN 0 OUT 0 PROB 1.0
FROM B1 TO B2 IN 2 OUT 1 PROB 1.0
FROM B2 TO B2 IN 2 OUT 1 PROB 1.0
FROM B2 TO B3 IN 0 OUT 0 PROB 1.0
FROM B3 TO B3 IN 0 OUT 0 PROB 1.0
START S0
FINAL A3 B3
```

Figure 3.9: File representation of the FST corresponding to the standard one-row grammar, $0^+1^+0^+|1^+2^+1^+$.

```
% Stochastic mapping from two output symbols to two observed symbols.
NOUTSYMBOLS 2
NOBSSYMBOLS 2
0.999 0.001 0.1 0.9
```

Figure 3.10: Sample channel file describing the stochastic mapping from output symbols to observed symbols, where the first two entries correspond to the first output symbol, and the following two entries correspond to the second.

We take the prior probability of the input field U to factorize into a product of pixel-wise probabilities, equal to the uniform distribution $P(u_{ij}) = 1/n$, where i and j are image row and column indices, u_{ij} is the input symbol at pixel position (i, j), and n is the size of the input alphabet. Other possibilities such as the incorporation of "linguistic" constraints can often be accomplished simply within the formalism.

The horizontal and vertical FSTs are specified in two separate files, examples of which are shown in Figures 3.8 and 3.9. Each file gives the total number of transitions in the FST, the size of the input and output alphabets, and a list of transitions, each described by its begin and exit states, the input and output labels, and a transition probability. This probability is for the most part equal to one, for reasons described in the next chapter. The regular expression at the top of each file gives the the grammar on the input alphabet which is satisfied by each machine (where for coding convenience we have remapped the input alphabet from alphabetic $\{a, b, c, \ldots\}$ to numeric $\{0, 1, 2, \ldots\}$).

The noise channel is specified by an array of bit-flip probabilities where each entry p_{kl} is the probability that output symbol k appears as observed symbol l. These probabilities satisfy the constraint $\sum_l p_{kl} = 1$. An example channel file is shown in Figure 3.10, where we sweep through the array of probabilities in row-major order. At present we set these parameters manually, based roughly on the level of noise we expect to see in the image. We will have more to say about this aspect of TR modeling in a later chapter.

In our present implementation, the observed image is a binary image. For layout analysis purposes, it is often sufficient to use low-resolution versions of the original images. The dimensions of these images are typically 100 to 300 pixels across, corresponding to a (linear) reduction of about 10 times in size. As mentioned earlier, we assume the image has been preprocessed to remove any skew.

Our policy for stopping the turbo iterations is to simply inspect the current best estimate $\hat{U} = \arg\max_U P(U)$ and stop when this reaches a stable value. In runs with a large number of images, we stop after a number of iterations which has been fixed by inspecting the performance of TR on similar images. In all cases so far, the number of iterations ranges from two to seven. More sophisticated policies, such as keeping track of the likelihood of the current $\hat{U}$, are easily envisioned.

We have found two failure modes for our current TR implementation. One is the presence of oscillations, in which the horizontal and vertical passes alternate between two different $\hat{U}$ configurations. This is theoretically allowed[9], although in our experience, this happens rarely, and only on very small images. The second failure mode is the occurrence of non-normalizable probabilities. Normally the decoding algorithm evaluates quantities at

[9] See the next section, and the following chapter.

each pixel which can be interpreted as probabilities (see Chapter 4), which in particular can be normalized to sum to one. However, if all of the quantities evaluate to zero, then this correspondence no longer holds, and the algorithm fails. This is most likely a computational issue (underflow), which we are presently investigating. This failure mode tends to occur on larger images with complex layouts.

An additional subtlety in our present formulation is the use of deterministic annealing, which helps to damp out both of the failure modes mentioned above. This requires the specification of a single parameter β, as discussed further in Sections 4.2.6 and 9.4.

The result of the TR algorithm is a quantity termed a *belief* by Pearl [118], which can be thought of as a probability distribution over input symbols at each pixel. This result is typically post-processed to choose the input symbol of maximum belief at each pixel, resulting in a two-dimensional field $\hat{\mathbf{U}}$ of the same dimensions as the observed image. This can be considered to be the final result of TR decoding.

In summary, the inputs to TR are files representing the horizontal and vertical FSTs and the noise channel, plus the observed (scanned) image. Additionally, a prior probability on $\mathbf{U}$ (typically uniform) and a stopping policy (manually determined at present) must be specified. The output of TR is an image $\hat{\mathbf{U}}$, equal in size to the observed image, that gives the input symbol of maximum belief at each pixel. This image in general summarizes both the physical and logical layout analysis results of TR.

3.6 TR algorithmically

In this section, we presume familiarity with HMM techniques, as described in standard textbooks such as Rabiner and Juang [88]. Given this background, the TR algorithm can be summarized as follows. The algorithm begins with the probability on $\mathbf{U}$ initialized in some fashion (typically to a uniform prior distribution). The $P(\mathbf{U})$ array plays the role of the transition probabilities in HMMs[10]. The forward-backward algorithm is then applied (independently) along each row, which has the result of updating $P(\mathbf{U})$. We call this one horizontal pass of the TR algorithm. This is followed by a vertical pass, in which the forward-backward algorithm is applied along each column, using the updated value of $P(\mathbf{U})$. This then results in a further update to $P(\mathbf{U})$. We call the combination of a horizontal and a vertical pass one *iteration* of the TR algorithm. These iterations are continued until $P(\mathbf{U})$ settles down to some constant value (typically one or zero at each pixel). As mentioned in the previous section, this convergence is not guaranteed. The horizontal and vertical passes can cause $P(\mathbf{U})$ to oscillate between two values, though empirically this happens only rarely.

In summary, the TR algorithm consists of iterative horizontal and vertical passes of the forward-backward algorithm, which communicate with each other through updates on $P(\mathbf{U})$, this having the effect of biasing the transition probabilities for the orthogonal

[10]More precisely, $P(\mathbf{U})$ plays the role of a transition probability at each pixel. If $P(\mathbf{U})$ varies spatially, then the model becomes akin to an inhomogeneous Markov chain, rather than a static HMM per se. For completeness, we also mention that the stochastic FSTs used in TR have transition probabilities, just like HMMs, but typically these are uninteresting, having a (static) value of one for the most part, as described elsewhere.

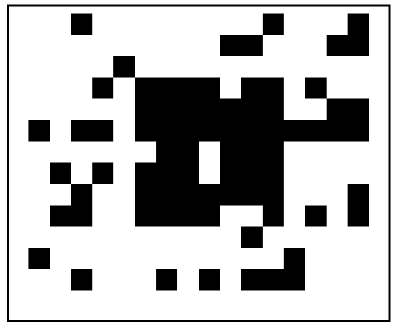


Figure 3.11: Noisy observed image.

pass.

3.7 Example run

Here we illustrate the progress of a TR run in a visual manner. Suppose we are given the noisy image shown in Figure 3.11. We attempt to decode this image using a standard one-rectangle grammar, $a^+ \mid\ b^+c^+b^+$ horizontally, and $a^+(b^+ \mid c^+)a^+$ vertically. Suppose we begin with a column pass. In this case, decoding along each column corresponds to finding the (at most) one best interval of black which explains the observed data along that column.

The result after applying the forward-backward algorithm to all the columns in the image is shown in Figure 3.12a. Intuitively, the TR algorithm is "denoising" the image by finding either zero or one interval of black pixels within each column. While from a pedagogical standpoint, this is quite adequate, we point out some subtleties to prevent misunderstanding. First of all, we emphasize that the TR algorithm is updating a probability *distribution*, so in fact using a single image to represent the progress of the algorithm is somewhat misleading. TR proceeds iteratively without making any hard decisions about which image is "best" at any intermediate step. The image that we present is for illustrative purposes only, and is the maximum of the calculated posterior probability (or the maxproduct analog thereof) at every pixel. This is shown in two ways, in terms of the input alphabet $\{a, b, c\}$ on the left, and then as a binary image on the right, where the c pixels are shown in black, and the remaining pixels are shown in white (see Figure 3.12a). Finally, we note that some of the columns are solid white, although it would appear that a single interval of black pixels somewhere in the column would better describe the observed data. This arises because more than one optimal decoding exists, so when combined together, neither dominates, and in fact the "blank" column solution wins out[11].

After the vertical pass is completed, the posterior probability $P(U)$ is updated (where again U is a 2D collection of variables, one at each image pixel, whose domain is the set of input symbols). This effectively changes the transition probabilities for the

[11] This is related to the fact that the maxproduct algorithm (see Chapter 4 and Appendix B) does not necessarily produce a grammatical result when there are two U fields which are equally likely (i.e. there are "ties" in the probability score).

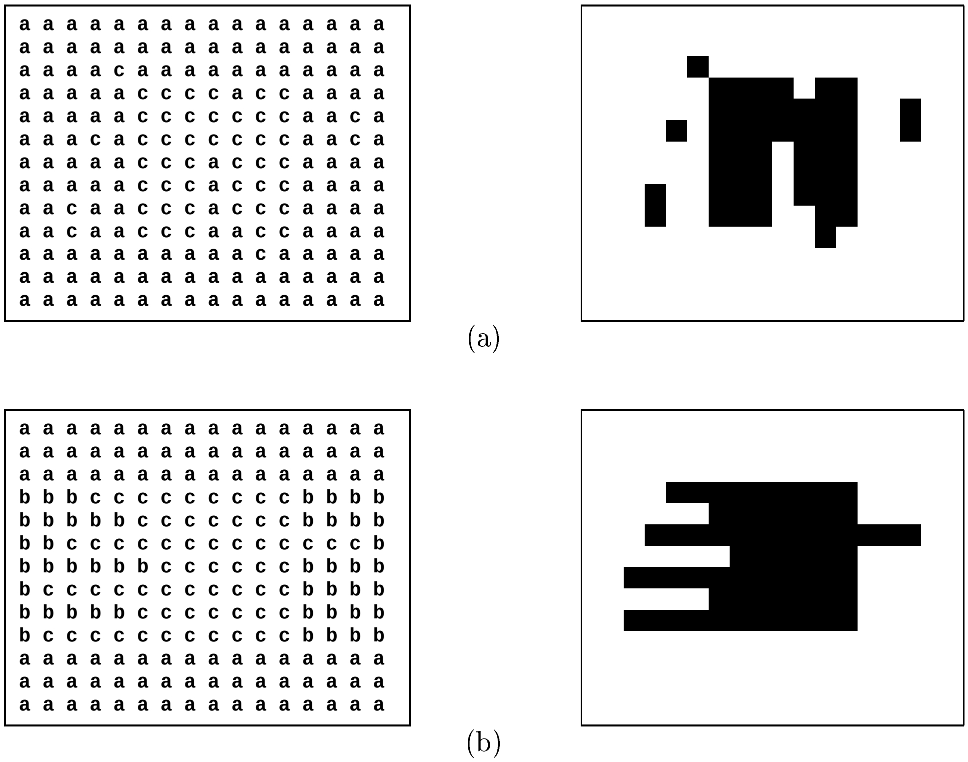


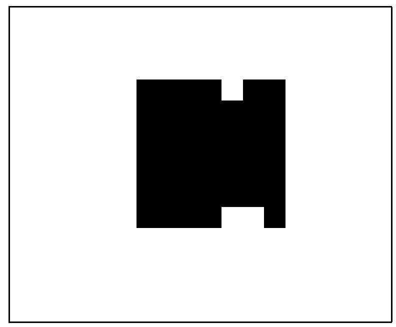
Figure 3.12: Representation of the TR result after the first (a) vertical and (b) horizontal pass.

transducers involved in the horizontal pass. The result of the horizontal pass is illustrated in Figure 3.12b (subject to the same caveats as mentioned above for the vertical pass).

The results for the vertical and horizontal passes in the second TR iteration are illustrated in Figure 3.13. This shows the algorithm has proceeded quite far in producing a one-rectangle interpretation of the original image, although there remain some features which satisfy the grammar in one direction but not in the orthogonal one.

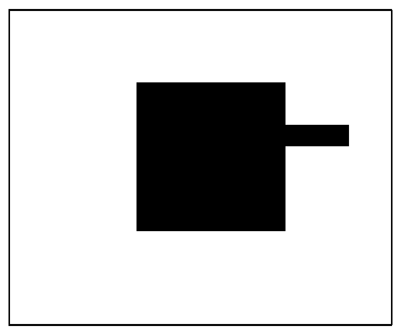
In this case, four TR iterations are required to converge to a stable solution, which is illustrated in Figure 3.14. This indeed turns out to equal the ground truth rectangle which was corrupted to produce Figure 3.11.

a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
b	b	b	b	b	c	c	c	c	a	c	c	b	b	b	b
b	b	b	b	b	c	c	c	c	c	c	c	b	b	b	b
b	b	b	b	b	c	c	c	c	c	c	c	b	b	b	b
b	b	b	b	b	c	c	c	c	c	c	c	b	b	b	b
b	b	b	b	b	c	c	c	c	c	c	c	b	b	b	b
b	b	b	b	b	c	c	c	c	c	c	c	b	b	b	b
b	b	b	b	b	c	c	c	c	a	a	c	b	b	b	b
a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a



(a)

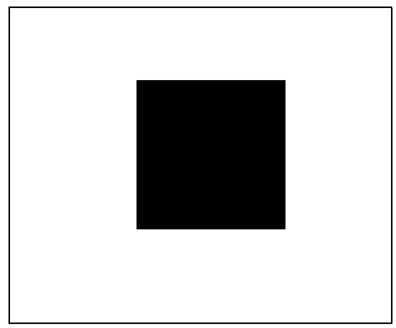
a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
b	b	b	b	b	c	c	c	c	c	c	c	b	b	b	b
b	b	b	b	b	c	c	c	c	c	c	c	b	b	b	b
b	b	b	b	b	c	c	c	c	c	c	c	c	c	c	b
b	b	b	b	b	c	c	c	c	c	c	c	b	b	b	b
b	b	b	b	b	c	c	c	c	c	c	c	b	b	b	b
b	b	b	b	b	c	c	c	c	c	c	c	b	b	b	b
b	b	b	b	b	c	c	c	c	c	c	c	b	b	b	b
a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a



(b)

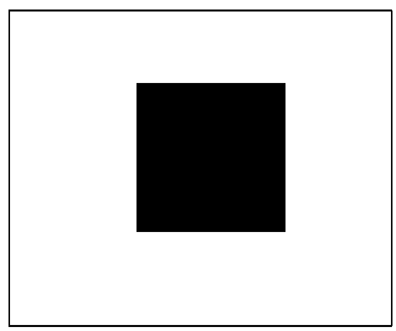
Figure 3.13: TR result after the second (a) vertical and (b) horizontal pass.

a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
b	b	b	b	b	c	c	c	c	c	c	c	b	b	b	b
b	b	b	b	b	c	c	c	c	c	c	c	b	b	b	b
b	b	b	b	b	c	c	c	c	c	c	c	b	b	b	b
b	b	b	b	b	c	c	c	c	c	c	c	b	b	b	b
b	b	b	b	b	c	c	c	c	c	c	c	b	b	b	b
b	b	b	b	b	c	c	c	c	c	c	c	b	b	b	b
b	b	b	b	b	c	c	c	c	c	c	c	b	b	b	b
a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a



(a)

a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
b	b	b	b	b	c	c	c	c	c	c	c	b	b	b	b
b	b	b	b	b	c	c	c	c	c	c	c	b	b	b	b
b	b	b	b	b	c	c	c	c	c	c	c	b	b	b	b
b	b	b	b	b	c	c	c	c	c	c	c	b	b	b	b
b	b	b	b	b	c	c	c	c	c	c	c	b	b	b	b
b	b	b	b	b	c	c	c	c	c	c	c	b	b	b	b
b	b	b	b	b	c	c	c	c	c	c	c	b	b	b	b
a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a



(b)

Figure 3.14: TR result after the fourth (a) vertical and (b) horizontal pass.

Chapter 4

A Graphical Model Formulation of Turbo Recognition

In this chapter, we treat turbo recognition within the framework of graphical models, an approach to statistical modeling that recently has attracted much interest in the statistics, artificial intelligence, and communication theory communities, amongst others [118, 116]. Although long-standing methods exist for performing probabilistic inference for HMMs [88], for example, which are closely related to the TR model, we have found that TR is sufficiently subtle that it is worthwhile to invoke graphical model techniques. In particular, we follow a straightforward recipe known as message passing in order to derive the TR decoding equations, which form the basis for our software. We are aided in this by the fact that essentially the same methods have already been used to treat turbo decoding, as developed by Kschischang and Frey [123] and McEliece et al. [124]. Our formulation follows the one given in the book by Frey [116]. We assume some familiarity with prerequisite work in this area in order to keep this chapter brief. Appendix B provides a succinct summary of graphical models.

4.1 The TR graphical model

The TR graphical model is a set of nodes and arcs, where the nodes represent probabilistic variables and the arcs denote conditional probability relationships between them. While graphical models look quite similar to other graphical objects that appear in this thesis[1], in particular FSTs (see Section 3.3 and Appendix A), they are in fact quite different. For example, a node in an FST represents a single distinct state. In contrast, a state node in the TR graphical model is a probabilistic variable whose domain is the *set* of FST states. We can think of one state node as roughly equivalent to the FST nodes all "collapsed" into one.

The ingredients of the TR graphical model are illustrated schematically in Figure 4.1. These include the input field $\mathbf{U}$, the horizontal output field $\mathbf{X}^{(h)}$, and the horizontal

[1]A third graphical representation that often appears in the literature is known as a trellis, which is described in Appendix A.

```
aaaaaaaaaaaaaa
aaaaaaaaaaaaaa
bbbccccccccbbb
bbbccccccccbbb
bbbccccccccbbb
bbbccccccccbbb
bbbccccccccbbb
bbbccccccccbbb
bbbccccccccbbb
aaaaaaaaaaaaaa
aaaaaaaaaaaaaa
```

U

```
00000000000000
00000000000000
00011111111000
00011111111000
00011111111000
00011111111000
00011111111000
00011111111000
00011111111000
00000000000000
00000000000000
```

$\mathbf{X}^{(h)}$

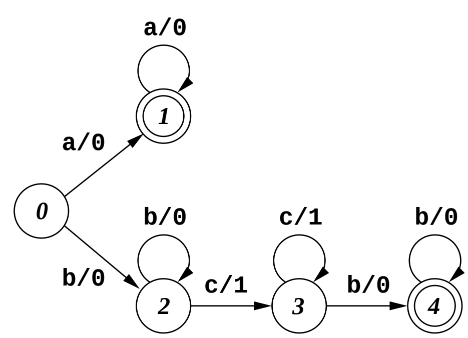
```
00000110010000
00000010000000
00010111110000
00011101101000
00010111110010
00011011111000
00001110101000
00011111111100
00011100111000
01000000010000
00001010000000
```

$\mathbf{Y}^{(h)}$

(a)

```
011111111111111
011111111111111
022233333333444
022233333333444
022233333333444
022233333333444
022233333333444
022233333333444
022233333333444
011111111111111
011111111111111
```

$\mathbf{S}^{(h)}$



(b)

Figure 4.1: The ingredients of the TR graphical model. (a) Sample configurations of the input field $\mathbf{U}$, the horizontal output field $\mathbf{X}^{(h)}$, and the horizontal observed field $\mathbf{Y}^{(h)}$. A separate probabilistic variable resides at each pixel position, whose domain is the alphabet (input, output, observed) associated with the type of field to which it belongs. (b) Sample configuration of the horizontal state field $\mathbf{S}^{(h)}$, where the numbers refer to the node labels of the FST shown on the right. The vertical fields $\mathbf{X}^{(v)}$, $\mathbf{Y}^{(v)}$ and $\mathbf{S}^{(v)}$ are not shown in this figure.

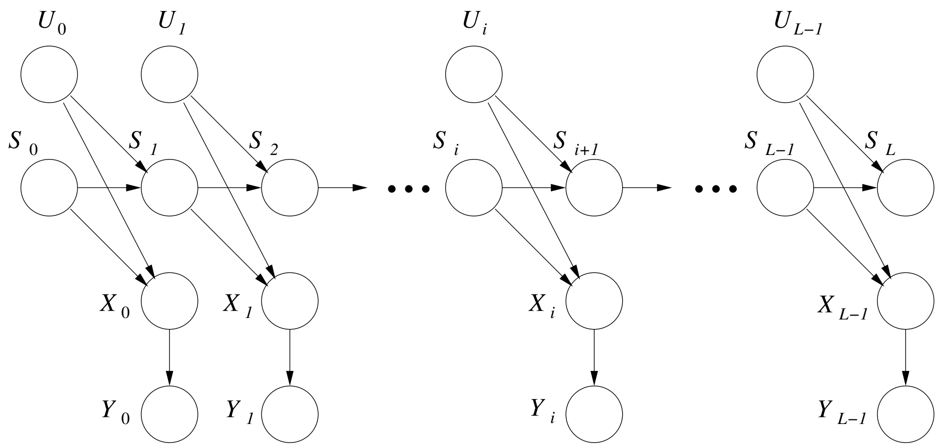


Figure 4.2: Initial TR graphical model for one row.

observed field $\mathbf{Y}^{(h)}$, all of which are equal size two-dimensional arrays of variables, whose domains are their respective alphabets (the vertical fields $\mathbf{X}^{(v)}$ and $\mathbf{Y}^{(v)}$ are not shown). In addition, there is a horizontal state field $\mathbf{S}^{(h)}$ (and corresponding vertical version $\mathbf{S}^{(v)}$), which is a two-dimensional collection of variables, each of whose domain is the set of FST states. We are guided in this choice of variables by previous work in the literature on HMMs and turbo codes. Alternative choices can be made, such as using the FST transitions instead states as random variables, and it may be of interest to explore such options in future work, for clarity of presentation and possible generalizations.

In the previous chapter (Section 3.3), we described the relationship between $\mathbf{U}$ and $\mathbf{X}^{(h)}$ as being mediated by a set of FSTs (one per row). In the present graphical model formulation, this relationship is mediated by the field $\mathbf{S}^{(h)}$, which indicates the (horizontal) FST state at each pixel. The relationship between $\mathbf{X}^{(h)}$ and $\mathbf{Y}^{(h)}$ is mediated by a noise channel, which acts independently at each pixel. Analogous statements hold in the vertical direction.

We note that the $\mathbf{U}$, $\mathbf{X}$, and $\mathbf{Y}$ fields that appear in Figure 4.1a are just sample configurations of those fields. The total number of configurations that the $\mathbf{U}$ field can take (and hence the size of the space on which $P(\mathbf{U})$ is defined), for example, is $|\Sigma_U|^N$, where $|\Sigma_U|$ is the size of the input alphabet, and N is the number of pixels in the image.

The complete TR graphical model is difficult to visualize. For descriptive purposes, we focus instead on a one-dimensional slice of the full model[2], say that part corresponding to the image row that is outlined in Figure 4.1a. The corresponding submodel is shown in Figure 4.2. For convenience, we refer to node variables collectively as $\vec{U}$, $\vec{S}$, $\vec{X}$, and $\vec{Y}$, and to the individual nodes generically as U_i, S_i, X_i, and Y_i, where i indexes pixel position within the row. Since $\vec{Y}$ corresponds to the observed image pixels, i ranges from 0 to $L-1$ for the U_i, X_i, and Y_i variables, where L is the width of the image. For the state variables $\vec{S}$, i ranges from 0 to L (i.e. one more than the other variables), since the variables U_i

[2]TR however cannot be said to *decompose* into these 1D components, since they are highly connected to each other.

and X_i (and correspondingly the observed image pixels Y_i) are emitted on the transitions *between* states[3].

The graph in Figure 4.2 is a shorthand notation that describes the probability distribution of the node variables, $P(\vec{U}, \vec{X}, \vec{Y}, \vec{S})$[4]. The arrows can be thought of roughly as describing cause-and-effect relationships. The variables S_i and U_i jointly influence (and largely determine) both S_{i+1} and X_i, reflecting the operation of the stochastic FST. The observed symbol Y_i is a stochastic function of X_i alone.

Since the root nodes S_0 and $\vec{U}$ have no incoming arrows, they enter the overall probability distribution via a product of priors for each one. The remaining variables then enter via conditional probabilities determined by the directed arrows in the graph. The structure of the overall probability distribution implied by the graphical model in Figure 4.2 is then of the following form:

$$\begin{aligned} P(\vec{S}, \vec{U}, \vec{X}, \vec{Y}) &\equiv P(S_0, U_0, X_0, Y_0, \ldots, S_{L-1}, U_{L-1}, X_{L-1}, Y_{L-1}, S_L) \\ &= P(S_0)P(U_0)P(S_1|S_0, U_0)P(X_0|S_0, U_0)P(Y_0|X_0)\ldots P(U_{L-1}) * \\ &\quad P(S_L|S_{L-1}, U_{L-1})P(X_{L-1}|S_{L-1}, U_{L-1})P(Y_{L-1}|X_{L-1}) \end{aligned}$$

It turns out that in all of our TR models so far, S_i and U_i together determine a unique value of X_i (although not necessarily the FST transition per se). Therefore, to streamline the presentation somewhat, we subsume the noise channel into a direct relationship between the pair (S_i, U_i) with Y_i, in effect removing X_i from the model[5].

An efficient method for performing inference (i.e., updating the probability model when some variables have been observed) in graphical models is known as *message passing*. In order for message passing to be guaranteed to work, the graphical model should have no loops, when the directions of the arrows are ignored [118]. This is to ensure both the convergence and the correctness of the message passing algorithm. In order to eliminate the loops in Figure 4.2, we use a trick called "node duplication" [116]. This is shown in Figure 4.3 for the loop involving U_0, S_0, S_1 and X_0. We add a redundant node U_0', which is understood to be identical to U_0, i.e., $P(u_0'|u_0) = \delta(u_0'|u_0)$, where $\delta(u_0'|u_0)$ is the Kronecker delta function. If we combine the pair of variables (S_0, U_0') to form a new random variable Z_0, the small loop in the model is eliminated. When the same transformation is done for each pair of variables (S_i, U_i)[6], then the TR graphical model becomes that shown in Figure 4.4. It is on this model that we will do message passing.

4.2 Derivation of the TR message passing equations

Given the graphical model shown in Figure 4.4, the TR decoding algorithm can be derived straightforwardly as an application of the message passing formalism. The general

[3] It is thus a slight misstatement to say that "S represents the FST state at each pixel." Rather, it represents the FST state *between* pixels.

[4] As mentioned in Appendix B, the graph is more accurately a shorthand for a whole *family* of probability distributions, whose structure is constrained by the connectivity of the graph.

[5] More formally, we can marginalize X_i away: $\sum_{x_i} P(x_i|s_i, u_i)P(y_i|x_i) = P(y_i|s_i, u_i)$.

[6] This is not done for $i = L$, since there is no U_L variable, so that Z_L is simply S_L, the state of the FST at the end of the row.

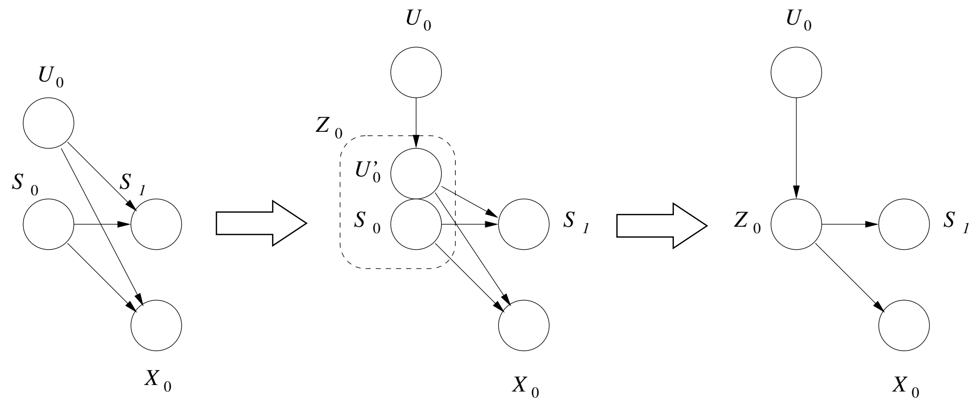


Figure 4.3: The duplicating node trick to get rid of a loop.

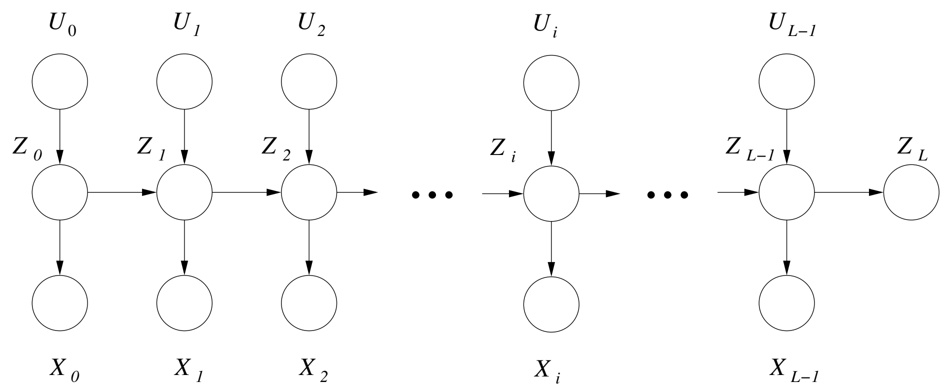


Figure 4.4: Final TR graphical model for one row, where $Z \equiv (S, U')$.

form of message passing is described in Appendix B. Here we concentrate on applying the general equations to the TR context.

4.2.1 Global TR model, and more loops

Up to this point, we have focussed on treating one-dimensional slices of the TR model, and for the most part we will continue to do so. In this section, we briefly step back to describe the TR model in its entirety.

Figure 4.5 summarizes schematically the entire TR graphical model[7] (where again the X nodes have been marginalized away, as described in the previous section). The $Y^{(h)}$ node represents the observed image (or more precisely, the set of random variables corresponding to the observed image pixels), as viewed by the horizontal FSTs. Similarly, $Y^{(v)}$ (which when instantiated happens to equal $Y^{(h)}$) represents the observed image pixels as seen by the vertical FSTs. $Z^{(h)}$ and $Z^{(v)}$ represent the collection of horizontal and

[7]The book by Frey [116] gives a more detailed representation of essentially the same graphical model for turbo codes, and the reader may find it interesting to compare the two.

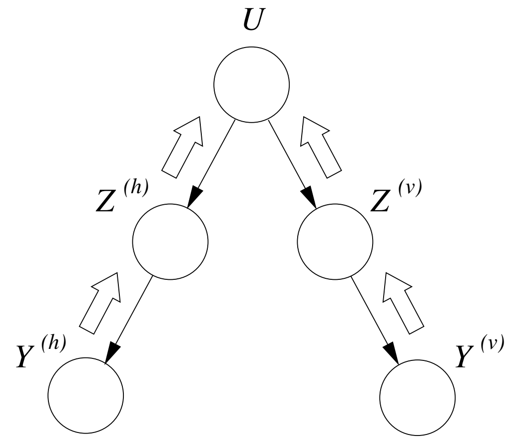


Figure 4.5: Schematic representation of the entire TR graphical model.

vertical FST state variables at all pixel positions, respectively. The U node represents the set of input symbol (message) variables, which are in one-to-one correspondence with the observed image pixels. The horizontal and vertical passes in TR are chiefly concerned with updating $P(U)$. In so doing, they permit the two portions of the graph, corresponding to the horizontal and vertical transducers, to communicate with each other (roughly speaking).

Given that the U node has two children, $Z^{(h)}$ and $Z^{(v)}$, the belief update at U has the following form,

$$
\begin{aligned}
BEL(U) &= \pi(U)\ \lambda(U) && (4.1)\\
&= \pi(U)\ \lambda_{Z^{(h)}}(U)\ \lambda_{Z^{(v)}}(U)\ \ . && (4.2)
\end{aligned}
$$

This relation also holds on a per-pixel basis. The value of $\pi(U)$ is simply the current value of $P(U)$. So our main concern here is to find expressions for the quantities $\lambda_{Z^{(h)}}(U)$ and $\lambda_{Z^{(v)}}(U)$. These have the same form for both the horizontal and vertical halves, so we will drop the h and v subscripts below and concentrate on deriving $\lambda_Z(U)$ for a generic Markov chain. This requires that we consider the quantities $\pi(Z)$ and $\lambda(Z)$ in addition.

Before turning to these, we make an important observation. Although it is not obvious from Figure 4.5, there is in fact another class of loopy structures in the TR graphical model that become evident when Figure 4.5 is "unrolled" to show the nodes at each pixel. We do not represent this graphically here, instead choosing to attempt a verbal description of one such loop. It is not necessary to understand this to follow the rest of this thesis.

We first extend our notation somewhat. Let $Z^{(h)}_{i,j}$ denote the Z node at position (i.e. column) j, in row i, in the horizontal half of the graph (so relative to Figure 4.4, we have switched from indices from i to j for labeling position within a row). Starting our loop from $Z^{(h)}_{i,j}$, we move one step along the chain (depicted in Figure 4.4) to $Z^{(h)}_{i,j+1}$. Then we follow one of the arrows in the graph to node $U_{i,j+1}$ (the input variable at row i and column $j+1$), where again, we note that the directions of arrows are to be ignored. We then can move down from $U_{i,j+1}$ to $Z^{(v)}_{i,j+1}$, the Z node at position (row) i, in column $j+1$, in the vertical half of the graph. We thus can hop from a $Z^{(h)}$ node to a $Z^{(v)}$ node at the same pixel position, through the intermediary of the U node at that position. Now we move along a vertical chain to a node at a position one row down, $Z^{(v)}_{i+1,j+1}$. We then hop to $Z^{(h)}_{i+1,j+1}$ through the intermediary of $U_{i+1,j+1}$. We move along a horizontal chain to a node one column back $Z^{(h)}_{i+1,j}$ and hop to $Z^{(v)}_{i+1,j}$ through $U_{i+1,j}$. Finally, we move up a row to $Z^{(v)}_{i,j}$ and reach $Z^{(h)}_{i,j}$ through $U_{i,j}$. In summary, we have moved around in a small loop in terms of image coordinates, using U as an intermediary to switch between horizontal and vertical steps on 4 different Z-chains. In this way, we have traced out one of many loops in the TR graphical model.

The consequence of having such loops is that message passing does not terminate with Equation 4.2. Rather, we can imagine message passing starting in the horizontal part of the graph (forward-backward along a Z-chain), with $P(U)$ being updated according to $P(U) = \pi(U)\ \lambda_{Z^{(h)}}(U)$ (where $\pi(U)$ is the previous value of $P(U)$). This updated $P(U)$ is then used in message passing in the vertical half, leading to another $P(U)$ update. The TR algorithm proceeds in this fashion, with messages passing back and forth between the horizontal and vertical halves of the network through U, until $P(U)$ converges.

Given this outline of the global TR algorithm, we now focus on deriving the message passing equations along a single row or column.

4.2.2 Recursion relation for $\pi(Z)$

Figure 4.6 shows the situation for the "forward" message pass for π at the node Z_{i+1}. The nodes are shown in a slanted orientation, relative to Figure 4.4, in order to show the parent-child relationships clearly. We note again for reference that the variable Z_i generically is a composite variable (S_i, U_i'), where U_i' is a duplicate of the variable U_i.

Following the rules of message passing in graphical models (Appendix B),

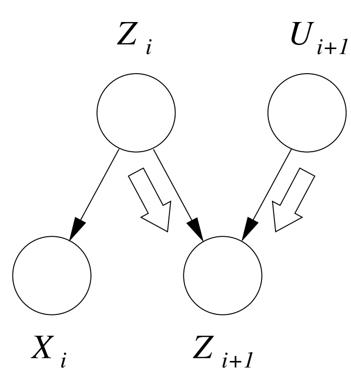


Figure 4.6: Nodes involved in message passing for $\pi(Z_{i+1})$.

$$\begin{aligned}
\pi(z_{i+1}) &= \sum_{z_i,u_{i+1}} P(z_{i+1} \mid z_i, u_{i+1})\ \pi_{z_{i+1}}(z_i)\ \pi_{z_{i+1}}(u_{i+1}) \\
&= \sum_{z_i,u_{i+1}} [P(s_{i+1} \mid s_i, u_i')\ \delta(u_{i+1}', u_{i+1})]\ [\pi(z_i)\ \lambda_{y_i}(z_i)]\ P(u_{i+1}) \\
&= \sum_{z_i} P(s_{i+1} \mid s_i, u_i')\ \pi(z_i)\ P(\bar{y}_i \mid z_i)\ P(u_{i+1}')\ ,
\end{aligned}$$

where in the last line, the sum over u_{i+1} has already been performed.

The quantity $P(\bar{y}_i \mid z_i)$ is given by the noise channel, where $\bar{y}_i$ denotes the value of the observed image pixel at position i. $P(u_{i+1})$ is the prior on u_{i+1} (which gets updated upon each TR pass). As mentioned previously, the conditional probability $P(s_{i+1} \mid s_i, u_i)$ is typically deterministic (i.e. equal to zero or one), and this is not the case only when there is more than one FST transition out of state s_i labeled with u_i. In any case, this term ensures that the sum is effectively only over "allowed" (s_i, u_i') pairs which correspond to transitions into s_{i+1} (which is fixed by the left hand side of the equation). Rearranging, the recursive relation for $\pi(Z)$ becomes[8],

$$\pi(z_{i+1}) = P(u_{i+1}') \sum_{\substack{\textit{allowed} \\ (s_i,u_i\text{'})\ \textit{pairs}}} P(s_{i+1} \mid s_i, u_i')\ P(\bar{y}_i \mid z_i)\ \pi(z_i) \tag{4.3}$$

The base case at $i = 0$ can be treated similarly:

$$\begin{aligned}
\pi(z_0) &= \sum_{u_0} P(z_0 \mid u_0)\ \pi_{z_0}(u_0) && (4.4) \\
&= \sum_{u_0} P(s_0)\ \delta(u_0', u_0)\ \pi(u_0) && (4.5) \\
&= P(s_0)\ P(u_0')\ . && (4.6)
\end{aligned}$$

The message pass into the final Z node (at the end of the chain) also differs from the general expression above. However, since the posterior probability of that node does not enter into the calculation of $P(U)$, we ignore it here.

4.2.3 Recursion relation for $\lambda(Z)$

Message passing for $\lambda(Z)$ proceeds from children to parent, as follows.

[8]Note for experts: This relation is essentially the forward recursion for the quantity α which appears in standard treatments of the forward-backward algorithm [88]. The correspondence is even more direct between $\pi(z_i)/P(u_i)$ and α, in which case the expression $P(u_i)P(s_{i+1} \mid s_i, u_i)$ plays the role of the "transition probability" (often denoted a_{ij}) in an HMM. In TR, $P(s_{i+1} \mid s_i, u_i)$ is just a constant (typically 1 or 0), while $P(u_i)$ is updated in each TR pass. Thus to a first approximation, we can say $P(u_i)$ *is* the probability of a transition between states. This would accord with the view of a finite state transducer where the input U field "drives" transitions between states. The (iterative) update of $P(U)$ in turbo recognition then amounts to updating the transition probabilities of an HMM, based on the observed evidence. Nevertheless, we choose to continue to work with the quantity $\pi(z_i)$ (and $\lambda(z_i)$, which up to a factor $P(u_i)$ corresponds to the quantity β for HMMs), in order to maintain the connection to the well-developed graphical model formalism.

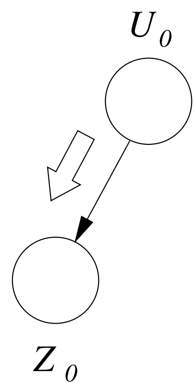


Figure 4.7: The base case at $i = 0$ for π message passing.

$$\lambda(z_i) = \lambda_{y_i}(z_i)\ \lambda_{z_{i+1}}(z_i) \quad ,$$

where the factors on the RHS take the following form,

$$\begin{aligned}
\lambda_{y_i}(z_i) &= P(\bar{y}_i \mid z_i) \\
\lambda_{z_{i+1}}(z_i) &= \sum_{z_{i+1}, u_{i+1}} P(z_{i+1} \mid z_i, u_{i+1})\ \lambda(z_{i+1})\ \pi_{z_{i+1}}(u_{i+1}) \\
&= \sum_{z_{i+1}, u_{i+1}} [P(s_{i+1} \mid s_i, u'_i)\ \delta(u'_{i+1}, u_{i+1})]\ \lambda(z_{i+1})\ \pi(u_{i+1}) \\
&= \sum_{z_{i+1}} P(s_{i+1} \mid s_i, u'_i)\ \lambda(z_{i+1})\ P(u'_{i+1}).
\end{aligned}$$

Therefore,

$$\lambda(z_i) = P(\bar{y}_i \mid z_i) \sum_{z_{i+1}} P(s_{i+1} \mid s_i, u'_i)\ P(u'_{i+1})\ \lambda(z_{i+1}) \tag{4.7}$$

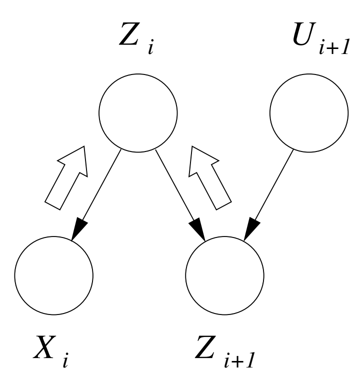


Figure 4.8: Nodes involved in forming $\lambda(Z_i)$.

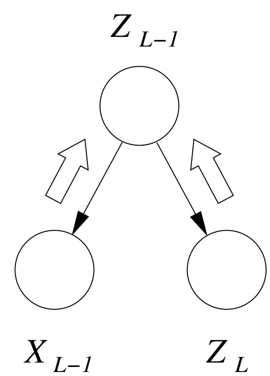


Figure 4.9: Nodes involved in the first message pass for λ at the end of the chain.

The first message pass for $\lambda(Z)$, at the end of the chain, also needs to be treated separately from the general case.

$$\begin{aligned}
\lambda(z_{L-1}) &= \lambda_{y_{L-1}}(z_{L-1})\ \lambda_{z_L}(z_{L-1}) && (4.8)\\
&= P(\bar{y}_{L-1} \mid z_{L-1}) \sum_{z_L} P(z_L \mid z_{L-1})\ \lambda(z_L) && (4.9)
\end{aligned}$$

Note that the last node z_L differs from the other Z nodes in the chain, in that it does not have a U node as a parent, and it ranges only over accepting ("final") states in the FST. I.e., z_L amounts to a bookkeeping device to ensure that the system terminates in an accepting state. Accordingly, $P(z_L \mid z_{L-1})$ can be nonzero only if z_{L-1} (which is fixed on the LHS of the equation) is associated with a transition into an accepting state. In addition, for the FSTs that we have used, a given z_{L-1} corresponds to *at most one* transition into an accepting state. Hence, the sum in Eq. 4.9 reduces to a single term,

$$\lambda(z_{L-1}) = P(\bar{y}_{L-1} \mid z_{L-1})\ \lambda(z_L),$$

when an accepting state z_L corresponding to the pair (s_{L-1}, u_{L-1}) exists, and equals zero otherwise.

4.2.4 The $\lambda_Z(U)$ message

The expression for $\lambda_Z(U)$ (the message from Z_i to U_i) is the following.

$$\begin{aligned}
\lambda_{z_i}(u_i) &= \sum_{z_i, z_{i-1}} P(z_i \mid z_{i-1}, u_i)\ \lambda(z_i)\ \pi_{z_i}(z_{i-1})\\
&= \sum_{z_i, z_{i-1}} P(s_i \mid s_{i-1}, u_{i-1})\ \delta(u_i, u_i')\ \lambda(z_i)\ \pi_{z_i}(z_{i-1})
\end{aligned}$$

Now,

$$\pi_{z_i}(z_{i-1}) = \pi(z_{i-1})\ \lambda_{y_{i-1}}(z_{i-1}) = \pi(z_{i-1})\ P(\bar{y}_{i-1} \mid z_{i-1})$$

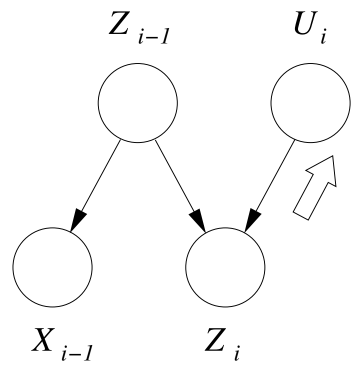


Figure 4.10: Nodes involved in forming the $\lambda_Z(U)$ message.

So,

$$\lambda_{z_i}(u_i) = \sum_{s_i, z_{i-1}} P(s_i \mid s_{i-1}, u_{i-1}) \ \lambda(s_i, u_i) \ \pi(z_{i-1}) \ P(\bar{y}_{i-1} \mid z_{i-1}) \tag{4.10}$$

$$= \sum_{s_i} \sum_{\substack{s_{i-1} \\ u_{i-1}}} P(s_i \mid s_{i-1}, u_{i-1}) \ \lambda(s_i, u_i') \ \pi(z_{i-1}) \ P(\bar{y}_{i-1} \mid z_{i-1}) \tag{4.11}$$

4.2.5 $P(U)$ update

Finally, $P(U)$ is updated via

$$P(u_i) = \pi(u_i)\lambda_{Z_i}(u_i) \ , i = 0, \ldots, L-1$$

where $\pi(u_i)$ is the current value of $P(u_i)$.

4.2.6 Deterministic annealing

For reasons that are described later, we also employ a technique known as deterministic annealing [125] to control the convergence of the TR decoding process. This involves modifying the $P(U)$ update equations to the following form,

$$P(u_i) = \pi(u_i)(\lambda_{Z_i}(u_i))^{\beta} \quad ,$$

in which the $\lambda_{Z_i}(u_i))$ message is raised to a power β before updating $P(U)$. The exponent β initially has a small value such as 0.15 and is modified over the course of the TR run according to some annealing schedule, such as being increased by a constant factor (e.g. 1.4) upon each turbo iteration, which using a physical analogy is akin to lowering the temperature. This has the effect of moderating the tendency of TR to get trapped in ungrammatical configurations, as described in Section 9.4.

4.3 Summary

To summarize, the TR decoding equations take the form of recursion relations for $\pi(Z)$, $\lambda(Z)$, and $\lambda_{Z_i}(U)$, plus update and boundary equations.

$$\begin{aligned}
\pi(z_{i+1}) &= P(u'_{i+1}) \sum_{z_i} P(s_{i+1} \mid z_i)\ P(\bar{y}_i \mid z_i)\ \pi(z_i) && (4.12) \\
\lambda(z_i) &= P(\bar{y}_i \mid z_i) \sum_{z_{i+1}} P(s_{i+1} \mid s_i, u'_i)\ P(u'_{i+1})\ \lambda(z_{i+1}) && (4.13) \\
\lambda_{Z_i}(u'_i) &= \sum_{s_i} \sum_{\substack{s_{i-1} \\ u_{i-1}}} P(s_i \mid s_{i-1}, u_{i-1})\ \lambda(s_i, u'_i)\ \pi(z_{i-1})\ P(\bar{y}_{i-1} \mid z_{i-1}) && (4.14) \\
P^{(new)}(u_i) &= P^{(curr)}(u_i)\ (\lambda_{Z_i}(u_i))^{\beta} && (4.15)
\end{aligned}$$

where in the $P(U)$ update equation, we have made a distinction between the current and updated values of $P(U)$. At the boundaries, we have the following expressions for $\pi(z_0)$ and $\lambda(z_{L-1})$:

$$\begin{aligned}
\pi(z_0) &= P(s_0)\ P(u'_0) && (4.16) \\
\lambda(z_{L-1}) &= P(\bar{y}_{L-1} \mid z_{L-1}) \sum_{z_L} P(z_L \mid z_{L-1})\ \lambda(z_L) && (4.17)
\end{aligned}$$

The following quantities also enter the decoding equations.

- FST transition probability: $P(s_{i+1} \mid s_i, u'_i)$
- noise channel (bit-flip) probability: $P(\bar{y}_i \mid z_i)$

The software programs described in Section 5.2 and available on our website[9] effectively encode Equations 4.12 - 4.17 as the TR algorithm.

[9]UCB Digital Library Project, Source Code, http://elib.cs.berkeley.edu/src/.

Chapter 5

Further details

We complete our introduction to TR with a discussion of some topics that we have left open so far, in particular the issue of actually creating grammars and channels, and some details of our TR implementation.

5.1 Creating grammars and channels

As described in previous chapters, the TR framework describes an image generation process with a pair of grammars and channels, and this leads to a recognition (decoding) algorithm. However, it does not tell us how to create the grammars and channels in the first place. We discuss this issue briefly here.

5.1.1 Grammars

The TR encoding model, which uses independent horizontal and vertical grammars to describe image structure, does not appear to be a very intuitive way to reason about two dimensions. Concepts such as "above," "to the right," "may or may not overlap," etc., do not translate directly into grammars, and a certain amount of modeling experience is required to develop intuition on how to perform such translations

One approach to manually create grammars is to first partition the image plane into regions, based on the primary structure of interest. An example is shown in Figure 3.4 for the case of a rectangle. Once such a partition is formed, it is straightforward to read off the relevant regular expressions, which lead to the FSTs shown in Figure 3.3.

This method works well for layouts where the structures of interest, typically rectangles, are aligned in either the horizontal or vertical direction. This leads in general to compact grammars. As the layout becomes more complex, it becomes more difficult to form a clean partition of the image plane in this manner. This led to the development of "one-layer" grammars, which are introduced in Section 7.1.1. This class of grammars allows the description of more complex layouts. Further classes of grammars will undoubtedly be created in the future.

Since many grammars can describe the same structure, as shown for example in Section 7.1.1, the choice of grammar depends on other factors, such as the size of the grammar (which affects speed of the decoding algorithm), flexibility, convenience, or simply personal preference. The author at present favors the use of one-layer grammars, although these grammars can exhibit a tendency to produce overly complex solutions.

The simplest approach for manually creating grammars is probably by modifying existing grammars. We have created a small library of grammars for this purpose, part of which is described in Chapter 7. This is available online at the UCB Digital Library website [126]. It is certainly conceivable that some aspects of the grammar construction process will be automated in the future, for instance through the use of a tool which manipulates regions within a prototypical page to create grammars by example. The work by Spitz [62] on interactively producing style sheets may be useful for this purpose. While we do not dwell on issues of minimization [110] here, techniques for doing so would undoubtedly be useful from an efficiency standpoint, since the runtime is linearly related to the number of FST states and transitions. This suggests the use of tools such as *lex*. We have found that juggling regular expressions can itself be an onerous task, and sometimes the simplest alternative is to write down the FST graph directly.

5.1.2 Channels

We follow DID tradition in treating channel noise as a straightforward bit-flip process at each pixel. The decoding process has proven to be remarkably robust to such simple channel descriptions of noise. In the OCR context, the actual noise in the image can rarely be characterized as bit-flip noise, yet the performance of DID can be outstanding. In the case of TR, we have also found such channels to be sufficient to successfully decode images with extended noise patterns, as shown for example in Section 9.1.

Geometric distortions such as local affine transformations may be less of an issue in the case of layout analysis, as opposed to OCR, since for example a bounding box can be placed (perhaps approximately) around a given region relatively easily, despite such distortions. The importance of such problems is probably application-dependent. We currently take the channel to be spatially invariant, although we do allow for separate horizontal and vertical channels. We also that images have been deskewed by other methods, although it may be of interest to develop a model of skew within the DID framework in the future.

We currently set the channel parameters manually, by inspection or prior knowledge of the level of noise in the image. An automatic procedure for estimating such parameters is left to future work. This issue is complicated by the fact that we also use the channel itself to model document content, as described in Chapters 8 and 9. Parameter setting in this case proceeds by trial and error, in which TR decoding is run several times with different channel settings on a small sample image, before batch processing of a larger set of similar documents can proceed. Automatic parameter estimation in this case would likely be quite similar to the case of real noise in the image. This step would become less important if other methods of characterizing document content, such as local texture analysis, were to be employed as a preprocessing step.

5.2 Code implementation

5.2.1 Inputs

The primary inputs to our current TR implementation are an observed image file, two grammar files (horizontal and vertical), a channel file, and the desired number of TR iterations. The format of the observed image file is simply a space-delimited ascii array of zeros and ones. Such a file can be created by saving the image in (ascii) portable bitmap (pbm) format, and stripping off the two line header[1]. Examples of the grammar and channel files were given previously (in Section 3.5).

Additional choices to be considered by the user are whether to use maxproduct or sumproduct message passing, the initial value of beta and its annealing schedule, and whether to start each TR iteration with a column pass or a row pass. We recommend using maxproduct message passing, which appears to be more robust than the sumproduct version and gives feedback on the state of the code (and potential bugs!), since the converged output results should satisfy the grammars (except in the case of degenerate maxima or oscillations, both of which occur rarely). We have largely settled on an initial value of 0.15 for beta, which is raised by a factor of 1.4 (i.e., roughly the square root of 2) on each iteration, although it is occasionally useful to experiment with these values. We also have had better success running our TR iterations "column-first." It is somewhat disconcerting at times to find that the output results depend on whether the column pass or row pass is done first in each TR iteration. In retrospect, this is probably related to the orientation of text lines (so that some Asian layouts, for example, may be best processed row-first). Our attempts so far to make our code insensitive to the order of the passes (by performing both vertical and horizontal passes "in parallel" before updating $P(\mathbf{U})$, for example) have not yielded good results[2].

5.2.2 Java code

The code is organized quite simply. The core decoding routines are encapsulated as methods within a `Decoder` object. The `Decoder` includes horizontal and vertical `FST` objects and an `Observed` image object as members. Each `FST` object in turn includes `State[]`, `Transition[]`, and `Channel` objects, as specified by the relevant grammar and channel files. The `Observed` object simply encapsulates the observed image data (i.e., the data to be decoded), together with height and width fields.

As mentioned previously, each TR iteration is composed of a vertical and a horizontal decoding pass. Each pass amounts to performing the forward-backward algorithm over each row or each column in the image. In particular, the "forward" algorithm is run by calling the `getpi()` method, and the "backward" algorithm by the `getlambda()` method.

[1]This format has been sufficient for prototyping purposes. We use a freeware utility called IrfanView [127] to create the requisite pbm files. The use of alternative formats such as tiff has been recently facilitated by the advent of the Java Advanced Imaging (JAI) class library [128].

[2]It appears, roughly speaking, that the decoding algorithm is more effective at producing a solution that is grammatical in *both* directions when each pass is allowed the full benefit of feedback from the orthogonal pass. When the passes are done in parallel, the solution appears to evolve in a way in which neither grammar is satisfied.

Each of these methods assigns values to local data structures (`localpi` and `locallambda`, respectively) where their action depends prominently on `localprobU`, the value of $P(\mathbf{U})$ over the given row or column. The values contained in `localpi` and `locallambda` are then used in `getlambdazu()` to evaluate the `locallambdazu` data structure. The names of these methods and data structures refer, naturally enough, to the quantities $\pi(Z)$, $\lambda(Z)$, and $\lambda_Z(U)$ that appear in the TR decoding equations (see the previous chapter). Finally, the method `updatePU()` is called to update the value of `localprobU`. This (when done for all rows or all columns) constitutes one pass of the TR algorithm

The `main()` driver routine (in TRMain.java) instantiates a `Decoder` using the command line arguments. It also contains the control structures governing the TR iterations (the forward-backward calls), including deterministic annealing (which as mentioned above simply amounts to multiplying `beta` by a constant factor after each iteration).

5.2.3 Output

The code terminates with a converged value of the posterior $P(U)$ as the basic output result. As described elsewhere in this thesis, this is not guaranteed to be the true posterior, although we treat it as such as a practical matter. In particular, we typically perform post-processing on this result, such as choosing the input symbol with the maximum probability at each pixel to form the (approximate) MAP image. This can be output to a file in portable graymap (pgm) format, for easy visualization. This can be considered as the TR decoding output.

Chapter 6

Experimental Comparison with Maximum Likelihood

In this chapter, we consider the question of the optimality of TR decoding. The TR algorithm, which can be viewed as a specialized form of turbo decoding [117], is subject to the same caveats as in the coding theory context. Namely, it is not yet understood in what situations the decoding algorithm produces the optimal result, and there are definitely situations where the optimal result is *not* obtained. Nevertheless, as described in Section 3.1, the overall empirical performance of turbo codes is outstanding, and it is reasonable to expect that such performance will translate to the image recognition context as well. We investigate this question in detail in this chapter.

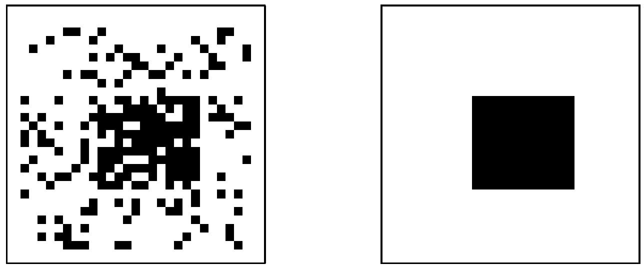
We make the issue concrete by considering the following problem: Given a noisy image such as Figure 6.1a, find the single (filled) rectangle such as Figure 6.1b which is most likely to have produced it, assuming a bit-flip noise process (this is the reverse of Figure 3.6). We can term this the "one-rectangle" problem. It is a simple matter to frame this as a TR decoding problem using one-rectangle grammars, and it also has the feature that the optimal estimate can be found independently in a straightforward manner.

In one dimension, the corresponding "one-interval" problem is, given a noisy bit sequence such as 001101101000, find the single interval of ones which best "explains" the data, such as 001111111000[1]. This problem is a straightforward application of hidden Markov models. In particular, using dynamic programming approach such as the Viterbi algorithm, we can efficiently recover the sequence which has minimum probability of error (the MAP estimate).

Unfortunately, there does not appear to be a correspondingly efficient dynamic programming solution to the one-rectangle problem.[2] This is further evidence that two

[1] This example is somewhat unusual in having two possible optimal solutions, the other one being 001111100000 (under a symmetric bit-flip noise model). This is an example of degenerate maxima, which occasionally plagues TR also. As an aside, such cases are one of the few situations where TR maxproduct message passing does *not* necessarily converge to a grammatical result (i.e., one that satisfies both the horizontal and vertical grammar simultaneously). The analog of this in the one-interval case might be a solution with two intervals of ones, even though the grammar only specifies one.

[2] A dynamic programming approach can be defined, but this requires an exponential explosion in the size of the state space.



(a) (b)

Figure 6.1: (a) Noisy observed image; (b) proposed source image which produced it.

dimensions is in some sense inherently different from one dimension. This is related to the fact that there is no natural notion of "before" and "after" (i.e. linear ordering) of pixels in two dimensions.

Thus even for this simple two-dimensional problem, we are reduced to doing an exhaustive search in order to be guaranteed of recovering the optimal (maximum likelihood) solution. This scales as N^2, where N is the number of pixels in the image.[3] In contrast, since the heart of the TR algorithm is a collection of 1D dynamic programming problems, it scales linearly in the number of pixels in the image (times the number of iterations). We thus investigate the optimality of TR decoding for this problem by comparing it with the maximum likelihood (ML) estimate obtained by exhaustive search, which is equivalent to the MAP estimate when the prior on U is uniform, which is what we will use.

The experiment in more detail goes as follows. The original source image is the 27 by 27 bitmap shown in Figure 6.1b. This is corrupted with symmetric bit-flip noise at noise levels (bit-flip probabilities) ranging from 0.1 to 0.3, in increments of 0.05, with 500 samples produced at each noise level. The ML estimates are found by an exhaustive search over all possible sizes and locations of the filled rectangle. The TR solutions were obtained using seven iterations of maxproduct message passing, done column-first (i.e. each iteration consists of a column pass followed by a row pass). The annealing schedule starts with $\beta = 0.15$, which is raised by a factor 1.2 on each iteration (this is slightly lower than our usual value of 1.4). The channel is tuned to the noise level (i.e. we presume to know the noise level beforehand)[4].

The results for a range of noise levels from $p = 0$ to 0.3 are summarized in Figure 6.2. This shows two curves that describe the probabilities that the TR and ML estimates each match the original image. The two curves are difficult to distinguish at this resolution, and we conclude that TR and ML are essentially equivalent in their ability to recover the original image.

The extent to which the TR solutions match the corresponding ML estimates for

[3]The complexity is given by the number of rectangles that can be defined within an image of size N. Each rectangle can be specified by its top-left and bottom-right corner, each of which has N possible values (up to double counting), and hence the number of rectangles scales as N^2

[4]This is fair in the present case, where the aim is to test the ability of TR to find the optimal solution for a given model. In general, a preprocessing step could be employed beforehand to estimate the noise level.

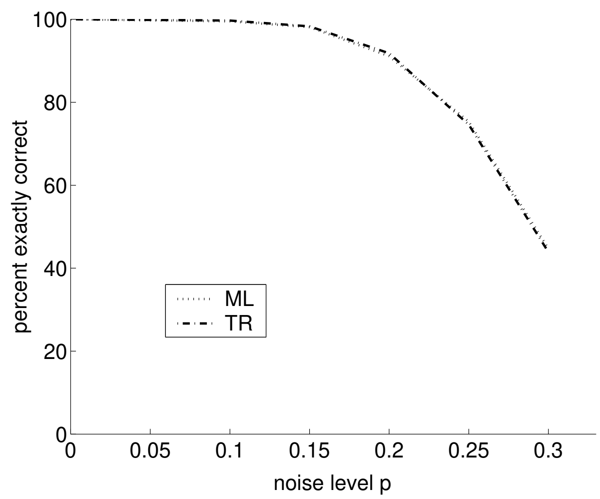


Figure 6.2: Recognition results for the one-rectangle problem. The dotted and dash-dotted lines, which nearly overlay each other, describe the probabilities with which the ML and TR solutions, respectively, match the original clean image.

each image is summarized in Figure 6.3. If the TR results were always finding the optimal estimate, we would expect the curve to remain at 1.0 for all noise levels. It is possible that TR is finding the optimal estimate at a rate better than is implied by this curve, if there are in fact several degenerate maxima, which is quite likely. In any case, the match between TR and ML begins to degrade at a noise level where ML itself starts to fail significantly in recovering the original image, so the question of how well TR matches ML is in some sense moot from a practical point of view.

Future work includes analyzing the cases where TR and ML fail to match. Recent work by Freeman and Weiss [119], for instance, shows that maxproduct message passing (also known as belief revision [118]) in loopy graphical models such as TR produces solutions that are optimal over an extended neighborhood of that solution.

For this small experimental image, TR is about twice as fast as ML exhaustive search, taking about 0.8 seconds per image (with five turbo iterations each). The difference in complexity scaling rapidly makes the speed differential more significant when applied to larger or more complex images.

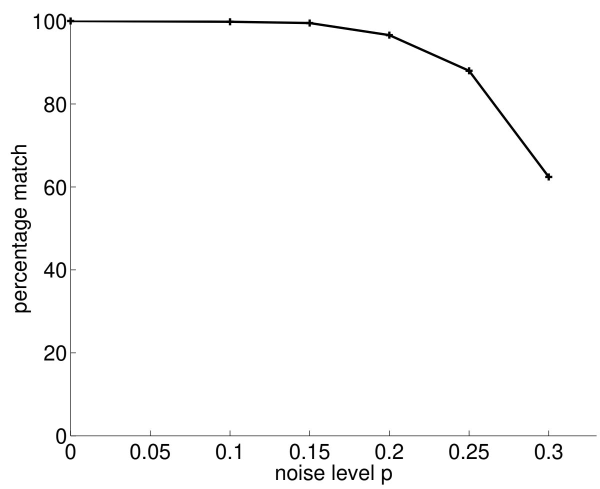


Figure 6.3: The solid line describes the probabilities with which the TR solutions match the ML solutions.

Chapter 7

The Art of Turbo Recognition

One of the strengths of TR is its foundation on statistical principles. This by itself, however, does not guarantee good performance and in particular is not sufficient to build a working system. The TR model degrees of freedom, primarily the horizontal and vertical grammars and the noise channels, need to be specified. This is both a strength, in that it allows the TR decoding approach to be tailored to a wide variety of tasks, and a potential weakness, in that such models need to be specified beforehand. In essence, TR provides a language for describing layout, but the best ways to speak this language are not yet clear. In this chapter we begin the investigation of the *art* of TR.

There are a number of considerations which enter into selecting a grammar. A primary consideration is to choose one that can decode the layout structures of interest. Another is that several grammars may describe the same physical structure, such as a single rectangle, or one grammar may generalize several others. Indeed, simpler grammars (such as the "blob" grammars described later) that supply only weak constraints on image structure could be used in place of a large class of other grammars that are more restrictive[1]. The choice of grammar must then be based on other considerations, such as noise-robustness, speed, or ease of post-processing. For example, grammars tuned to the class of structures of interest tend to produce results which are more robust against noise.

Here we explore the space of TR grammars with the aim of better understanding how expressive these grammars can be. We are thus primarily concerned with the ability of TR to describe abstract shapes, or more generally, image structure. In later chapters, we will turn to applications more directly related to the analysis of document images.

At present, we are exploring the space of grammars by constructing models by hand. We note that, in principle, TR grammars could be specified using some externally defined standard, e.g. by an SGML document type definition[2], or they could be learned, by training on a database of documents with similar structure and noise characteristics. Automatic grammar *selection* is also of interest, in which for example a number of different grammars are applied to an image, and a refinement of the one that fits best is used to

[1]This is akin (though clearly not identical) to saying that a regular grammar can be constructed whose language forms a superset of the language for any given context-free grammar.

[2]This would probably need to be augmented by description of the physical, as opposed to just logical, structure of the page, e.g. in a language such as DSSSL

actually process the page image. Such topics will undoubtedly be the subject of future research. Our current work provides the foundations for a toolkit of grammars that can be referenced according to the needs of specific applications, and we hope it will help inform the future efforts of researchers and users.

At present, the choice of channel is made on an ad hoc basis. This is typically a matter of choosing four or six parameters, the total number being equal to the number of output symbols times the number of observed symbols. Since these parameters represent bit-flip probabilities, this actually amounts to choosing only two or three numbers. We set these manually according to the percentage of black pixels per unit area ("black pixel density") that we expect to see in the regions corresponding to each output symbol. Further details are given below.

7.1 An exploration of TR grammars

In this section we begin exploring the space of structures that can be parsed by TR, starting with simple cases and growing more complex. This can be thought of as progressively making the TR generative model more expressive. The fact that TR models are based on regular grammars defines a fundamental limit on their expressiveness. Nevertheless, there is much room to explore, and we do not claim to have exhausted the design space by any means. As mentioned earlier, we focus here on the description of ideal shapes, with the expectation that such work can form the core of later work in both physical and logical layout analysis. In particular, we defer questions about the noise robustness of such descriptions to later chapters.

7.1.1 One rectangle

As mentioned in Section 3.3, the following grammars can be used to describe a rectangle:

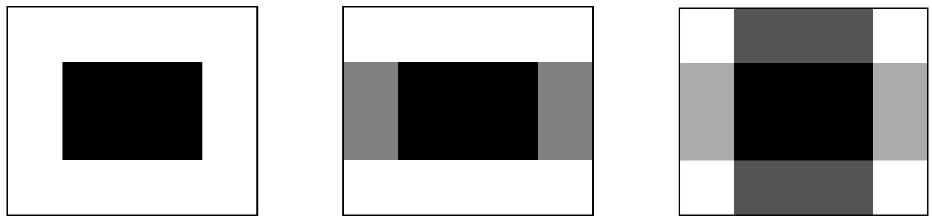
$$\begin{array}{rl} \textit{horizontal}: & a^+ \mid b^+c^+b^+ \\ \textit{vertical}: & a^+(b^+ \mid \; c^+)a^+ \quad . \end{array}$$

When both a and b are mapped to white (i.e., a bitmap value of 0) and c is mapped to black (i.e., a bitmap value of 1), these grammars describe a single black rectangle on a white background. Given this grammar, TR parses the image shown in Figure 7.1a with the partition given in Figure 7.1b. For historical reasons, we will refer to this as the "standard" one-rectangle grammar.

This manner of describing a rectangle is by no means unique. For example, a grammar such as the following,

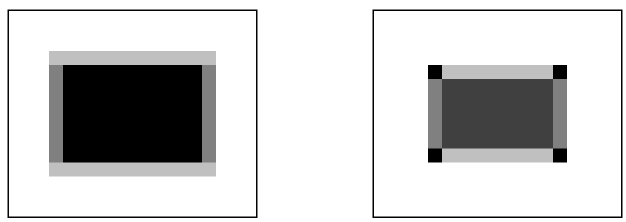
$$\begin{array}{rl} \textit{horizontal}: & a^+b^+a^+ \mid c^+d^+c^+ \\ \textit{vertical}: & a^+c^+a^+ \mid b^+d^+b^+ \quad , \end{array}$$

where a, b, and c are background symbols, and d is the sole foreground symbol, is aesthetically pleasing because, unlike the previous grammar, it treats the horizontal and vertical directions on an equal footing. The partition of the canvas induced by this grammar is illustrated in Figure 7.1c.



(a) (b) (c)

Figure 7.1: (a) Image containing a single black rectangle; (b) the image partition induced by the standard one-rectangle grammar; (c) the partition induced by the symmetric version of the grammar.



(a) (b)

Figure 7.2: One-layer descriptions of a rectangle: (a) asymmetric version using four symbols; (b) symmetric version using five symbols.

For both cases above, information about the location of the rectangle is propagated far away from the rectangle itself, namely out to the edges of the image canvas. We will call grammars with this characteristic "nonlocal" grammars. Such grammars often are easy to design and use a relatively small number of input symbols.

While this nonlocal character is sometimes useful (for example for postprocessing or visualization purposes), it can also become a significant nuisance when trying to design grammars for more complex structures. The following *one-layer* grammar gets around this problem:

$$\begin{array}{rl} horizontal: & a^+ \mid (\ a^+(b^+ \mid\ \ c\ d^+c\ ))^+a^+ \\ vertical: & a^+ \mid (\ a^+b\ (c^+ \mid d^+)\ b\ )^+a^+ \end{array} \quad ,$$

where a represents the background that extends (figuratively) to infinity, d represents the foreground (the black rectangle), and b and c form a "protective layer" one pixel thick which surrounds the rectangle, as shown in Figure 7.2a. The role of b and c above is quite similar to that of a and b in the standard one-rectangle grammar. The fact that the symbols that enforce rectangularity are localized to the immediate vicinity of the rectangle will prove to be useful feature.

There is again a choice in the design of one-layer grammars, which mirrors that for the standard grammars. The partition induced by a symmetric one-layer grammar, in analogy with Figure 7.1c, is shown in Figure 7.2b. This also exhibits a version of the grammar where the layer symbols correspond to printing (black) output, so that the layer is included as part of the rectangle, rather than wrapping around the outside of it.

7.1.2 Generalizations using nonlocal grammars

The standard one-rectangle grammar can be generalized to a grid structure with an arbitrary number of columns and/ or rows with the addition of a few Kleene + symbols:

$$\begin{array}{rc} \textit{horizontal}: & a^+ \mid (\, b^+c^+)^+b^+ \\ \textit{vertical}: & (a^+b^+)^+a^+ \mid (a^+c^+)^+a^+ \end{array}$$

This describes rectangular blocks whose edges are perfectly aligned, both horizontally and vertically. Variations on this theme, where some cells in the grid are blank, for example, are easily produced.

The alignment between blocks across rows can be relaxed using a generic "multi-row" grammar:

$$\begin{array}{rl} \textit{horizontal}: & a^+ \mid (\, b^+c^+)^+b^+ \\ \textit{vertical}: & (a^+(b^+ \mid c^+))^+a^+ \end{array}$$

In the horizontal grammar, the strict separation between a^+ and $(\, b^+c^+)^+b^+$ implies that the rectangles in the image form into rows, with their top and bottom edges aligned. However, the vertical grammar, in comparison to the previous version, now allows runs of b^+ and c^+ (separated by runs of a^+) to be freely mixed, implying that the vertical edges of the rectangles in the image need not be aligned.

7.1.3 Generalizations using one-layer grammars

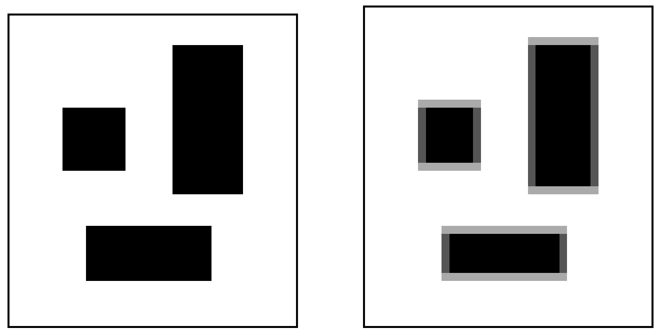
When alignment between blocks is not pertinent at all, it is convenient to switch to the above-mentioned one-layer description of rectangles. This allows the rectangles to be freely aligned relative to each other, as shown in Figure 7.3. If a nonlocal grammar were used in such a situation, the regions extending to the edges of the image (see Figure 7.1) would interfere with each other, requiring the use of more input symbols and regular expressions to handle the different possible alignments. Such an approach would not be scalable to a large number of rectangles with arbitrary alignments.

A general Manhattan layout can be described using one-layer grammars with the addition of an input symbol that denotes the corner points. An application of such a grammar is shown in Figure 7.4. We note in particular that this layout cannot be analyzed using XY cuts (cf. Section 1.2.5 and Figure 1.2). The input alphabet in this case has five symbols, and the horizontal and vertical transducers each have eight states and fifteen transitions (Figure 7.5).

7.1.4 Blob grammars

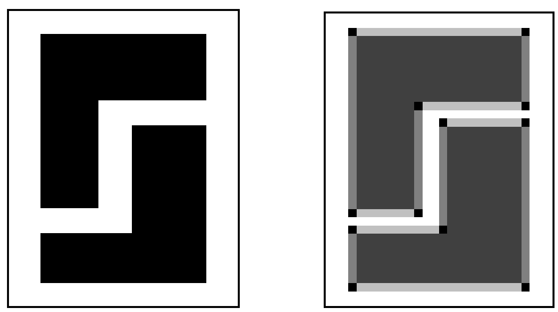
While we have emphasized the ability of TR to parse Manhattan layouts, it can be applied to more general layouts as well. Consider for example the following "single blob" grammar:

$$\begin{array}{rl} \textit{horizontal}: & a^+ \mid \; a^+b^+a^+ \\ \textit{vertical}: & a^+ \mid \; a^+b^+a^+ \;. \end{array}$$



(a) (b)

Figure 7.3: (a) Freely aligned rectangles; (b) TR decoding using a one-layer grammar.



(a) (b)

Figure 7.4: (a) Original image with Manhattan layout; (b) TR decoding using a one-layer Manhattan grammar.

The horizontal and vertical grammars are identical, and describe a run of a's which is interrupted by at most one run of b's. If we associate a with background (0) and b with foreground (1), we see that this grammar describes an arbitrary shape, subject only to the one-interval constraint (in both directions). Note that this grammar is very similar to the standard one-rectangle one, except we have now eliminated the seemingly redundant background symbol (called b in that case) that was required to enforce rectangularity. This grammar is easily generalized to multiple blobs in the usual manner, by the addition of a Kleene + symbol:

$$\begin{aligned} horizontal: &\quad a^+ \mid (\ a^+b^+)^+a^+ \\ vertical: &\quad a^+ \mid (\ a^+b^+)^+a^+\ . \end{aligned}$$

We will call this a "blobs" grammar.

We demonstrate the character of the solutions produced with these blob grammars by applying them to the arch-shaped blob shown in Figure 7.6a. Using a blobs grammar, we get the result shown in Figure 7.6b, which in fact is identical to Figure 7.6a, as one

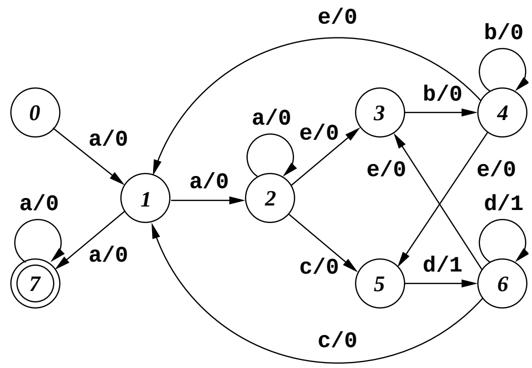
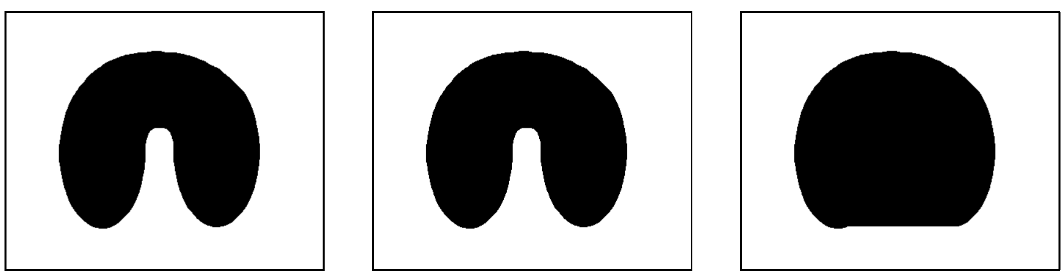


Figure 7.5: Horizontal FST used in decoding Figure 7.4. The vertical FST is identical with a switch of the b and c symbols.

might expect, given the flexibility of the grammar. Using a single-blob grammar yields the result shown in Figure 7.6c. The decoder output now satisfies the constraint that there be at most one interval of black (i.e. of the input symbol b) in either direction. As an aside, careful viewers may notice that the left lobe protrudes slightly from the flat bottom along most of the blob, implying that the left lobe of the original image is slightly lower than the right.



(a) (b) (c)

Figure 7.6: (a) A two-lobed blob; (b) TR decoding using a multi-blob grammar; (c) TR decoding using a single blob grammar.

7.2 Texture segmentation

Up to this point, we have been concerned with the flexibility offered by the choice of grammar. This section now considers a way of exploiting the channel for modeling purposes. To recap, TR views the image generation process as having two stages, in which a source message is first encoded into an ideal output image and then transmitted through a noisy channel to produce an observed image. Such a description suggests that the output and observed image pixels are derived from the same (e.g. binary) alphabet, but this need not be the case. In addition, rather than viewing the transmission process as simply corrupting the image, we can consider it to be an aspect of the modeling process in general, even for clean images.

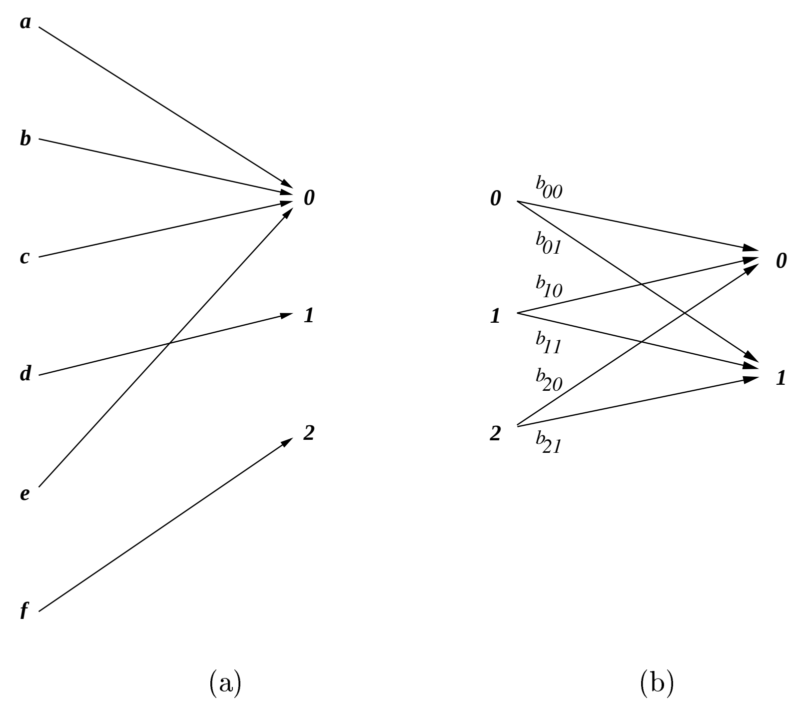


Figure 7.7: Image generation model used in decoding Figure 7.8, which utilizes more output symbols to allow a simple form of texture discrimination. (a) Input to output symbol translation; (b) stochastic mapping from output to observed symbols.

Figure 7.7 gives an example of an encoding and channel model which maps from six input symbols to three output symbols, and then to two observed symbols[3]. Each output symbol represents a zone with its own characteristic density of black pixels. We can apply such a model to decode an image such as Figure 7.8 (a). Here the observed image (which can be considered uncorrupted) has three types of regions, which we term "background," "sparse black," and "dense black." The sparse and dense regions were created by applying 20% and 80% bit-flip noise, respectively, to a blank rectangle. This image can be classified into zones using the following TR model. The grammar is a one-layer grammar which describes an arbitrary number of rectangles, which has been modified to so that both d and f are "foreground" symbols:

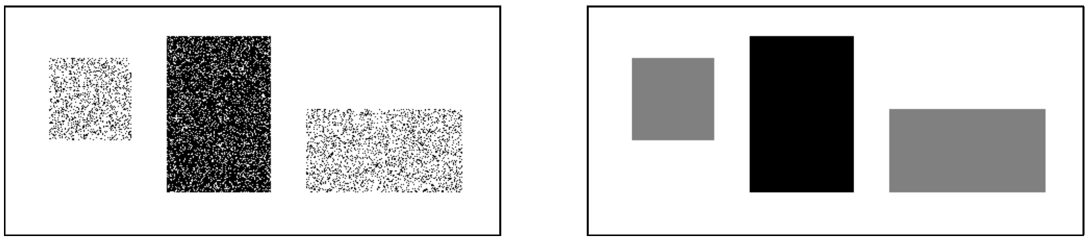
$$a^+|(a^+(e\ b^+\ e)|(c\ d^+\ c)|(c\ f^+\ c))^+a^+$$

horizontally, and

$$a^+|a^+((e\ c^+\ e)|(b\ d^+\ b)|(b\ f^+\ b)) + a^+$$

[3]In this thesis, we limit ourselves to binary (bitonal) observed images, but the number of output symbols can be arbitrarily large, depending on the application.

vertically. The channel parameters in this case are $\{\{0.9990.001\},\{0.80.2\},\{0.10.9\}\}$. The resulting TR decoding in Figure 7.8. This can be viewed as a simple form of texture discrimination.



(a) (b)

Figure 7.8: (a) Image having zones with two different black pixel densities; (b) TR decoding result with a rectangular grammar that uses three output symbols to represent background and two different foreground textures, as described in Figure 7.7. White, gray, and black correspond to the input symbols a, d and f, respectively. The b, c, and e symbols are not shown, and would form a one-pixel wide boundary layer around each rectangle, similar to that shown in Figure 7.4.

Chapter 8

Applications of TR to text lines

In order to develop TR as a page layout analysis method, we consider text images as an important first application. As mentioned previously, it is possible for humans to recognize the layout of a page, and recognize that certain zones within a page contain text, without necessarily being able to recognize individual characters. This implies that it is fruitful to consider page layout analysis directly as a shape recognition task, which abstracts away from the level of individual characters. Nevertheless, as the reader may have noted in Chapter 3, it is unclear exactly in what sense that we may treat a text column as a single black rectangle, for example. In this section, we investigate this issue and develop some methods for bridging this gap in abstraction.

Before doing so, we step back for a moment to take a more global perspective. TR is one point in a spectrum of DID methods, and we have treated it as largely divorced from previous DID techniques for the purposes of this thesis. However, a more elegant (if more compute-intensive) solution to the problems of text modeling discussed below would be to combine TR with a DID OCR model having trained character templates. This would create an integrated probability model for both layout and character recognition. Relative to such a grand model, what follows is a "poor man's" approach to dealing with text in the TR framework. This has the advantage of being relatively quick and easy to use. A variety of layers of sophistication could be added, however, and it would be of interest to do so in the future.

On a more practical note, an important issue for present DID OCR decoders is efficiently locating text on the page. Previous work by Kam and Kopec [129] integrated text line finding into the character decoding process (in a manner akin to what is being advocated in the previous paragraph) by using separable models and *admissible* heuristics to speed up decoding. This approach, however, is limited to a single column of text and is somewhat heavyweight in its formulation, requiring training of text line projections profiles, for example. In practice, this has resulted in the reliance on non-DID methods to find baselines or extract columns. Finding text line or text block regions (and the relations between them) is also of interest in and of itself, and it would valuable to have methods addressed to such tasks without requiring a full OCR decoding to be done at the same time. The following can be viewed as the beginnings of an effort to develop such methods within the framework of DID.

8.1 A single text line

We begin our investigation by considering a single text line. We can illustrate many features of our current TR approach by using such simple examples. While one of the main advantages of TR is its ability to deal with noise, in the next few sections we will be primarily concerned with its behavior on clean images.

Figure 8.1a shows a single text line imaged at 100 dpi (roughly 400 by 70 pixels)[1]. We consider decoding this image using the standard one-rectangle grammar, $a^+ \mid b^+c^+b^+$ horizontally and $a^+(b^+ \mid c^+)a^+$ vertically, where again a and b are considered background symbols and c is a foreground symbol. It is quite evident that a text line is not simply a black rectangle. In order to proceed, we view the text line as having arisen from a single filled rectangle, which upon transmission through the channel became "corrupted" to yield the rather sparse arrangement of black pixels forming the text line. In other words, we treat the text as bit-flip noise. It may strike the reader as unusual to model what is presumably the "signal" (the text) as "noise," and in any case it hardly needs to be said that this is a very impoverished model of text. Nevertheless, bit-flip noise models have proven to be quite useful in the DID OCR context [95], and our work here can be taken as further evidence of the mileage that can be gained even from such simple models[2].

The overall image generation process can be sketched out as follows. A symbol field in an input alphabet $\{a, b, c\}$ is translated by the FSTs (grammars) into a field in the output alphabet $\{0', 1'\}$, where we can call $0'$ the background symbol and $1'$ the foreground symbol (where we now use primes to distinguish the output from the observed alphabet). This output field is mapped stochastically (independently at each pixel) by the channel to an observed field in the alphabet $\{0, 1\}$, which represents the actual white and black pixels in the image we are decoding.

We first consider decoding the image with the following channel[3],

$$\begin{array}{ll} 0.99 & 0.01 \\ 0.5 & 0.5 \end{array} .$$

The top row of channel parameters says that the output $0'$ symbol (and by extension a and b) are very likely to produce an observed 0 (white pixel), while the bottom row says that the output $1'$ symbol (and hence c) has a 50-50 chance of producing either an observed 0 or 1 (white or black pixel). We will denote such channels using the linearized notation $\{\{0.99, 0.01\}, \{0.5, 0.5\}\}$ from now on.

The resultant TR decoding obtained after three iterations using maxproduct message passing is shown in Figure 8.1b. The black region associated with c finds the body of the text line, while cutting off both ascenders and descenders. Colorfully speaking, TR is trying to surround a high concentration of black pixels with a single rectangle, while leaving as few black pixels out as possible. This "clumping" behavior is often a useful way to understand TR output.

---

[1] This figure was generated electronically using TeX, dvips, and finally ghostscript to produce a pbm file.

[2] More sophisticated alternatives that come to mind include using a set of texture features to model the probability that a given local region represents text, and detailed DID character template models as mentioned above.

[3] We choose the horizontal and vertical channels to be identical in what follows.

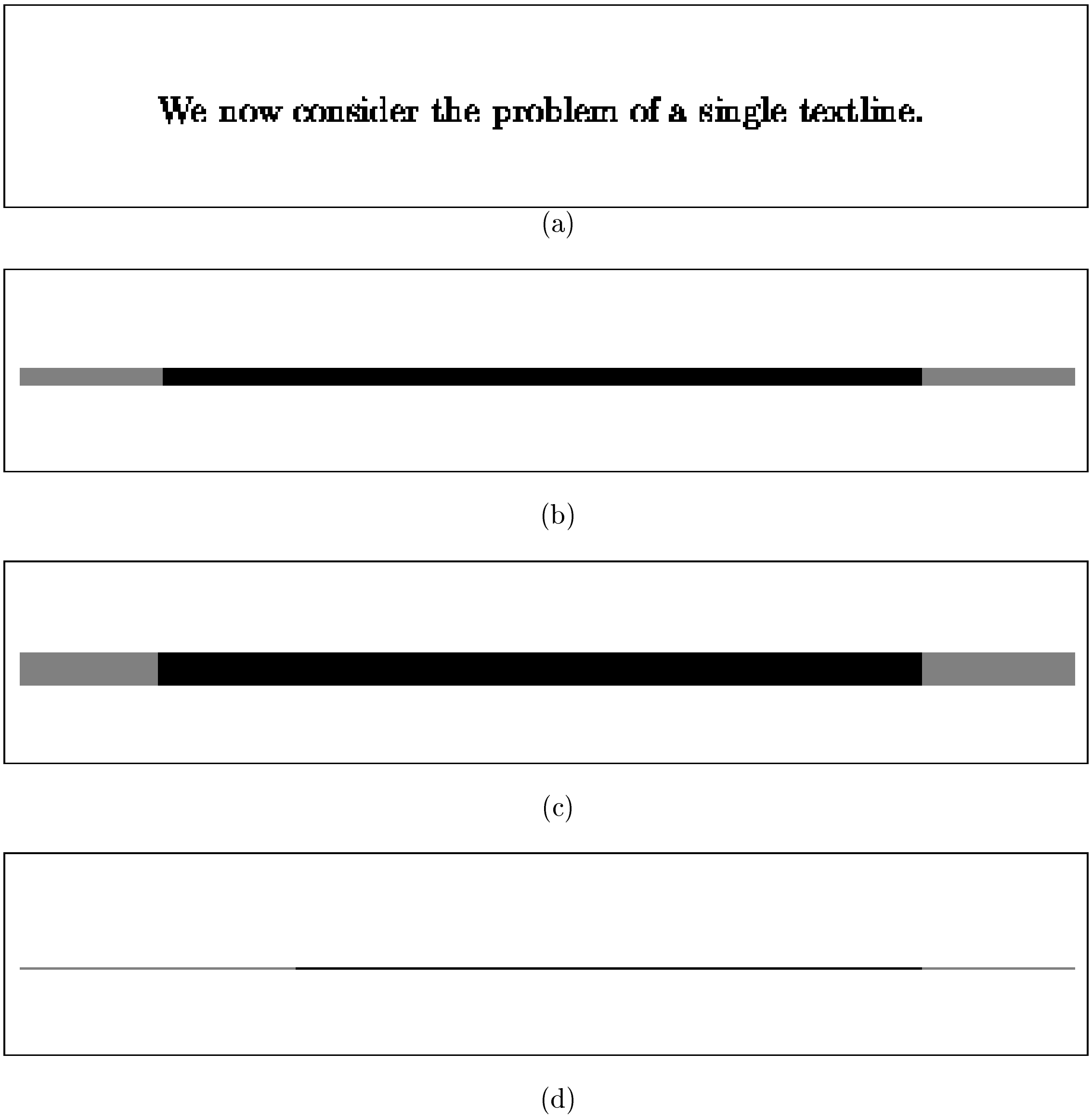


Figure 8.1: (a) Image with one text line at 100dpi. (b) Sample TR decoding with a standard one-rectangle grammar, where white = a, grey = b, and black = c. (c) TR decoding using a "brighter" channel in which foreground output pixels are mapped with high noise onto the output pixels; (d) TR decoding using a "darker" channel in which foreground pixels are required to nearly always be black.

For Figure 8.1c, the channel has been skewed strongly towards producing white (0) observed symbol, $\{\{0.999, 0.001\}, \{0.99, 0.01\}\}$. This allows TR to be more generous in what it includes within the foreground region. The rectangle returned by TR now covers most of the text pixels, though it does not quite extend to the full height and depth of the text line.

Figure 8.1d illustrates the effects of using a "darker" channel, $\{\{0.95, 0.05\}, \{0.05, 0.95\}\}$, on this example. This channel requires the foreground ($1'$) symbol to appear only in regions with very high black (1) pixel density. Indeed, if the grammar permitted it, a solution with no foreground pixels at all (an "empty rectangle") might be optimal, since in general the black pixel density of text is quite low. However, the grammar forces TR to place a rectangle somewhere, and Figure 8.1c shows that its choice is a rectangle only 1 pixel high. This happens to be a reasonable estimate of the text baseline. This is probably a consequence of the presence of serifs in the Times-Roman font, and it is hence unlikely that this approach would produce a robust baseline finder. It is nevertheless indicative of the kind of freedom the user of TR has to design the algorithm to take advantage of features of interest.

In summary, this example shows that the TR apparatus as presented so far is sufficient to segment a line of text from background, using a one-rectangle grammar and the simple artifice of treating text as bit-flip noise. The exact form of the TR output shows some sensitivity to the channel parameters, as might be expected with this model. In a practical application, the result shown in Figure 8.1b would perhaps be the most useful, since it focuses on the body of the text line (with the ascenders and descenders stripped off), which is a relatively robust feature. Measurements such as font size and baseline position could then be extracted. While the extent to which such features can be reliably found has yet to be investigated, the ability of TR to provide simple summaries of "where the text is" may be one of its most useful features.

8.2 Multiple text lines

We now consider a straightforward generalization of the previous example, namely a text block containing several text lines, as shown in Figure 8.2. The image resolution for this example is again 100 dpi[4]. We decode this image using a simple modification of the one-rectangle grammar. The horizontal grammar remains the same, $a^+ \mid\ b^+c^+b^+$, and allows at most one interval of black (c) pixels. The vertical grammar is modified to allow multiple occurrences of b or c, $((a^+b^+)^+ \mid (a^+c^+)^+)a^+$. The grammar does not allow b and c to be mixed within a column, however, which forces the left and right ends of all decoded rectangles to be the same.

Figure 8.2b shows the TR result using this grammar with the following channel, $\{\{0.95, 0.05\}, \{0.5, 0.5\}\}$ As advertised, the left and right edges of all of the rectangles are aligned. In most respects, we can characterize the individual rectangles in this TR decoding as simply repetitions of the one found in the single text line example (Figure 8.1b in particular). However, the first and last text lines in the original image are clearly shorter than the others, so it may be objectionable to represent these with rectangles that extend

[4]As the reader may have surmised, the (spurious) difference in size of the text glyphs, for example, in Figure 8.1 and 8.2 is due simply to the fact that they are scaled differently.

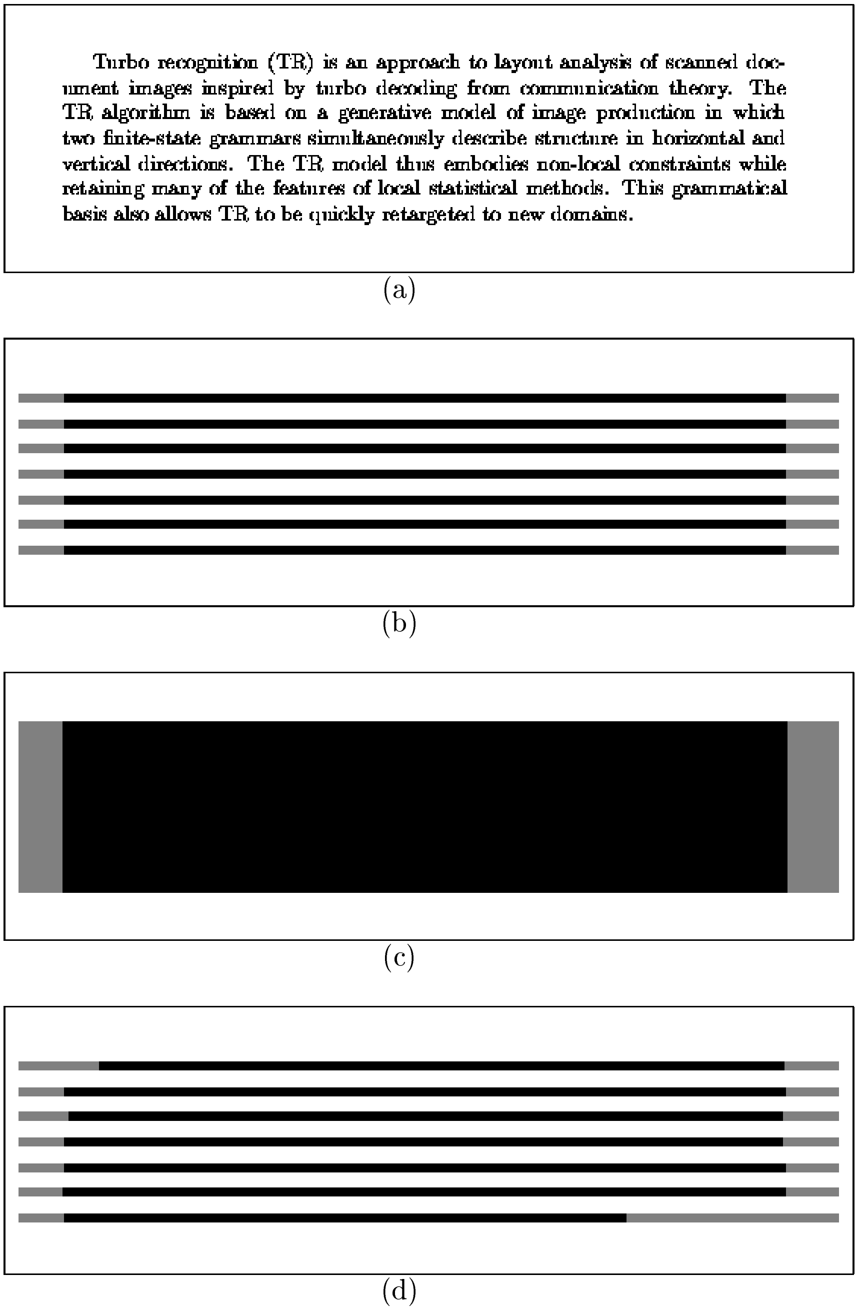


Figure 8.2: (a) Text block imaged at 100 dpi; (b) TR decoding using a multi-line grammar, where the left and right edges are aligned; (c) TR decoding using a different channel which recovers the full text block. (d) TR decoding using a multi-line grammar with no constraint on the left and right edges of each rectangle.

over the full width of the text block. We remedy this with a slightly different grammar below.

Figure 8.2c shows that the text block as a whole can be recovered as a single rectangle, even though we are using a multi-line grammar[5]. This results from using a very "bright" channel that models a sparse density of black pixels, $\{\{0.999, 0.001\}, \{0.99, 0.01\}\}$ We thus can focus on different levels of the document layout hierarchy by choosing different channel parameters.

To provide a more faithful representation of "ragged" text lines, we modify the vertical grammar to allow b and c to be mixed within a column: $(a^+(b^+ \mid c^+))^+a^+$. The resulting TR decoding is shown in Figure 8.2d, where the channel is $\{\{0.95, 0.05\}, \{0.5, 0.5\}\}$

As a brief diversion from our investigation of TR on clean images, we consider the noisy image shown in Figure 8.3a, in which 20% bit-flip noise has been added to Figure 8.2a. Using a standard $\{\{0.95, 0.05\}, \{0.5, 0.5\}\}$ channel, we get the result shown in Figure 8.3b. If we use knowledge of the amount of noise in the image (which could be estimated in some way), we get the result shown in 8.3c. The channel in this case is $\{\{0.8\,0.2\}, \{0.5\,0.5\}$. This result is nearly indistinguishable from the TR result on the clean image (Figure 8.2). Simple text line finding remains a nontrivial problem in DIA, and behavior of this sort could be useful, e.g., as a preprocessing step in script identification [130], especially in the presence of noise.

8.3 Word boxing

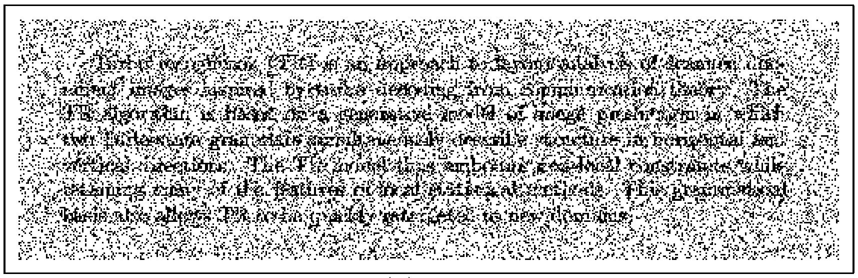
We now consider going one step further down the layout hierarchy, to find individual words. This task has a variety of applications, one being the "enlivening" of scanned documents in a multivalent document browser [131]. The modification required to the grammar for this task is again a simple one. We now allow an arbitrary number of black intervals (c^+) within a row: $a^+ \mid (b^+c^+)^+b^+$. The vertical grammar is the same as the one used in the previous section, $(a^+(b^+ \mid c^+))^+a^+$, which does *not* require rectangles in different rows to be aligned with each other.

For convenience, we show the original text block image in Figure 8.4a. The TR decoding on this image with the above grammar and a sparse channel $\{\{0.999, 0.001\}, \{0.9, 0.1\}\}$ is shown in Figure 8.4b. There is a unique rectangle for every word in the text block, and naturally each text line is segmented out at the same time. There are some additional features, such as "splinter" lines which correspond to character descenders, and separate boxes for three out of the four periods at the end of a sentence. An application of this technique to word boxing would thus have to deal with such noisy (from the point of view of word boxing) results[6].

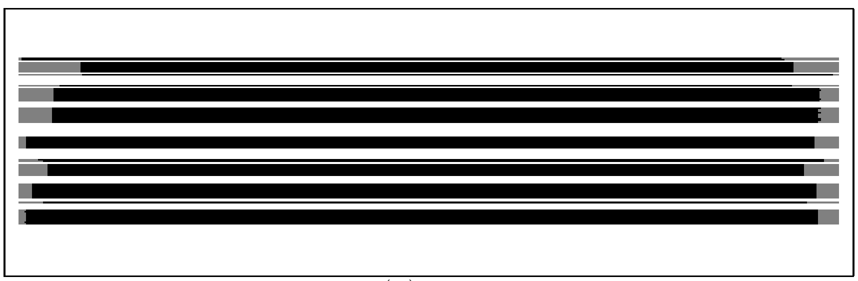
Recovering word boxes in the presence of noise is more difficult. Figure 8.5b shows the TR decoding of the noisy text block image given previously (shown again in Figure 8.5a for convenience), using the above grammar and a channel which has been "dialed down" carefully to take account of the noise, $\{\{0.7, 0.3\}, \{0.6, 0.4\}\}$. Some word boxes span two words, and others split a single word. Nevertheless, this result can be considered reasonably

[5] A one-rectangle grammar would also serve well for this purpose.

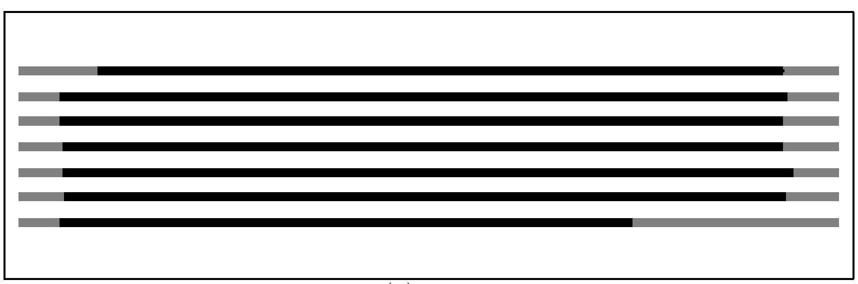
[6] This seems to not pose much difficulty, though we have not investigated the problem in detail.



(a)

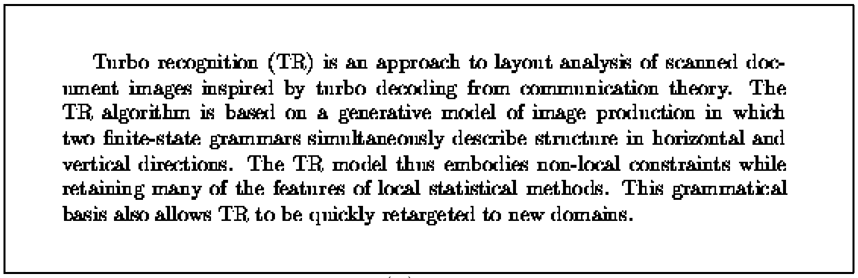


(b)



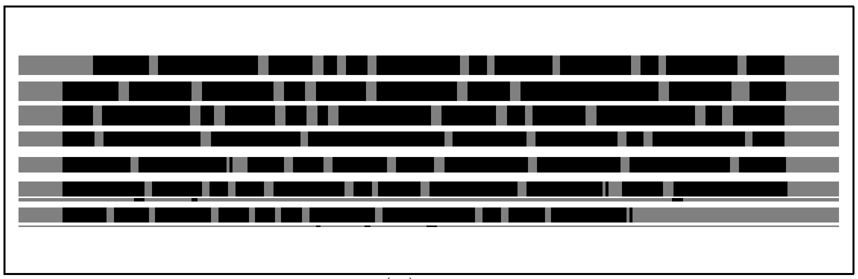
(c)

Figure 8.3: (a) Text block image corrupted with 20 % noise; (b) TR decoding using a channel which is not a good match to the image; (c) TR decoding using a better channel.



Turbo recognition (TR) is an approach to layout analysis of scanned document images inspired by turbo decoding from communication theory. The TR algorithm is based on a generative model of image production in which two finite-state grammars simultaneously describe structure in horizontal and vertical directions. The TR model thus embodies non-local constraints while retaining many of the features of local statistical methods. This grammatical basis also allows TR to be quickly retargeted to new domains.

(a)



(b)

Figure 8.4: (a) Text block image (repeated for convenience); (b) TR decoding using a grammar and channel which approximates word boxing.

good, given the rather extreme noise conditions. In addition, the text lines themselves have all been located. This decoding is the result after seven TR iterations (as opposed to 3 for the clean image runs), and in fact it is not yet fully converged, as evidenced by small one pixel "stubs" which protrude from some rectangles. In general, decoding such noisy images require a greater number of iterations before converging. The overall character of the decoding, however, can be said to appear much earlier. The location of the text lines, for example, appears after the first iteration, and indeed the text line segmentation appears to be a more robust feature of the decoding output than word boxes per se. A robust word boxing algorithm would probably need to incorporate more knowledge about words than is used by the present algorithm, which mainly utilizes the fact that words have a higher black pixel density than their surroundings (and that they are arranged in lines). A DID OCR engine with trained character templates built on top of a TR line segmenter might be able to serve this purpose.

Figures 8.5c and d show what can happen when the channel is not matched well to the observed image. In the first case (Figure 8.5c), the channel is the same as that used for Figure 8.4b, namely $\{\{0.999, 0.001\}, \{0.9, 0.1\}\}$. Since the foreground for this channel corresponds to a sparse density of black pixels (approximately 10%), the entire image is treated as foreground. Figure 8.5d exhibits the opposite extreme, where the foreground is expected to be nearly solid (90 %) black, $\{\{0.9, 0.1\}, \{0.1, 0.9\}\}$. TR in this case treats most of the image as background, with the foreground reduced to a small vertical sliver of black pixels in order to satisfy grammatical constraints.

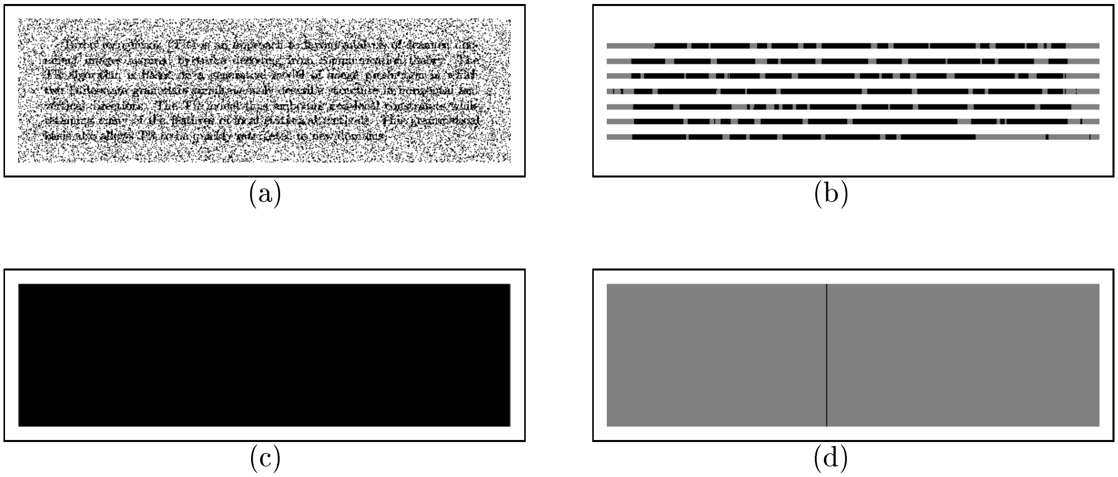


Figure 8.5: (a) Noisy text block image (shown earlier in Figure 8.3); (b) a well-tuned grammar and channel that yields a result (approximate word boxing) similar to that obtained in the noise-free case; (c) example decoding when the channel is too "bright"; (d) example decoding when the channel is too "dark."

8.4 One-layer grammars

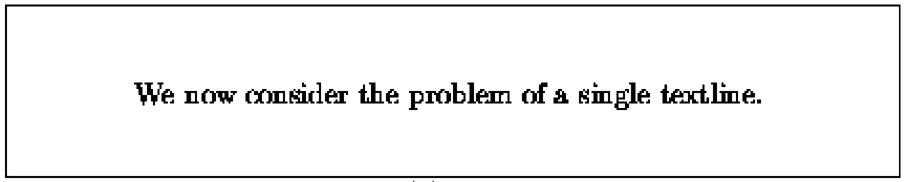
The results in the previous sections show that the TR framework is able to perform tasks such as finding text lines and possibly words, even in noisy conditions, by simply treating text as bit-flip noise. Due to the straightforward layout of the text in these images, we have employed "nonlocal" grammars thus far. Nonlocal grammars have some important limitations, however, as described previously in Section 7.1.1. It is of interest therefore to check whether one-layer grammars, mentioned previously as a more flexible alternative to nonlocal grammars, can also be applied to text images. We briefly summarize the results in this section.

8.4.1 Single text line

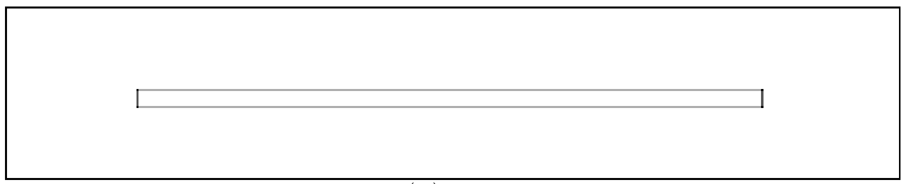
Given the single text line image shown in Figure 8.6a (which is identical to Figure 8.1a), the TR decoding output using a one-rectangle one-layer grammar and a channel set to $\{\{0.99, 0.01\}, \{0.5, 0.5\}\}$ is shown in Figure 8.6b. Only the pixels in the "protective layer" are shown. The text line is cleanly segmented in the by now expected manner, concentrating on the body of the text line, with ascenders and descenders cut off (cf. Figure 8.1b). As before, the character of the decoding varies somewhat with the choice of channel parameters.

8.4.2 Text block

Figure 8.7a shows the segmentation of a clean text block image (Figure 8.2a) using a single-column, multi-row grammar. This grammar limits the number of rectangles in any given row to be at most one. Figure 8.7b shows this result overlaid on the original observed image. In addition to the larger rectangles that correspond to the text line bodies,



(a)



(b)

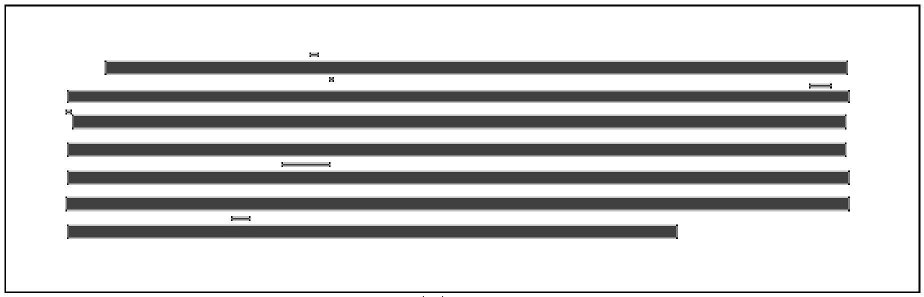
Figure 8.6: (a) One text line imaged at 100dpi (same as Figure 8.1); (b) TR decoding using a one-rectangle one-layer grammar.

TR returns small rectangles at the locations of a few ascenders and descenders. This is an example of how TR decoding behavior can differ somewhat from what one might first expect. Sometimes TR displays such an amazing capacity for unexpected interpretations of the observed data that managing its behavior is akin to “herding cats.” This probably reflects the inadequacies of the bit-flip model of text, suggesting that it might be fruitful to augment or replace it in some cases (some possibilities are discussed in the next chapter). There may also be an inherent ambiguity in the concept of a “shape” of a text line, for example. In any case, the small rectangles in Figure 8.7b could be removed with postprocessing if they are not of interest.

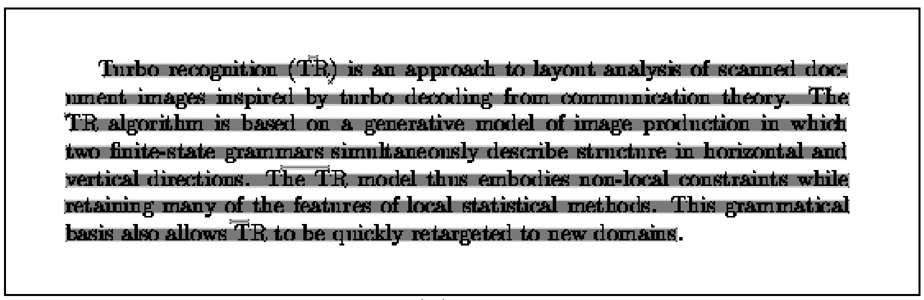
Using a multi-column, multi-row one-layer grammar on the text block image results in approximate word boxing, as shown in Figure 8.7c. This segmentation is evidently not ideal, with some words split into two pieces, although these splits do occur in natural places, namely where the character spacing is abnormally large within a word. It is also of interest that three of four periods are found as separate rectangles (as before). Increasing the image resolution would doubtless improve the results, although at a concomitant cost in speed.

We note that similar results on such clean images could be obtained using faster, and in some sense simpler, methods such as run-length smearing. We personally find that experimenting with TR models (such as adjusting channel parameters) is relatively simple and intuitive. The future role of TR in both research and practical application may come down to a matter of useability, as opposed to pure speed per se, and this is a topic we have yet to explore.

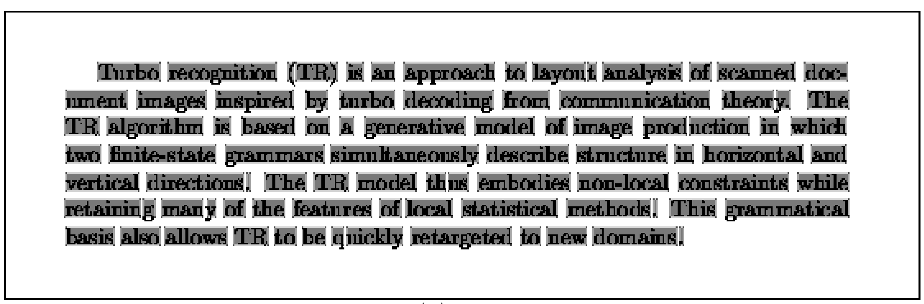
As noted earlier (in Section 8.3), if it were essential to accurately find word boxes, it would probably be best to call upon a character recognition engine (perhaps in conjunction with such TR results), rather than rely solely on this segmentation method which has no concept of words per se. The result shown in Figure 8.7c may nevertheless be useful for tasks such as estimating the setwidth of the “whitespace” character, which is an important



(a)



(b)



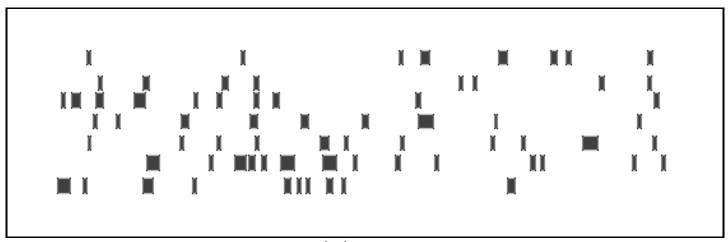
(c)

Figure 8.7: TR decoding output using one-layer grammars on a clean text block image. (a) Segmentation of a text block (Figure 8.2a) using a single-column, multi-row grammar; (b) the result in (a) overlaid on the original observed image; (c) Segmentation of the same block using a multi-column, multi-row grammar.

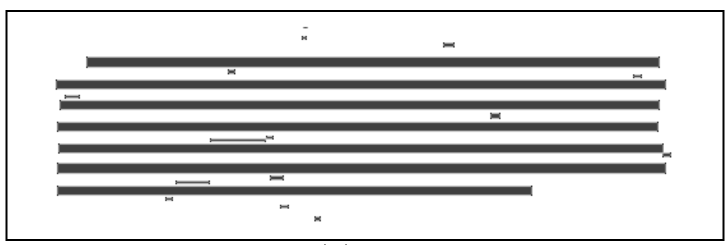
step in the initial training of a DID OCR decoder. The vertical edges of the (approximate) word boxes simplifies the measurement of the distance between them.

8.4.3 Noisy text block

The TR decoding behavior using one-layer rectangular grammars on the noisy text block image (Figure 8.3a) is described in Figure 8.8. Figure 8.8a is the output using the multi-column/multi-row grammar used in Figure 8.7 above. The noise has made word boxing largely infeasible using this method. Given the large amount of noise, this is not too surprising. It is interesting to note that the result using nonlocal grammars (Figure 8.5b is superior, suggesting that nonlocal grammars may be more robust against noise for those structures for which they apply[7]. Figure 8.8b shows the segmentation of the noisy text block image obtained using a single-column, multiple-row one-layer grammar. This result is similar to Figure 8.5c, and



(a)



(b)

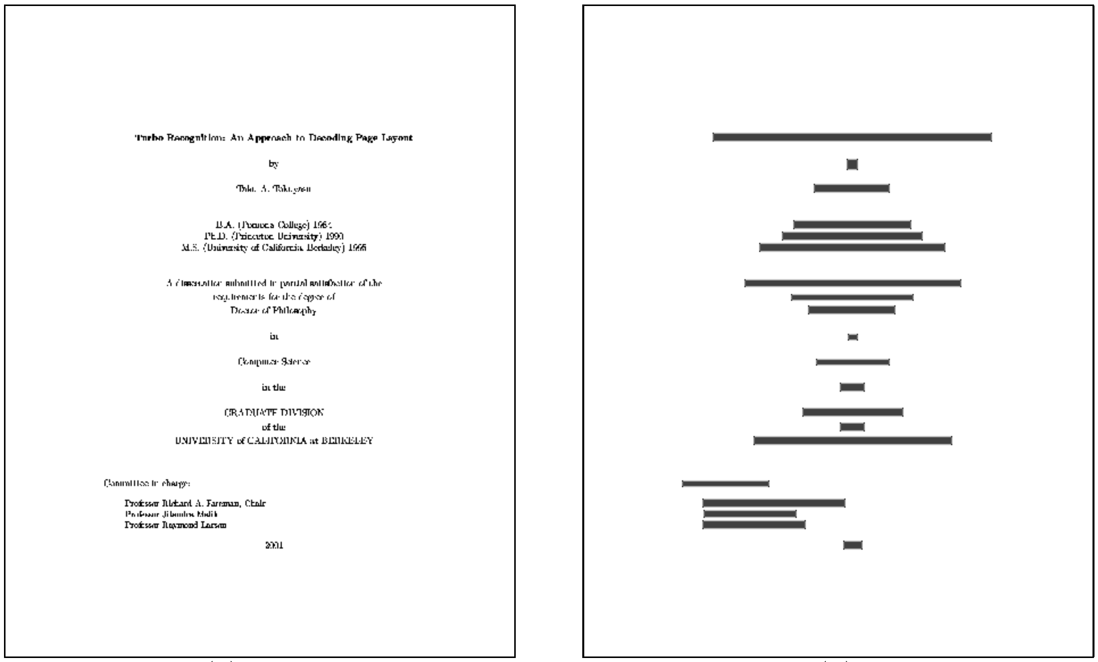
Figure 8.8: TR decoding output using one-layer rectangular grammars on a *noisy* image. (a) Output using the multi-column/multi-row grammar used in Figure 8.7 above. The noise has made word boxing infeasible using this method. (b) Segmentation of the same image using a single-column, multiple-row grammar.

8.4.4 Application

As a mild proof of concept, we apply our current approach to analyzing text lines to the title page of this thesis, shown in Figure 8.9a. The result shown in Figure 8.9b

[7]We have not examined why the decoding results differ in the two cases. The nonlocal and one-layer grammars are apparently describing the same grammatical structures, but differences in the FST transition probabilities may at least partially explain the difference.

was obtained using a one-column, multi-row one-layer grammar, with the channel given by $\{\{0.999, 0.001\}, \{0.5, 0.5\}\}$. Such output could be used with minimal postprocessing to find baselines, for subsequent use by a DID OCR engine, for example.



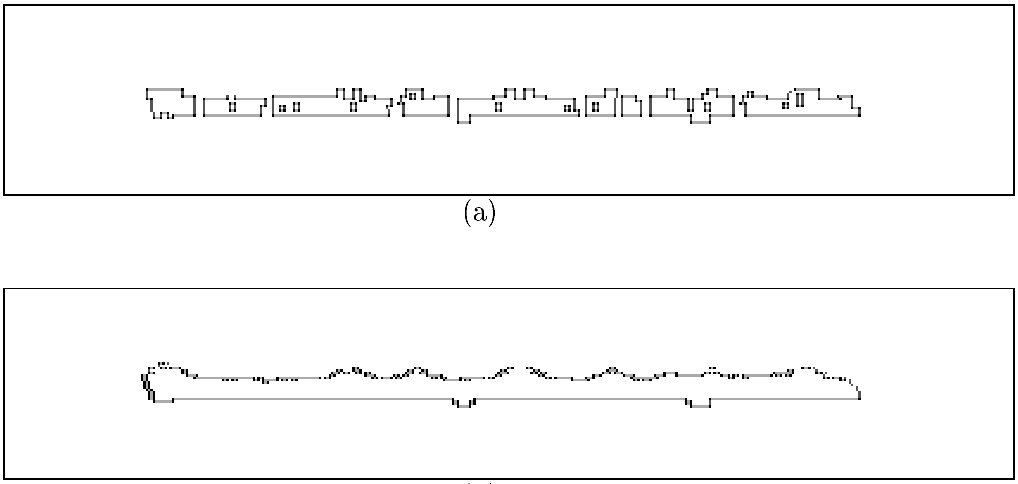
(a) (b)

Figure 8.9: Proof of concept: (a) The title page of this thesis; (b) TR result using a one-column/multi-row one-layer grammar.

8.5 Manhattan grammars

For the sake of completeness, we consider the use of Manhattan grammars on the same set of text images discussed previously. It is unlikely that the flexibility offered by Manhattan grammars would be of much use for simple text layouts, but in any case it is of interest to examine the range of behaviors exhibited by TR using Manhattan grammars in this context, which is somewhat surprising. We note that most of the output results shown in this section are not fully converged, for reasons that are discussed in detail elsewhere.

We first consider using the Manhattan grammar corresponding to Figure 7.5 on the clean single text line image used previously (Figure 8.6), which is 413 by 68 pixels in size. The channel is given by $\{\{0.999, 0.001\}, \{0.7, 0.3\}\}$. Seven turbo iterations were applied, with a runtime of approximately 22 seconds on a 900MHz PC. Figure 8.10a shows the TR decoding result. This almost appears like the original text line re-typeset in a strange font. The careful viewer may note that this is not a fully converged result. The decoding algorithm evidently has difficulties (due partially to the design of the grammar)



(a)

(b)

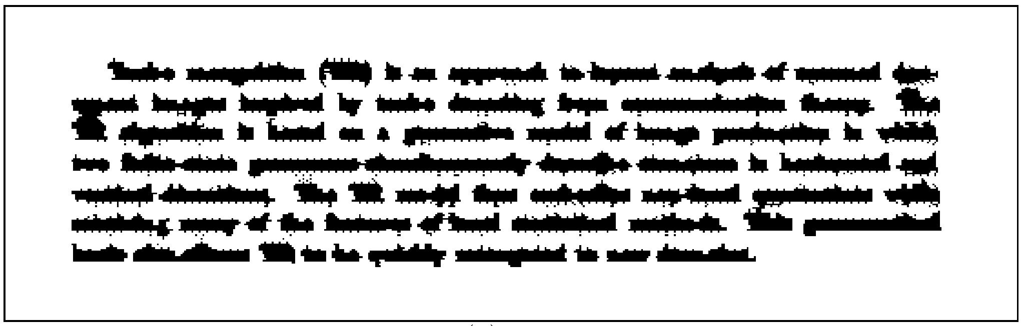
Figure 8.10: Application of the Manhattan grammar to a clean single text line. (a) Approximate word boxing; (b) Outline of the entire text line, produced by attempting to suppress the number of corner points.

in following bumps and other detailed image structure that are only one pixel wide, for example.

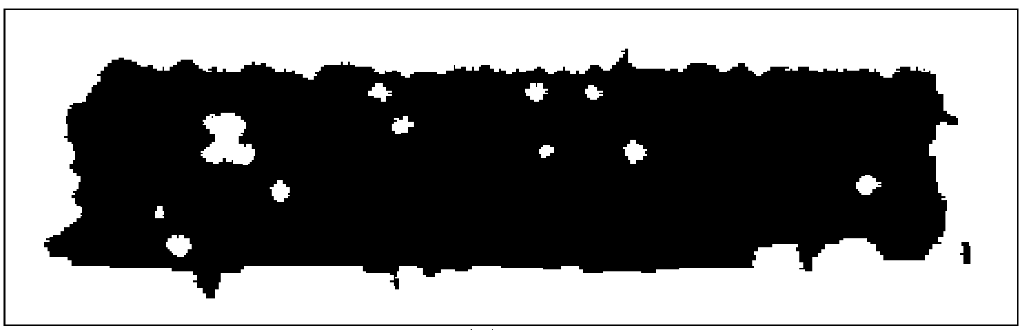
The result shown in Figure 8.10a has a rather complex appearance, and one might wonder if it might be possible to reduce this complexity by somehow suppressing the number of edges, to produce results akin to those in previous sections. One possible way to accomplish this is to change some of the transition probabilities in the FSTs. As mentioned previously in Chapter 4, the quantity that plays the role of a "transition probability" that is familiar from the HMM literature is the expression $P(u_i)P(s_{i+1} \mid s_i, u_i)$, where $P(s_{i+1} \mid s_i, u_i)$ corresponds to the FST transition probabilities specified in our grammars (see Chapter 4 and the footnote in Section 4.2.2 for further details and explanation of the notation). $P(u_i)$ is being updated by the TR turbo iterations, and hence we can view the progress of the TR algorithm as updating such HMM-equivalent transition probabilities. $P(u_i)$ is initially just a fixed constant for all i, however, so we can provide an initial bias on the TR algorithm by manipulating $P(s_{i+1} \mid s_i, u_i)$. Namely by drastically lowering the probability of some transitions, say from 1.0 down to 0.01, we can affect the character of the TR decoding output[8]. Figure 8.10b shows the result obtained by lowering the probability of transitions labeled with the input symbol b in the vertical grammar (to reduce the number of horizontal edges) and similarly lowering the probability of transitions labeled with c in the horizontal grammar (to reduce the number of vertical edges). Paradoxically, the number of edges seems to *increase* using the modified grammar, although the fact that the text line is segmented as one connected component could be taken as a form of reduction in complexity. The general issue of how best to manage the complexity of TR decoding results remains an open research question.

[8]Additional transitions, to the accepting state for example, need to be added so that the transition probabilities from each state continue to add up to one.

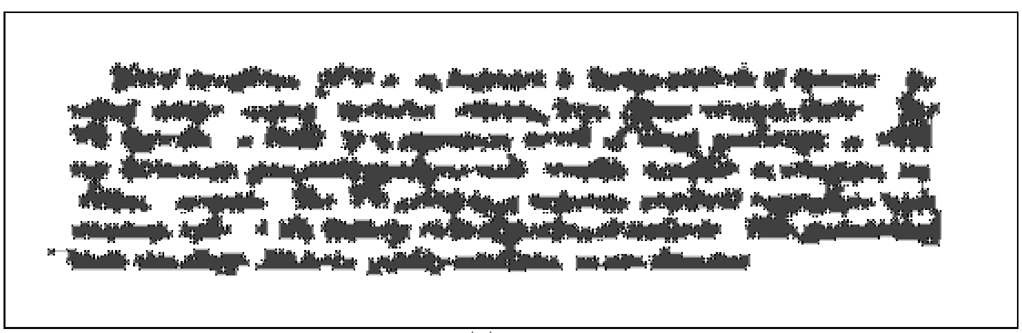
Figure 8.11 shows the application of Manhattan grammars to the text block images. Figure 8.11a shows the result of using the "suppressed" Manhattan grammar mentioned above (i.e., the one with modified transition probabilities) and a channel equal to $\{\{0.999, 0.001\}, \{0.5, 0.5\}\}$. This time only the foreground pixels are shown. The TR decoding is approximately returning words. The penchant of TR for complex decoding output when using Manhattan grammars is clearly seen.



(a)



(b)



(c)

Figure 8.11: Application of the Manhattan grammar to a text block. (a) Approximate word boxing on a clean text block; (b) complex decoding of a text block; (c) approximate word boxing of the noisy image.

Using the same grammar on the noisy text block image results in Figure 8.11b.

The channel in this case is $\{\{0.8, 0.2\}, \{0.5, 0.5\}\}$. The result is pretty much unuseable, except perhaps as a vague description of where the text is located. It is amusing that since the Manhattan grammar allows the formation of "holes," Figure 8.11b takes on the character of "Swiss cheese."

If the channel is modified to $\{\{0.8, 0.2\}, \{0.2, 0.8\}\}$, we obtain the result shown in Figure 8.11c, which roughly approximates word segmentation. Again, although we are trying to reduce the complexity of the TR decoding output by "suppressing" certain transitions, this approach clearly has not succeeded as well as one might like.

These are just examples of the TR decoding behavior using Manhattan grammars. A basic theme is the tendency of TR to produce complex patterns in an attempt to closely mimic the observed, perhaps more closely than one would like. Further applications of Manhattan grammars are given in the next chapter.

8.6 Blob grammars applied to text

In this section, we apply the blob grammars introduced in Section 7.1.4 to text lines. These grammars only minimally constrain image structure. They nevertheless may be useful, especially on clean text images.

Figure 8.12 shows the application of a progression of different blob grammars and channels to the single text line image given in Figure 8.1a. In Figure 8.12a the text line has been segmented using a single-blob grammar and a channel set to $\{\{0.999, 0.001\}, \{0.9, 0.1\}\}$. The only constraint provided by this grammar, which is nevertheless a powerful one, is that any row or column have at most one run of black pixels. While we have not extensively tested the use of this grammar, its simplicity is compelling and the results appear promising. Figure 8.12b shows the segmentation obtained using the same grammar with a slightly different channel $\{\{0.99, 0.01\}, \{0.5, 0.5\}\}$. The foreground pixels now expected to have a higher black pixel density, so that the segmentation "blob" tends to squeeze down into the higher density regions of the text line.

Figure 8.12c uses a single-blob (perhaps better termed as a single-interval) grammar vertically, but a blobs (multiple-interval) grammar horizontally, together with a channel equal to $\{\{0.999, 0.001\}, \{0.9, 0.1\}\}$. This results in segmenting each of the words in a manner somewhat akin to run-length smearing, with at most one black interval in each vertical slice.

Figure 8.12c shows the result with the obtained when a blobs (multiple-interval) grammar is used in *both* directions, with a channel equal to $\{\{0.99, 0.01\}, \{0.5, 0.5\}\}$. As might be expected, the TR result is *identical* to the original text line image, given in Figure 8.1a.

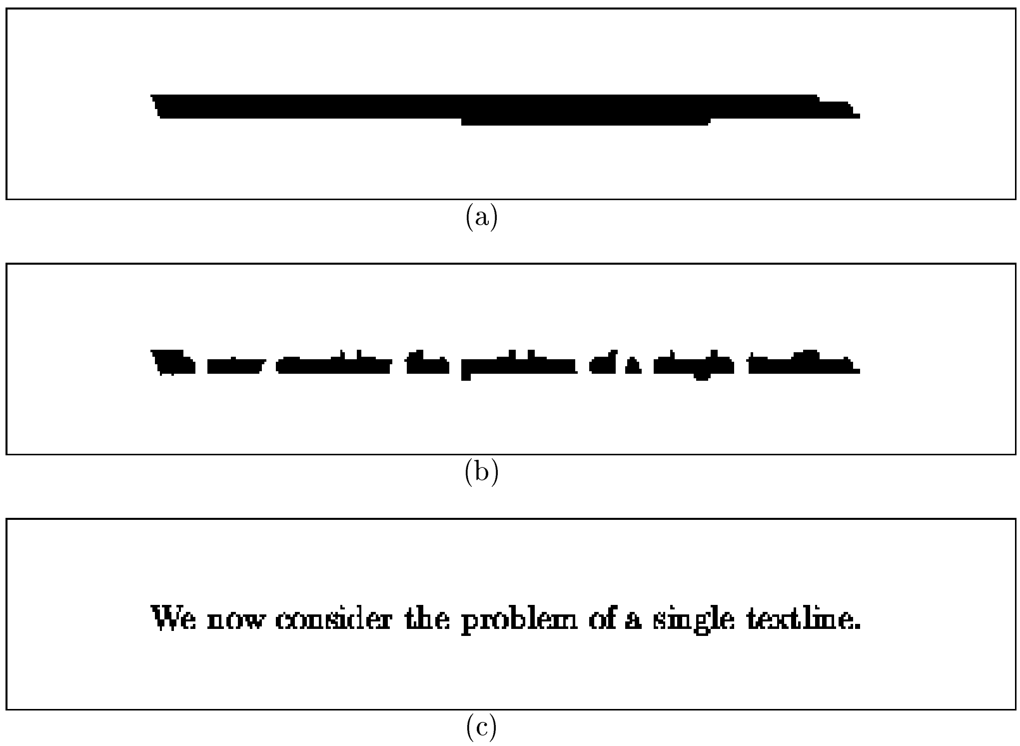


Figure 8.12: Application of the blob grammars to a single clean text line. (a) TR segmentation using a single blob grammar; (b) approximate "word boxing" using a blobs grammar horizontally and a blob grammar vertically; (c) decoding result using a multi-blob grammar with a dense channel, which in fact is the same as the original image.

Chapter 9

Further Applications of Turbo Recognition

In the previous chapter, we verified that the TR model, developed originally for recognizing grammatical structures in two dimensions, can be fruitfully applied to the document domain, in particular the physical layout analysis of text lines and text blocks. This involved the rather unusual step of modeling text as a form of noise. In this chapter, we elaborate further on this idea to treat a number of more complex examples. We also describe an application of TR to logical layout analysis. We then demonstrate an alternative method of modeling document content which does not treat it as noise. We close the chapter with a discussion of the noise robustness of TR.[1]

9.1 Two columns

We first turn to a variation on an old theme. Figure 9.1a shows a text document image 153 by 184 pixels in size, corresponding to roughly 20 dpi. To make the decoding more challenging, we add 10% bit-flip noise and some pen marks that would foil a projection profile approach, resulting in Figure 9.1b. We propose that the layout of this document can be described with the two column structure shown in Figure 9.1c. This leads to a two-column TR grammar that is a simple modification of the standard one-rectangle grammar described in Section 3.3, namely $a^+ \mid b^+c^+b^+c^+b^+$ horizontally and $a^+(b^+ \mid c^+)a^+$ vertically, where now we find it convenient to say b and c are separate foreground symbols. With the channel equal to $\{\{0.5, 0.5\}, \{0.88, 0.12\}, \{0.8, 0.2\}\}$, we obtain the result shown in Figure 9.1d, where only the c symbol is displayed, showing that the two column structure can be successfully extracted. This highlights the ability of TR to accumulate local evidence over long distances, in a manner that enforces non-local constraints, without making hard decisions at an intermediate step. This distinguishes it from methods that rely on local statistics or metrics alone such as most connected component and Markov random field methods.

[1] In some of the following figures, we use different shades of gray to denote different symbols in the TR output. In the printed version of this document, the different grayscale values are not apparent (i.e. the medium gray values look black). We apologize for this in advance.

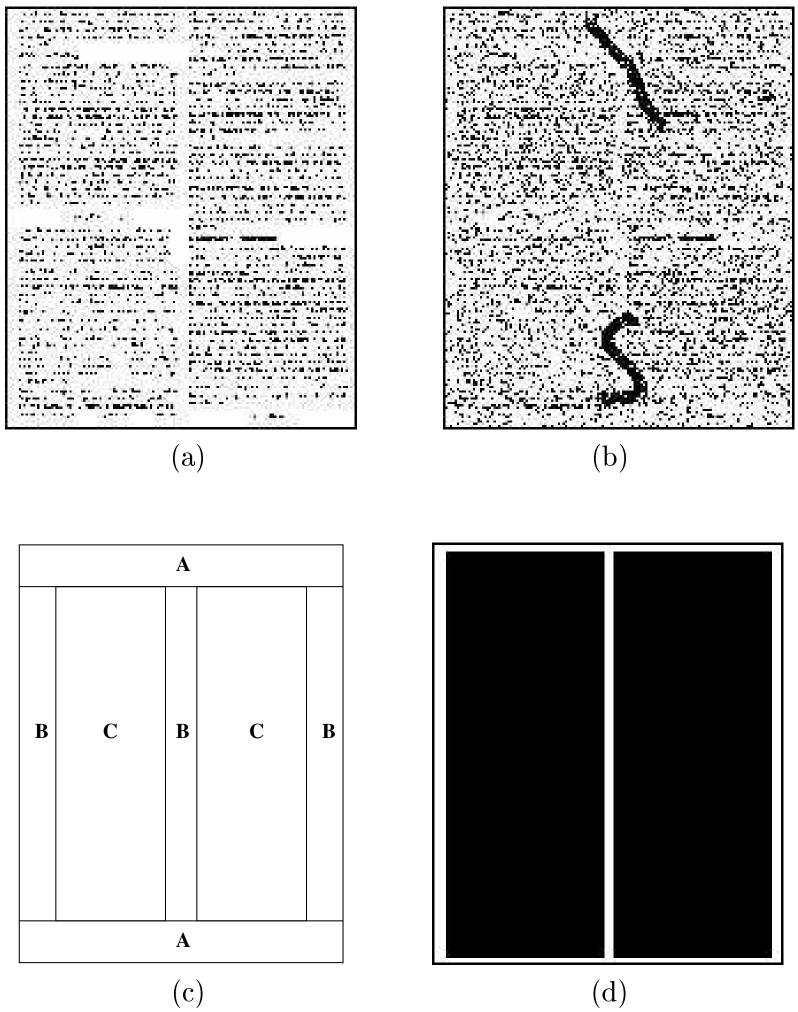
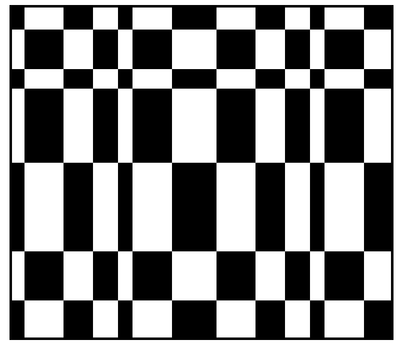


Figure 9.1: (a) Original two-column text image; (b) noisy observed version of (a); (c) layout structure described by a TR two-column grammar; (d) TR decoding result.

(a)



(b)

Figure 9.2: (a) Checkerboard image with 30% bitflip noise; (b) TR result with a checkerboard grammar with no constraint on size or number of blocks.

9.2 Checkerboard

To further explicate some of TR capabilities, we consider the noisy checkerboard image given in Figure 9.2a, which has been corrupted with thirty percent bit-flip noise and an extended pencil scrawl. Using a grammar for such patterns, which does not fix either the number or size of the individual blocks, TR easily recovers[2] the original clean image. This shows the ability of TR to integrate information over *two* dimensions (the entire image) before returning an estimate of the global optimum. Note that local estimates of the individual checkerboard blocks are unlikely to result in a pattern which is globally optimal or even consistent.

9.3 Mathematical expression

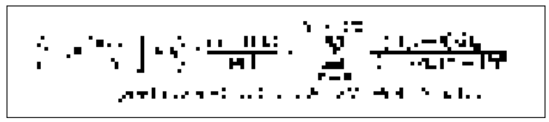
A somewhat unusual application of TR is shown in Figure 9.3a. This shows a mathematical expression taken from an integral table and imaged at very low resolution. We can extract structure from this rather chaotic collection of components by using the following multi-row grammar:

$$\begin{aligned} \mathit{horizontal}: &\quad a^+ \mid (\ b^+c^+)^+b^+ \\ \mathit{vertical}: &\quad (a^+(b^+ \mid c^+))^+a^+ \end{aligned}$$

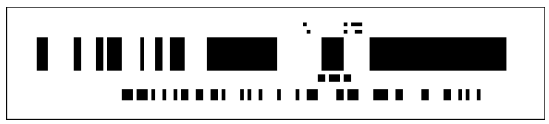
TR in this case is focussed on "clumping" black pixels into rectangles in a manner consistent with the grammar. This is largely in accord with typesetting conventions implicit in Figure 9.3a, allowing TR to summarize the image as a readily interpretable structure. For instance, the fact that this expression consists of two linear sequences, a large one on top and a smaller one below, can be extracted with minor postprocessing. The centerline,

[2]Run time in Java with no attempt at optimization is about four seconds on a 900 MHz Pentium PC. The image is 156 by 136 pixels and requires three turbo iterations. Typically three to seven iterations are required for convergence.

a hypothetical reference line passing horizontally through the center of the formula, can then be determined, which is crucial for any subsequent parsing [103]. In addition, the two fractions appear as prominent blocks, and the summation symbol with limits above and below is easily identified. While this does not demonstrate that TR is suitable for an actual mathematical parsing task, it is gratifying that such a summarization of image structure comes about with very little effort, with no need to build in application-specific knowledge via thresholds, etc.[3]



(a)



(b)

Figure 9.3: (a) Mathematical expression at low resolution; (b) TR decoding (c-symbol only).

9.4 Complications

In this section, we briefly discuss some of the problems we have encountered in using TR. At the end of this section, we introduce an innovation that holds some promise in dealing with many of these problems.

9.4.1 Non-convergence

One prominent difficulty is the failure of the TR decoding algorithm, or more precisely, our current implementation of it, to properly converge. This has become more noticeable as we deal with more complicated examples. The problem can be illustrated using the image shown in Figure 9.4a. This is a simple text image, but nevertheless it can offer a complex problem of "shape" recognition in its own right. Here we take shape to mean "a collection of rectangles." Figures 9.4b and c show intermediate TR results after one and two turbo iterations, respectively, without the use of deterministic annealing (see Section 4.3 for a description), and using a channel equal to $\{\{0.999, 0.001\}, \{0.1, 0.9\}\}$. We emphasize for clarity that these figures are just snapshots of the decoding process, displaying the most likely input symbols at each pixel at a given time, and in particular, no hard decision is made about the configuration of $\mathbf{U}$ at any point, except when the

[3]This result is reminiscent of run-length smoothing (RLS) [67] in either the horizontal or vertical direction (or both). It is unlikely that an RLS approach could exactly reproduce the structure in Figure 9.3, since the amount of smoothing required would vary across the image.

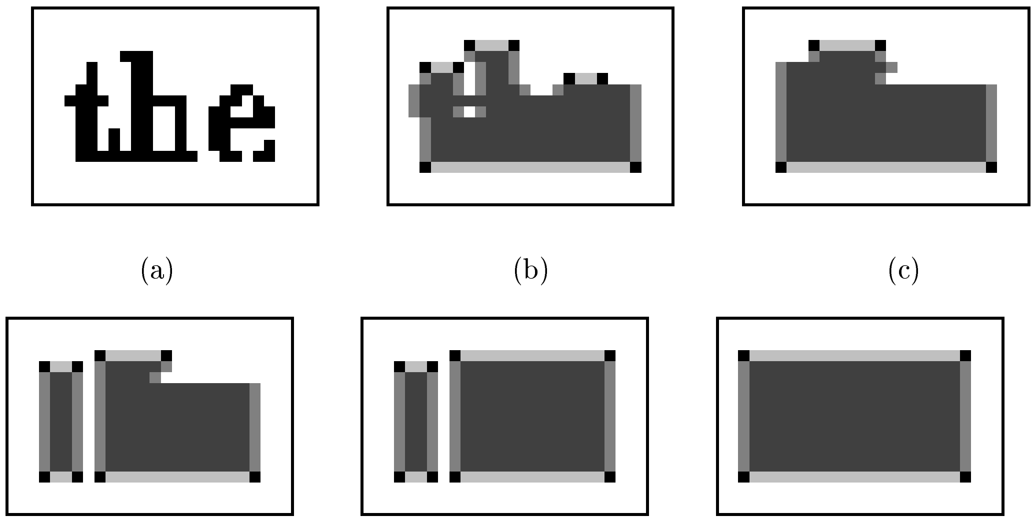
algorithm terminates. Having said this, we also note that the TR update process often rapidly drives the probabilities at each pixel to very close to zero or one. This can be a problem, since the $P(\mathbf{U})$ updates are the only means for the FSTs in the horizontal and vertical directions to communicate with each other. If the vertical FSTs, say, drive $P(\mathbf{U})$ too rapidly to zero or one, this may force critical transition probabilities for the horizontal FSTs (where "transition probability" is understood here in the HMM sense, as described in the footnote to Section 4.2.2) to zero, with the result that the horizontal FSTs are prevented from reaching the final state(s). Loosely speaking, "overconfidence in one direction yields problems in the orthogonal direction." Such problems should not arise in principle but do in practice, due to the lack of infinite numerical precision.

In the case of Figure 9.4, such problems arise on the next iteration. Specifically, during the backward pass for one of the image rows, the value of $\lambda(Z)$ evaluates to zero for all the values of Z, implying that $P(\mathbf{U})$ itself would evaluate to zero for all input symbols at the given pixel (and subsequent pixels on that row).

In order to deal with such images that are in a sense "far from grammatical," we use deterministic annealing [125]. Here we do not use the $\lambda_Z(U)$ message directly, but rather raise it to a power β first, before updating $P(U)$ (see Section 4.3). We typically start with $\beta = 0.15$ and increase it by a factor of 1.4 on each turbo iteration. The result of using deterministic annealing in this fashion is shown in Figure 9.4d. This is the result after six iterations, after which TR in fact again fails. We note though that the algorithm has made some progress towards a grammatical solution. Figure 9.4d shows the result when we start with $\beta = 0.15$ (as before) and increase it by a smaller factor of 1.2 on each iteration. This "slow" annealing process finally gives the TR algorithm enough room to find a grammatical solution.

Lest the reader get the impression that this problem is pervasive, even on such small images, we mention that this behavior can be moderated or eliminated entirely by the proper choice of channel. As shown in Figure 9.4f, TR has no difficulty segmenting this image using a moderately different channel, $\{\{0.999, 0.001\}, \{0.5, 0.5\}\}$, where we now employ our standard deterministic annealing schedule (β initially equal to 0.15 and increased by a factor of 1.4 on each iteration), and convergence is achieved in six iterations.

We note here that non-local grammars of the sort used in Figure 8.4, for example, are able to deal with this example with relative ease, over a wide range of channel parameters. Therefore, currently we are faced with a tradeoff of flexibility versus robustness. Simple grammars that are relatively inflexible exhibit better robustness, while flexible grammars that have the ability to encompass layouts of nearly arbitrary complexity seem to break more easily. Fundamental to this issue may be the lack of a characteristic length scale (besides the pixel spacing) in the TR model. One-layer Manhattan grammars, for example, have the ability to capture almost "fractal-like" aspects of the observed data. What seems to be required is some way to control this sensitivity to detail, by penalizing the number of corner points, for example, or inventing new classes of grammars. We leave this to future work.



(a) (b) (c)

(d) (e) (f)

Figure 9.4: (a) Text image at 100 dpi; (b) result after one turbo iteration, without deterministic annealing; (c) result after two iterations; (d) result after six iterations using an annealing parameter of 1.4; (e) result after 10 iterations using an annealing parameter of 1.2; (f) result after six iterations using a slightly different channel.

9.4.2 Two-as-one grammars

We introduce here what we call, for lack of a better name, "two-as-one" grammars, as a different approach to dealing with the above problems. The name arises since two input symbols together play a role akin to a single symbol in the previous grammars. We are motivated by the impression that currently we are forcing TR to model document content in some detail, which is not what TR is designed to do. Instead, we note for example that it is simple for us as humans to ignore details of a printed word, such as its font, presence of serifs, etc., to view it simply as a rectangle within which black pixels are compactly arranged. It thus may be fruitful if our recognition algorithm can similarly ignore such details and simply concentrate on forming grammatical regions, within which black and white pixels are arbitrarily distributed.

An implementation of this idea is shown in Figure 9.5, which shows a one-layer horizontal multi-column FST along the lines of the general Manhattan FST shown in Figure 7.5. The transitions for input symbols d and f now come in pairs, so that in any location where d can appear, f can also legally appear, and vice versa. Hence, the columnar constraints implied by this FST (when used in conjunction with a corresponding vertical multi-row FST, for example) are satisfied by regions within which d and f are freely intermixed.

We demonstrate how such grammars work by returning to the image of the word "the" used in the previous section (reproduced in Figure 9.6a for convenience). There are now two foreground symbols, $1'$ and $2'$, and the channel we use is a modification of the one used previously, $\{\{0.999, 0.001\}, \{0.5, 0.5\}, \{0.1, 0.9\}\}$. The result is shown in Figure 9.6b,

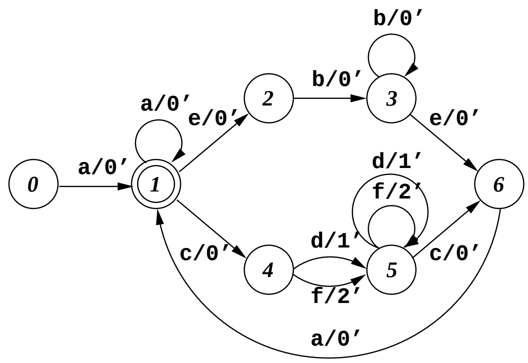
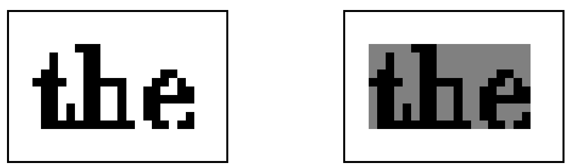


Figure 9.5: Example of an FST where the transitions for input symbol f duplicate those for d. The d and f symbols can thus appear freely intermixed within overall regions that satisfy the TR layout constraints. This figure shows a horizontal multi-column FST. The corresponding vertical multi-row FST is identical, except the b and c symbols are interchanged.

where grey corresponds to input symbol d and black to symbol f. As such, the black pixels in Figure 9.6b are not simply identical to the ones in Figure 9.4a, though they happen to be the same in this case. It is rather remarkable, given the experience in the previous section, that this result is achieved in only four iterations and *without* using deterministic annealing.



(a) (b)

Figure 9.6: (a) Observed text image used in the previous section (shown again here for convenience); (b) segmentation obtained using a multiple-row, multiple-column two-as-one grammar.

This result shows that, when TR is relieved of the responsibility of modeling text in any detail, many of the above-mentioned difficulties are at least alleviated. The use of two-as-one grammars by itself is not enough to solve all of our problems, on several counts, as will be seen in the following section. Nevertheless, this points at the sort of innovations that will be required in order to make TR more generally applicable to the document domain.

9.5 Mixed text and graphics

A natural generalization of the work on text-only content in the previous chapter is to consider images which contain both text and halftones. As suggested above, we have been developing two approaches to the analysis of composite documents of this kind. In the first approach, we extend the grammars introduced in the previous chapter to use two foreground symbols instead of just one, in essence allowing the grammatical regions to come in two types, which differ in their characteristic black pixel densities. This is an application of the "texture discrimination" approach described in Section 7.2. In the second approach, we again use two foreground symbols, but this time we allow them to be freely intermixed within grammatical regions, as described in the Section 9.4.2 above.

Figure 9.7a shows a multi-column image, 300 by 400 pixels in size (roughly 40 dpi), that includes both textual and graphical content in the form of binarized grayscale images. We first attempt to analyze this image with a one-layer multiple rectangle grammar with two foreground symbols, which was used previously for the synthetic example in Figure 7.8. In other words, we attempt to segment the image using two kinds of rectangles having different "textures." The channel has the reasonably standard form $\{\{0.999, 0.001\}, \{0.7, 0.3\}, \{0.1, 0.9\}\}$. After four iterations, we obtain the result shown in Figure 9.7b. This is mildly successful in forming a segmentation of the image according to texture. We note that the pixels making up the title text regions in the larger fonts towards the top of the page are placed in the same class as the graphics. This is natural, given that on the scale of a few pixels, such large characters indeed have a high black pixel density. The regions are not rectangular since the algorithm in fact has not converged, which is representative of a general phenomenon described in the previous section. These results nevertheless demonstrate how it is possible to bring some measure of texture segmentation capability within the TR framework, using simple bit-flip noise as a modeling tool.

The focus of the above approach on modeling document content as bit-flip noise is in a sense too low-level. For example, both extended regions of black and extended regions of white might be indicative of the presence of a graphic, as opposed to text, but it is impossible to capture both possibilities using a single level of noise.

This suggests that TR could benefit from a pre-processing phase, in which such knowledge about local correlations (at the level of a few pixels) is incorporated into the "observed evidence" that is input to the TR algorithm. The large body of research on texture analysis methods, including work cited in Chapter 1 for example, could undoubtedly be drawn upon in this regard, to allow TR to take one step back from the low-level details of the observed image.

For now, we consider taking the opposite tack from the above approach of incorporating more knowledge, in order to incorporate *less*. Thus, we employ the method described in Section 9.4.2, in which content regions are modeled by two symbols that are allowed to be arbitrarily mixed together. We can view this as a workaround within TR to eliminate the need to model textures directly. If we use essentially the same grammar as that used in Figure 9.7b, namely one which allows for an arbitrary number of rectangles, but instantiated as a two-as-one grammar, we obtain for example the result shown in Figure 9.7c. The channel in this case equals $\{\{0.999, 0.001\}, \{0.6, 0.4\}, \{0.1, 0.9\}\}$, and five TR iterations were applied. With simple postprocessing, such a result might be sufficient to find most of the text regions and text baselines, which could then be input for example to a DID OCR decoder. Figure 9.7d shows the result using the same two-as-one grammar, with the channel now equal to $\{\{0.999, 0.001\}, \{0.99, 0.01\}, \{0.1, 0.9\}\}$ and again with five TR iterations applied. The decoding has merged the components in Figure 9.7c into coherent regions. We note as before that the black pixels in this figure represent decoded regions of high black pixel density, rather than the original black image pixels per se. The extracted regions generally have a cleaner appearance than those in Figure 9.7b. Although these two-as-one grammar runs exhibit good progress towards reaching a grammatical solution, they both do not converge completely. Such output results nevertheless demonstrate the potential of such methods, and could be useful as they stand.

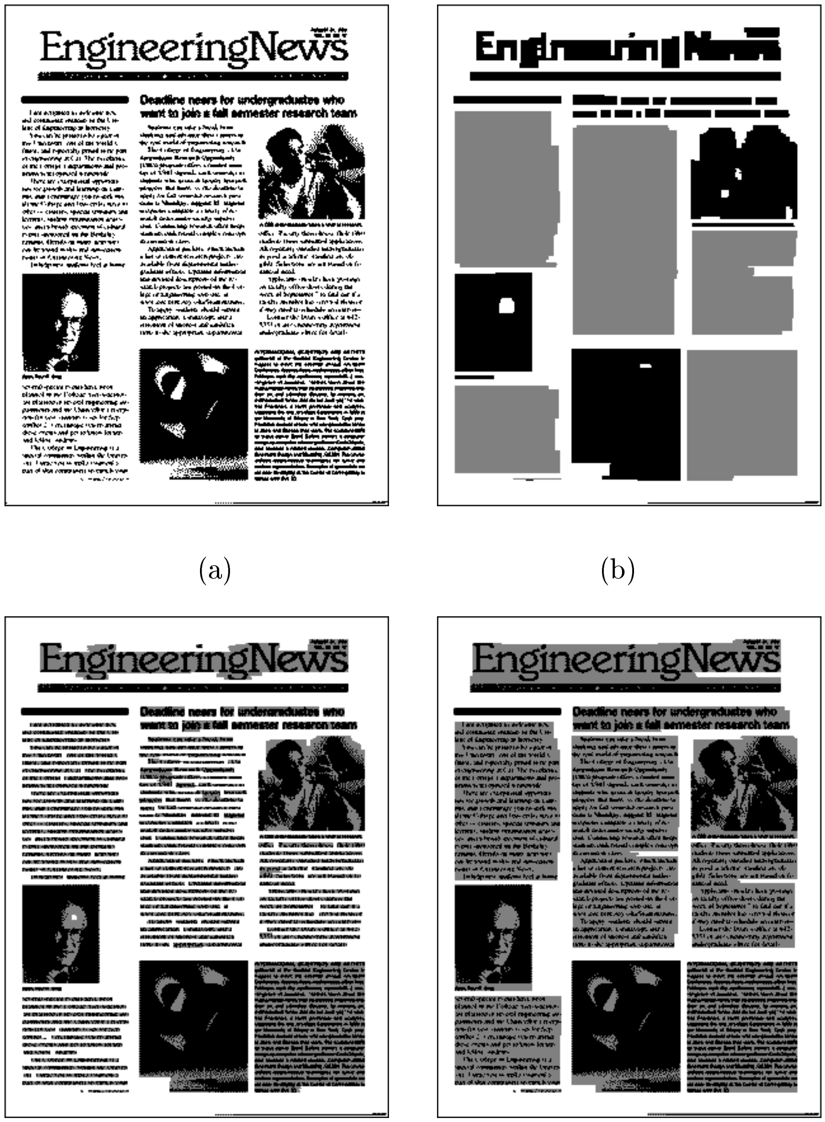
Given the above two approaches, which could be termed "texture segmentation" and "two-as-one," it is natural to consider whether it is possible to merge the two somehow. We have not yet been able to do so, although we believe such integration should be possible by increasing the size of the input and output alphabets.

9.6 Logical layout analysis

Much of this thesis has been devoted to understanding how to use TR to describe shapes and regions, and how to apply this knowledge to the document domain. Our exploration of the use of TR to analyze relationships *between* regions has been rudimentary. We have, for example, described structures such as multiple rows, multiple columns, and checkerboards, and also relatively unconstrained structures such as rectangles with arbitrary relative alignments and general Manhattan layouts. In many problem domains, however, such as table reading and understanding of business letters, more detailed knowledge is often available about the structure of such documents (see Sections 1.2.8 and 1.2.10 for some pointers to the literature). A TR grammar can be an effective medium for incorporating such knowledge. This shifts the focus of TR from purely physical layout analysis concerns to parsing the two-dimensional structure of a document into meaningful components.

As an example, we consider the journal article title page shown in Figure 9.8a. Metadata extraction from such pages is an important application of layout analysis [18]. We use a TR grammar that describes the structure of the page as the article title, followed by the abstract, followed by body text, where the first two components are single column and the last has two columns. The TR result is shown in Figure 9.8b. Here four TR iterations were applied, and a somewhat unusual channel was used, $\{\{0.999, 0.001\}, \{0.5, 0.5\}\}$ horizontally, and $\{\{0.999, 0.001\}, \{0.9, 0.1\}\}$ vertically.

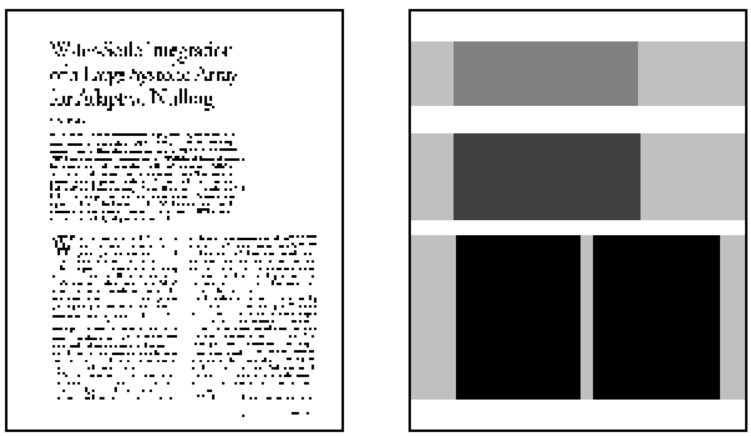
The different shades of gray in the top two blocks denotes the fact that they have been decoded by different input symbols. The TR insistence on grammatical interpretations suggests that it can boost overall zoning performance, even on such clean images.



(a) (b)

(c) (d)

Figure 9.7: (a) Binary image of a composite multi-column document; (b) segmentation into regions reflecting different textures, using a channel equal to $\{\{0.999, 0.001\}, \{0.7, 0.3\}, \{0.1, 0.9\}\}$; (c) decoding using a two-as-one grammar with the channel equal to $\{\{0.999, 0.001\}, \{0.6, 0.4\}, \{0.1, 0.9\}\}$; (d) decoding with the channel now equal to $\{\{0.999, 0.001\}, \{0.99, 0.01\}, \{0.1, 0.9\}\}$.



(a) (b)

Figure 9.8: a) Journal article title page; b) TR decoding into several logical zones.

Chapter 10

Conclusion

This dissertation describes turbo recognition (TR), a statistical page layout analysis approach based on communication theory. A TR model is based on two finite state machines, which describe image structure in the horizontal and vertical direction simultaneously. In this way, the model defines a set of grammatical images, corresponding to the usual one-dimensional notion of grammatical strings in a language. When combined with a noise channel, this forms a stochastic image generation process with an accompanying probability distribution over images. A TR decoder then seeks the optimal decoding of an observed image, consistent with the image grammars.

The TR decoding equations are essentially those used for turbo decoding, which are in turn closely related to techniques used for analyzing hidden Markov models. We borrow techniques from the the graphical model literature to derive the specific form of these equations. This results in an iterative decoding process which is linear in the size of the observed image. To improve the convergence properties of the algorithm, we employ deterministic annealing.

Since TR (as in turbo decoding) is not guaranteed to produce optimal results, we investigate the optimality of TR solutions with a simple experiment. The results show that TR indeed performs very close to optimally in extracting a single black rectangle from a noisy image, while remaining linear in complexity. This problem in two-dimensional recognition is sufficiently difficult that exhaustive search is the only method that can guarantee optimality, at the cost of quadratic complexity.

We demonstrate how TR can be used in the analysis of abstract image structure, and have explored the expressiveness of the TR formulation by developing a hierarchy of grammars of increasing complexity. We show how this framework can be applied to images of text lines, which shows promise for possible integration with OCR methods such as those previously developed within DID. We propose that the layout analysis of composite documents made up of text and graphics be treated as texture segmentation together with grammatical constraints. We show how the TR model as it stands is sufficiently flexible to make significant headway on this problem. The exploitation of the grammatical basis of TR to integrate physical and logical layout analysis has only just begun.

We have thus far limited ourselves to relatively simple regular grammars. With the work described here as a basis, there is a large variety of applications, such as the analysis of tables, forms, business letters, and articles, layout classification of documents, etc., that represent attractive avenues for future research. It may also be of interest to explore more powerful grammars, such as those having a fixed level of recursion. Such a grammar could describe a multi-column table within a multi-column document, for example. In general, it would be of interest to investigate if grammars could be combined in a modular fashion, and if so, to develop a formalism for hooking them together. The adoption of tools like *lex* could help automate the grammar generation process. This may also be useful from a performance standpoint, by automatically minimizing the finite state machines to be as small as possible.

We have yet to attempt large scale comparisons with other layout analysis methods, one reason being that TR can be targeted at regimes where other methods simply fail, such as when connected components yield a highly corrupted signal. TR also has much room to evolve. For example, our model of text is essentially bit-flip noise, which is clearly very impoverished. A form of TR operating on the results of even very simple local texture analysis methods could be quite powerful.

Whether or not TR proves to be the best layout analysis method according to any given performance metric, it remains attractive as a longer-range research target because of the possibility of integrating it with DID OCR models. In other words, rather than employing TR for layout analysis on its own, it could become part of a larger probabilistic model of a document page which encompasses both the layout and character levels. This would further reveal the depth available in the DID approach. We also hope our work in general fosters the further development of probabilistic models of layout, both within DID and in the DIA community at large.

Appendix A

Finite State Machines

This appendix provides an overview of finite state machines (FSMs). FSMs are the basis of TR encoding models and, amongst other things, control the complexity of the structures that can be decoded by TR. We collect some information about them and introduce our terminology here for reference purposes. We note in passing that the FSM nodes and arcs that appear below have a completely different meaning from those that appear in graphical models (which are described in Appendix B).

A.1 Introduction

A finite state machine is a dynamical system that can reside in one of a finite number of states at any given time. It evolves by taking one of a set of allowed transitions between states at discrete time steps. We note that in place of discrete time, any linear index can play the same role, such as a discrete spatial coordinate along a line.

The structure of an FSM can be illustrated diagrammatically as in Figure A.1, where each node represents a state of the system, and each directed arc denotes an allowed transition. The system begins in the distinguished initial ("start") state 0, marked by the

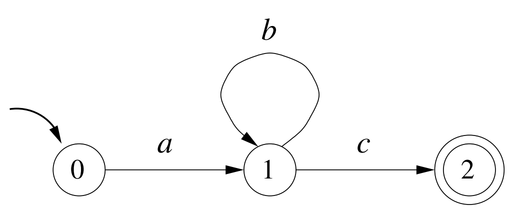


Figure A.1: Deterministic finite state automaton (DFA)

curved arrow.[1] It then evolves by following directed arcs between states. This evolution is allowed to terminate only in the "final" or "accepting" state, represented by the doubly-circled node 2. In general, an FSM can have multiple accepting states, and also transitions that exit them, allowing for further time evolution of the machine.

[1] We always display the initial state on the left, so we dispense with this arrow elsewhere.

Such machines are related to string grammars through labels on the arcs that are drawn from some input alphabet Σ. The machine processes a string by consuming it one character at a time, meanwhile following an arc corresponding to the character at hand (if such an arc exists). If there is more than one such transition, then the machine chooses one "non-deterministically."[2] If the machine terminates in the accepting state[3], the machine is said to *accept* the string, in which case we term the input string *grammatical* or *legal*. Otherwise, the machine rejects[4] the string as ungrammatical or illegal. Such a machine with two possible outputs (accept/reject) is called a *finite state automaton* (FSA). The *language* that an FSA accepts is defined to be the set of strings that drive the FSA in this manner to an accepting state. For instance, the FSA in Figure A.1 accepts exactly those strings in the language denoted by the regular expression $\alpha\beta^*\gamma$, where $\alpha = \{a\}$ $\beta = \{b\}$, $\gamma = \{c\}$. By an abuse of notation we will denote this as ab^*c.

Formally, a deterministic finite automaton (DFA) can be specified as a 5-tuple $M = (\Sigma, S, s_0, F, \delta)$, where Σ is the input alphabet, S is the set of states, s_0 is the initial or start state, $F \subseteq S$ is the set of final states, and $\delta : \Sigma \times S \rightarrow S$ is the transition function. The function δ is typically viewed as a partial function, i.e. it does not necessarily yield a result for all values in its domain.[5] The same machine can also be thought of as a generator of strings in ab^*c, by emitting (as opposed to "recognizing") the symbols that adorn its transitions.

A.2 Finite state transducers

FSMs can be made more expressive by labeling the transitions with both input and output labels, in which case we call the machine a *finite state transducer* (FST). For example, the FST in Figure A.2 is a transducer from the input alphabet $\{a, b, c\}$ to the output alphabet $\{0, 1\}$[6]. We again imagine the FST consuming an input symbol stream,

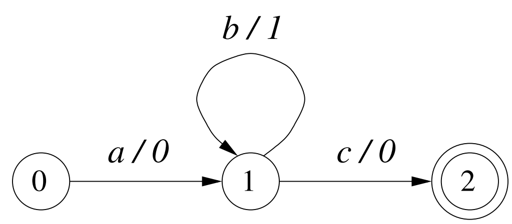


Figure A.2: A deterministic finite state transducer.

[2]This description may be objectionable to a purist, for whom non-determinism is simply a shorthand for an equivalent deterministic process. The algorithm for converting a non-deterministic FSM into a deterministic one is given for example by Hopcroft and Ullman [110].

[3]Formally, this requires that the input string be terminated by a special character which informs the machine that it has reached the end of the string.

[4] Strictly speaking, the machine as drawn in Figure A.1 cannot reject strings. An additional *reject* state can be added, with transitions constructed to it from all the other states, corresponding to input symbols which do not appear on the "legal" transitions.

[5]The addition of a reject state, as in Footnote 4, would remedy this situation.

[6]The arc labels in a transducer can be considered to be ordered pairs of symbols, in which case we can say the transducer encodes a regular *relation* between the input and output labels ([132]).

taking transitions according to each input symbol that it sees. The FST now also emits an output symbol for each transition that it takes, thus effecting a translation between the input and output streams. If no allowed transition exists, the input stream is declared illegal, and the output stream is thrown out.[7] For example, the FST in Figure A.2 accepts input strings ab^*c and emits output strings 01^*0.

Alternatively, an FST can be thought of as nondeterministically wending its way from the start state to the accepting state (or states), meanwhile generating both input and output streams in parallel. In particular, the FSA in Figure A.1, viewed as a generator of strings, can be considered a special case of this, where there are no input symbols[8].

In passing, we note that a nondeterministic machine with N states can be transformed into an equivalent deterministic machine with at most 2^N states [110] which accepts the same language. It could even be said more emphatically that a nondeterministic machine is precisely a shorthand for its deterministic counterpart. While such a transformation may be necessary for a non-deterministic machine to be well-defined, this not true in the case of probabilistic machines. Indeed, probabilities are often introduced exactly in situations which would otherwise be described as nondeterministic.

Finite state machines can be viewed in terms of the Chomsky hierarchy of formal languages, where they are equivalent to regular grammars, the lowest rung of the hierarchy. They are distinctly less powerful than context-free grammars, the next rung in the hierarchy. However, FSMs have the advantage in terms of parsing efficiency.

A.3 Probabilistic FSTs

So far, we have discussed deterministic FSMs (or their non-deterministic counterparts). In either case, for a given FSM, it is always clear whether a given sequence is grammatical or not, i.e. if it could have been produced (or accepted) by the machine. In many applications, however, it is useful to allow for relative judgments, rather than categorical declarations regarding grammaticality.

By way of motivation, consider the slightly more complicated FSM (Figure A.3). which describes a sequence a^+bb^+, with one or more a's, followed by two or more b's. The

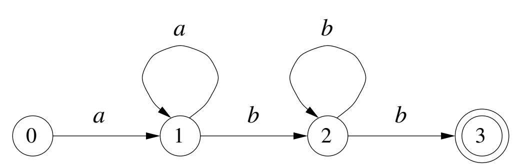


Figure A.3: Deterministic version of an FSM for recognizing sequences given by the regular expression a^+bb^+ in the presence of noise.

question arises, can we use this machine to help us interpret a sequence such as $aabbabb$? We could declare the sequence ungrammatical and simply ignore it. Clearly, for recognition

[7] A more careful statement would be, "If no legal transition exists, the FST makes a transition to a 'reject' state, and the translation fails."

[8] The labels on the arc in this generative mode can be represented as ϵ/x, where ϵ is the null input symbol, and x is a symbol from the output alphabet.

purposes, where the whole purpose is to analyze what we see, this is not very productive. Instead, we may note that the above sequence is "close" to the grammatical sequence *aabbbbb*, up to noise in the fifth bit, and hence we may consider this the best interpretation of the observed evidence within the model.

We formalize this notion by introducing probabilities into the model. First consider adding probabilities to the transitions (Figure A.4). The machine now randomly takes

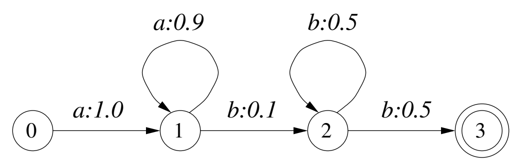


Figure A.4: Stochastic FSM for recognizing a^+bb^+ in the presence of noise. The final version (not shown) replaces each transition label (figuratively speaking) by a distribution over symbols in the alphabet, resulting in a type of hidden Markov model (HMM).

transitions from state to state, according to the probabilities on each arc[9]. The probabilities of the transitions exiting each state thus sum up to one. The probability of a given "path" from initial state to final state is the product of the transition probabilities along the path. We take this probability to also be the probability of the symbol sequence output by the machine. I.e., the machine naturally induces a probability distribution over the symbol sequences.[10]

This however is still not sufficient to solve our problem. We now add probability into the model in a second way. We replace, figuratively speaking, the single output symbol on an arc with a probability distribution over symbols. With a view to future applications, we maintain the deterministic relation between the input u and output symbol x on each arc, and now imagine the output symbol as being corrupted by random noise, before obtaining the *observed* symbol y. We assume a particular form for this noise process, where the probability of "bit-flip" for a given output symbol is constant (i.e. the same for a given output symbol, wherever it appears in the machine) and independent of the other symbols in the sequence. This defines a memoryless *channel*, specified by $P(y|x)$.

The general subject of probabilistic automata is treated in e.g. the book by Paz [133]. Probabilistic automata have been used in many contexts, such as chip design, communication theory, linguistics, speech recognition, and machine learning, to name a few. There is a discernible convergence of these disparate fields, but a standard terminology has yet to emerge. A summary of the main varieties of PFAs, which at least attempts to give them standard names, is provided by Murphy [134].[11]

[9] We note again that this is distinct from non-determinism.

[10] To be precise, the machine induces a probability distribution for states at a given time. Hence, it is customary to view the machine as inducing a probability distribution over strings of a fixed length (i.e. each string of a different length is a new event). Since we will be concerned with recognition (which is in a sense the reverse of generation), we will only be dealing with fixed-length strings in any case.

[11] In these terms, HMMs that we use as an example are output-NPFAs, and the probabilistic transducers that we use later are input-NPFAs.

A.3.1 Optimality

As mentioned previously, probabilistic automata give us relative judgments about differing interpretations of the observed data. We now describe methods for determining the best such interpretation. There are (at least) three standard conceptions of the word "best." For concreteness, we imagine having a string in hand, such as *aabbabc* above. The *maximum a posteriori* (MAP) solution maximizes the probability of all unobserved variables, given the evidence,

$$\arg\max_{\vec{s}} P(\vec{s} \mid \vec{x})$$

where $\vec{s}$ denotes the set of all unobserved ("hidden") variables, which in this case are the states. This is equivalent to maximizing the joint probability distribution for the entire model, since $P(\vec{s}|\vec{x}) = P(\vec{s},\vec{x})/P(\vec{x})$, where $P(\vec{x}) = \sum_{\vec{s}} P(\vec{s},\vec{x})$ is simply a constant, once $\vec{x}$ is fixed.

A closely related solution is the *maximum likelihood* estimate, $\vec{s}_{ML} = \arg\max_s P(x|s)$. This is identical to the MAP solution when the prior probability $P(s)$ is the uniform distribution (i.e. a constant).

We can also maximize the probability of each unobserved (hidden) variable separately, $\arg\max_{u_i} P(u_i \mid \vec{x})$, where a marginalization over all the other unobserved variables $u_j, j \neq i$ is implied. We note that the system configuration given by the set of maximum marginal values may actually have low, or even zero, probability, since memory about correlations that were present in the joint distribution were effectively lost in doing the marginalization. On the other hand, the MM solution minimizes the per-symbol error, i.e. the expected number of symbols which differ from the original values of $\vec{u}$.

We now turn to methods for obtaining such solutions. As these are described in many textbooks [88, 116], we simply summarize them here.

A.3.2 Viterbi algorithm

The Viterbi algorithm is a dynamic programming method for obtaining the best state sequence given the evidence (i.e. the MAP solution). We illustrate the algorithm for an HMM, where the observed symbols are emitted by the states as opposed to on the transitions. The algorithm proceeds by evaluating a quantity called, perhaps confusingly, the likelihood $\mathcal{L}$, which is defined on a *trellis* (see Figure A.5). The algorithm proceeds from left to right in the trellis, according to the recurrence relation

$$\mathcal{L}(s_j, t+1) = \max\ a_{ij} b_j(o_{t+1}) \mathcal{L}(s_i, t) \quad ,$$

where a_{ij} is the transition probability and $b_j(o_{t+1})$ is the probability for the observed symbol o_{t+1}) when the HMM is in state j.

$\mathcal{L}$ keeps track of the highest probability for reaching a given point of the trellis, starting from the initial state. If we maintain backpointers to the previous state from which the best path came, at each point in the trellis, then we can recover the overall best path by finding the trellis node with maximum $\mathcal{L}$ at $t = T-1$ (using a 0-based indexing convention), and following backpointers recursively until we reach the initial state.

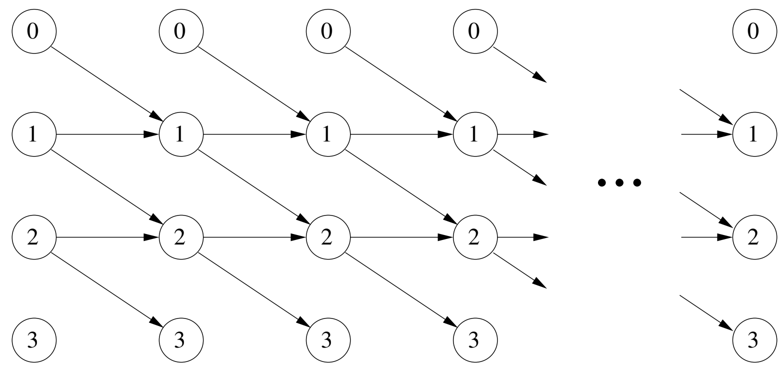


Figure A.5: Trellis for a four-state HMM.

A.3.3 Forward-backward algorithm

Whereas the Viterbi algorithm seeks to find a single optimal path through the model, the forward-backward algorithm is a means for calculating marginal probabilities for each unobserved *node* in the model. A quantity $\alpha(s,t)$ is defined using a recurrence which proceeds forward in time:

$$\begin{aligned} \alpha(s_i, t=0) &= \pi(s_i) b_i(o_t) \\ \alpha(s_j, t+1) &= \sum_i a_{ij}\ b_j(o_{t+1})\ \alpha(s_i, t) \end{aligned}$$

$\alpha(s_j, t+1)$ is the probability of emitting the observed symbol at time $t+1$ and reaching state s_j. Note the similarity to the Viterbi algorithm. Instead of maximizing over incoming paths, we take the sum. Otherwise the algorithms are the same.

Similarly, a quantity $\beta(s,t)$ is defined going backwards in time:

$$\begin{aligned} \beta(s_i, t=T-1) &= 1. \\ \beta(s_i, t-1) &= \sum_j a_{ij} b(o_t) \beta(s_j, t) \end{aligned}$$

This is the probability of being in state i at time $t-1$, given that you emit the remaining observed symbols, beginning at time t. Taking the product of $\alpha(s_i,t)$ and $\beta(s_i,t)$ at each node in the trellis yields a quantity $\gamma(s_i,t) = \alpha(s_i,t) \times \beta(s_i,t)$, which is (proportional to) the probability $P(s_i \mid \vec{x})$. Taking the maximum of $\gamma(s_i,t)$ at each time instant then yields the MM solution.

The reader may find it interesting to compare these equations with Sections 4.2.2 and 4.2.3. We note that there are slight differences, due to the fact that HMMs emit symbols from the nodes, whereas FSTs emit symbols on the transitions. The Viterbi and forward-backward algorithms are prominent examples of the maxproduct and sumproduct algorithms discussed in Appendix B.

Appendix B

Graphical Models

In Chapter 4, we employ methods from the field of graphical models to derive the TR decoding equations. Here we outline some basic background material on graphical models for reference purposes. For further details, the reader is referred to the influential work of Pearl [118] and also the book by Frey [116].

B.1 Introduction

A graphical model describes the probability distribution for a set of random variables by a graph, where the nodes represent the variables, and the arcs represent their statistical relationship. While both directed and undirected versions of graphical models exist, here we are only concerned with directed graphs.[1] Roughly speaking, a directed arc connotes the existence of a cause-and-effect relationship between the nodes at the tail and head of the arrow. It turns out that it is actually the *absence* of an arc between two nodes which is more significant, as it denotes their conditional *independence*. We also note that, strictly speaking, the graph does not describe a specific probabilistic model, but rather a whole family of distributions consistent with the constraints described by the graph.

For example, in the graph shown in Figure B.1, we can speak informally of the variable A as being the cause of the effects B and C. Specifically, the structure of the graph implies that the joint probability distribution $P(A,B,C)$ can be decomposed as follows:

$$P(A,B,C) = P(A)P(B|A)P(C|A,B) = P(A)P(B|A)P(C|A) \quad ,$$

where the first equation is the chain rule decomposition of the probability (i.e. is completely general), and the second follows from the above-mentioned conditional independence of B and C, given A. The form on the right-hand side can be read off directly from the graph as a product of factors, each being the conditional probability of a node given its parents.

In order for a directed graph to be a consistent probabilistic model, it cannot contain a directed loop, i.e., a cycle of nodes obtained by following directed arcs in the

[1] Undirected versions of graphical models differ somewhat from directed models in their semantics, and offer a different set of methods for describing probability distributions and doing inference. They may turn out to be of utility for turbo recognition, but we do not use them in this thesis.

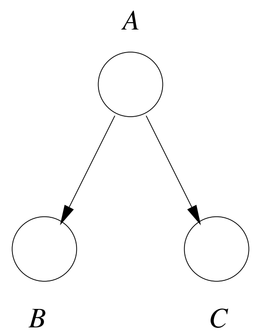


Figure B.1: Graphical model depicting the probabilistic relationship between three variables A, B, and C.

graph. The message passing methods described below require the stricter condition that the graph not contain any loops, even when the direction of the arcs is ignored. A graph consistent with this constraint takes the form of a *causal polytree* [118], as exemplified by Figure B.2. A causal polytree generalizes the notion of a tree, in that each node can have

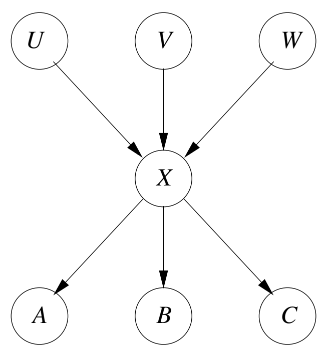


Figure B.2: Environment around a given node X in a causal polytree

multiple parents, in addition to multiple children. Thus there can be multiple root nodes, but loops are still not allowed, and the structure as a whole must be connected. Hence, causal polytrees retain the property that, between any two nodes, there is only one path. This implies that, by traveling "up" from a given node X to one of its parents, any path beyond that point leads to nodes which are not reachable by going "down" from X towards one of its children. X thus *separates* the graph into nodes "above" it and those "below" it. Following Pearl, we will use plus $(+)$ and minus $(-)$ to denote these portions of the graph, although we will also continue to use the informal terms up and down.

A basic task performed with such models is that of *inference*, i.e. determining how probabilities in the model change when new evidence is obtained. Suppose we have a graphical model which describes a set of variables $\mathbf{A} \cup \mathbf{B} = \{A_1, A_2, ..., A_N\} \cup \{B_1, B_2, ..., B_M\}$, with a corresponding probability distribution $P(\mathbf{A} \cup \mathbf{B})$. Suppose now that the variables in set $\mathbf{B}$ are observed to have specific values $\mathbf{B} = \mathbf{b}$, where this is construed as a vector equation, $B_i = b_i \ , \ i \in \{1, \ldots, M\}$. We would like to know how this influences the probability model for the remaining variables $\mathbf{A}$. For instance, we may be interested in the quantity

$P(A_i = a_i \mid \mathbf{b})$. This leads to the *maximum marginal* (MM) assignment, in which the probability for each variable A_i is maximized separately, $a_i^* = \arg\max_{a_i} \; P(A_i = a_i \mid \mathbf{b})$, $i \in \{1, \ldots, N\}$. Note that this implies a marginalization over all the other unobserved variables, and hence the MM assignment taken as a whole may have low or even zero probability in the original model.[2] Another quantity of interest is the *maximum a posteriori* (MAP) assignment $\mathbf{A} = \mathbf{a}^*$, where $\mathbf{a}^* = \arg\max_{\mathbf{a}} \; P(\mathbf{a} \mid \mathbf{b})$. A general method for producing such inferences in causal polytrees is known as message passing.

B.2 Message passing

In this thesis and DID in general, recognition is framed as a problem in statistical optimization, such as finding the maximum marginal or the MAP estimate. In the context of graphical models, the pioneering work by Pearl [118] describes such problems in terms of the *belief*, $BEL(x) \equiv P(x|e)$, i.e. the posterior distribution for the variable X, given the observed evidence e.[3] By evidence, we mean variables that have been instantiated to specific values. This results from the system having been observed in some way. A key feature of graphical models (subject to the above restrictions on the absence of loops) is the existence of a formal *message passing* algorithm for performing inference given such observations. We point out here that message passing comes in two varieties, known variously as belief updating and belief revision, or the sumproduct and maxproduct algorithms. We discuss the sumproduct version here, since in some ways it is more intuitive. The maxproduct algorithm is exactly the same, with the sums over a given set of variables replaced by maximizations over the same set of variables. The results described in this thesis were mostly obtained using the maxproduct algorithm.

We now sketch the derivation of the general (sumproduct) message passing equations, leaving technical details to the literature. We follow the treatment given in Pearl [118]. Consider a generic variable X embedded in a causal polytree, as shown in Figure B.3. Observed evidence in the form of instantiated variables will be distributed throughout the graph in some manner, and this can be divided up into two parts, the evidence e^+ that is embedded in the graph above X, and the evidence e^- in graph below X, $\{e\} = \{e^+\} \cup \{e^-\}$. We often drop the set notation in what follows for convenience.

We can separate out the effects of e^+ and e^- on $P(x|e)$ in the following way:

$$
\begin{aligned}
P(x|e) &= P(x,e)/P(e) \\
&= P(x,e^-,e^+)/P(e) \\
&= P(x,e^-|e^+)P(e^+)/P(e) \\
&= P(e^-|x,e^+)P(x|e^+)P(e^+)/P(e) \\
&= P(e^-|x)P(x|e^+)P(e^+)/P(e) \quad ,
\end{aligned}
$$

where the third and fourth equations follow from the definition of conditional probability, and the last equation follows from the fact that X separates e^+ from e^-. Since $P(e^+)$ and

[2] In the case of turbo recognition, this means that the MM image does not necessarily satisfy the FST grammar.

[3] We note again that the belief $P(x|e)$ entails a marginalization over the hidden variables besides X, taking into account *all* observed evidence.

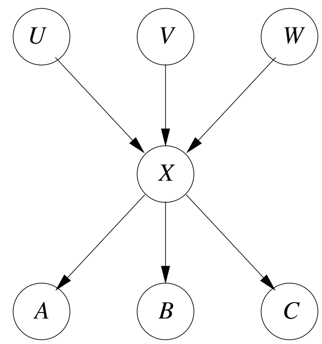


Figure B.3: Environment around a given node X in a causal polytree. All other nodes in the figure should be considered to be embedded in a similar environment, namely one with one or more parents, and one or more children (except of course at root and leaf nodes).

$P(e)$ are just constants, the x-dependence of $P(x|e)$ is captured by the following relation,

$$P(x|e) \propto \lambda(x) * \pi(x) \ ,$$

where $\lambda(x) \triangleq P(e^-|x)$ and $\pi(x) \triangleq P(x|e^+)$. The *belief* for X is thus the product of a *likelihood* from the evidence below X and a *prior* from the evidence above X, up to constant factors. We now focus on the two terms on the RHS in turn.

In the generic situation illustrated in Figure B.3, X has three children A, B, and C, which each make a separate contribution to $\lambda(x)$,

$$\lambda(x) = P(e^-|x) = P(e^-_{XA}, e^-_{XB}, e^-_{XC}|x) = P(e^-_{XA}|x)P(e^-_{XB}|x)P(e^-_{XC}|x) = \lambda_A(x)\lambda_B(x)\lambda_C(x) \ ,$$

where e^-_{XA} is the evidence embedded in the sub-polytree obtained by following the edge XA, $\lambda_A(x) \triangleq P(e^-_{XA}|x)$, etc. This relation holds since X separates the subgraphs formed by following XA, XB, and XC into disjoint polytrees. The likelihood $\lambda(x)$ thus factorizes into a product of likelihoods coming from each of the child branches.

As for $\pi(x)$, the evidence e^+ divides up into parts stored in each of the component branches that come into X from above. For the situation displayed in Figure B.3, we have $e^+ = e^+_{UX} \cup e^+_{VX} \cup e^+_{WX}$, where e^+_{UX} is the evidence embedded in the portion of the graph obtained by following the edge UX upwards, etc. The contribution from each of these branches to $\pi(x)$ can be accumulated as follows:

$$\begin{aligned}
\pi(x) &= P(x|e^+) \\
&= P(x|e^+_{UX}, e^+_{VX}, e^+_{WX}) \\
&= \sum_{u,v,w} P(x,u,v,w|e^+_{UX}, e^+_{VX}, e^+_{WX}) \\
&= \sum_{u,v,w} P(x|u,v,w)P(u,v,w|e^+_{UX}, e^+_{VX}, e^+_{WX}) \\
&= \sum_{u,v,w} P(x|u,v,w)P(u|e^+_{UX})P(v|e^+_{VX})P(w|e^+_{WX}) \ .
\end{aligned}$$

The last step follows from the fact that, when marginalized, parents of a given node are mutually independent.[4] Introducing some more notation, we summarize the above relation as

$$\pi(x) = \sum_{u,v,w} P(x|u,v,w)\pi_X(u)\pi_X(v)\pi_X(w)$$

where $\pi_X(u) \triangleq P(u|e^+_{UX})$ is the "prior" on U coming from evidence stored in the UX sub-tree, and $\pi_X(v)$ and $\pi_X(w)$ are defined similarly. Unlike the case of $\lambda(x)$, the contributions from each branch to $\pi(x)$ are not simply multiplied together, but must be "linked in" via the conditional probability $P(x|u,v,w)$ and a sum over the parental variables. In general, in situations where all the children of a given node are involved, the corresponding expression is simply a product of factors from each child branch, whereas when all the parents of a node are involved, the corresponding expression requires linking the factors from each parent branch together with a conditional probability and a summation.

To summarize our progress so far, we have shown that the belief $BEL(x)$ at node X is proportional to the product of a prior $\pi(x)$ and a likelihood $\lambda(x)$. Each of these factors in turn can be derived from *messages* such as $\lambda_A(x)$ and $\pi_X(u)$ coming from the children and parents of X, respectively, in accordance with the formulas given above.

To complete the picture, we need to specify recursively how the messages are constructed from π and λ at other nodes, and the messages coming into those nodes. Consider for example what X should send upwards to a parent U, in support of the belief calculation at U. From U's point of view, the message that it expects to receive from X is the likelihood $\lambda_X(u) \triangleq P(e^-_{UX}|u)$ that describes the evidence in the UX sub-polytree. As before, we can develop an expression for $\lambda_X(u)$ by first examining how different portions of the evidence contribute to it. Note that the evidence e^-_{UX} is not simply stored in the children of X, but also in the portions of the graph accessible by going up through other parents of X. In the situation given in Figure B.3, the evidence e^-_{UX} can be divided up into the portions stored above V and W and that stored below X,

$$e^-_{UX} = e^+_{VX} \cup e^+_{WX} \cup e^-_X \ .$$

It follows that

$$\begin{aligned}
P(e^-_{UX}|u) &= P(e^+_{VX}, e^+_{WX}, e^-_X|u) \\
&= \sum_{x,v,w} P(x,v,w,e^+_{VX}, e^+_{WX}, e^-_X|u) \\
&= \sum_{x,v,w} P(x,e^+_{VX}, e^+_{WX}, e^-_X|u,v,w)P(v,w|u) \\
&= \sum_{x,v,w} P(e^+_{VX}, e^+_{WX}, e^-_X|x,u,v,w)P(x|u,v,w)P(v)P(w) \\
&= \sum_{x,v,w} P(e^+_{VX}|v)P(e^+_{WX}|w)P(e^-_X|x)P(x|u,v,w)P(v)P(w) \quad .
\end{aligned}$$

[4] A general algorithm for determining such independence judgments is known as the "Bayes' Ball" algorithm. We refer to the graphical literature for details.

Since $P(e^+_{VX}|v)P(v) = P(v|e^+_{VX})P(e^+_{VX})$ and similarly $P(e^+_{WX}|w)P(w) = P(w|e^+_{WX})P(e^+_{WX})$ by Bayes' Rule, we get upon rearranging terms,

$$\begin{aligned}\lambda_X(u) &= \sum_{x,v,w} P(x|u,v,w)P(v|e^+_{VX})P(w|e^+_{WX})P(e^-_X|x)P(e^+_{VX})P(e^+_{WX}) \\ &\propto \sum_{x,v,w} P(x|u,v,w)\pi_X(v)\pi_X(w)\lambda(x) \ .\end{aligned}$$

The message that X sends to U can thus itself be constructed from the messages $\pi_X(v)$ and $\pi_X(w)$ coming from the siblings of U and the λ messages from the children of X that form $\lambda(x)$. In effect, we have pushed the recursion for message passing one step further out towards the roots and leaves of the polytree.

We now consider the message that X has to send to its child A. The evidence e^+_{XA} can be decomposed as follows, $e^+_{XA} = e^+_X \cup e^-_{XB} \cup e^-_{XC}$, i.e. into the evidence above X and the evidence stored below the siblings of A. The message $\pi_A(x)$ then is

$$\begin{aligned}\pi_A(x) &= P(x|e^+_{XA}) \\ &= P(x|e^+_X, e^-_{XB}, e^-_{XC}) \ .\end{aligned}$$

We find it more convenient to work with a quantity that is proportional to $\pi_A(x)$, namely $P(x, e^-_{XB}, e^-_{XC}|e^+_X) = P(x|e^-_{XB}, e^-_{XC}, e^+_X)P(e^-_{XB}, e^-_{XC}|e^+_X)$ where the last factor is a constant. Applying the definition of conditional probability to pull out x, we get

$$\begin{aligned}P(x, e^-_{XB}, e^-_{XC}|e^+_X) &= P(e^-_{XB}, e^-_{XC}|x, e^+_X)P(x|e^+_X) \\ &= P(e^-_{XB}|x)P(e^-_{XC}|x)P(x|e^+_X) \\ &= \lambda_B(x)\lambda_C(x)\pi(x) \ .\end{aligned}$$

where the second line follows from the fact that X separates the subgraphs containing the evidence e^-_{XB} and e^-_{XC}.

B.2.1 Summary

We summarize the message passing algorithm as follows. The belief at a node X is the product of a likelihood λ and a prior π,

$$\begin{aligned}BEL(x) &\triangleq P(x|e) \\ &= P(x,e)/P(e) \\ &= P(x,e^-,e^+)/P(e) \\ &= P(x,e^-|e^+)P(e^+)/P(e) \\ &= P(e^-|x,e^+)P(x|e^+)P(e^+)/P(e) \\ &= P(e^-|x)P(x|e^+)P(e^+)/P(e) \\ &\propto \lambda(x) * \pi(x) \ .\end{aligned}$$

The quantity $\lambda(x)$ is a product of λ messages from each of the children, which for the situation shown in Figure B.3 takes the following form,

$$\begin{aligned}\lambda(x) = P(e^-|x) &= P(e^-_{XA}, e^-_{XB}, e^-_{XC}|x) \\ &= P(e^-_{XA}|x)P(e^-_{XB}|x)P(e^-_{XC}|x) \\ &= \lambda_A(x)\lambda_B(x)\lambda_C(x) \quad .\end{aligned}$$

$\pi(x)$ is a product of π messages from each of the parents, linked in with a conditional probability,

$$\begin{aligned}\pi(x) &= P(x|e^+) \\ &= P(x|e^+_{UX}, e^+_{VX}, e^+_{WX}) \\ &= \sum_{u,v,w} P(x,u,v,w|e^+_{UX}, e^+_{VX}, e^+_{WX}) \\ &= \sum_{u,v,w} P(x|u,v,w)P(u,v,w|e^+_{UX}, e^+_{VX}, e^+_{WX}) \\ &= \sum_{u,v,w} P(x|u,v,w)P(u|e^+_{UX})P(v|e^+_{VX})P(w|e^+_{WX}) \\ &= \sum_{u,v,w} P(x|u,v,w)\pi_X(u)\pi_X(v)\pi_X(w) \quad .\end{aligned}$$

The messages that appear in the above equations can be formed recursively from messages coming from further out in the graph. For example, the message produced by X for its parent U has the following form,

$$\begin{aligned}\lambda_X(u) &= P(e^-_{UX}|u) \\ &= P(e^+_{VX}, e^+_{WX}, e^-_X|u) \\ &= \sum_{x,v,w} P(x,v,w,e^+_{VX}, e^+_{WX}, e^-_X|u) \\ &= \sum_{x,v,w} P(x,e^+_{VX}, e^+_{WX}, e^-_X|u,v,w)P(v,w|u) \\ &= \sum_{x,v,w} P(e^+_{VX}, e^+_{WX}, e^-_X|x,u,v,w)P(x|u,v,w)P(v)P(w) \\ &= \sum_{x,v,w} P(e^+_{VX}|v)P(e^+_{WX}|w)P(e^-_X|x)P(x|u,v,w)P(v)P(w) \\ &= \sum_{x,v,w} P(x|u,v,w)P(v|e^+_{VX})P(w|e^+_{WX})P(e^-_X|x)P(e^+_{VX})P(e^+_{WX}) \\ &\propto \sum_{x,v,w} P(x|u,v,w)\pi_X(v)\pi_X(w)\lambda(x) \quad .\end{aligned}$$

The message created by X for each of its children is a simple product of factors, as exemplified by the following expression,

$$\begin{aligned}
\pi_A(x) &= P(x|e^+_{XA}) \\
&= P(x|e^+_X, e^-_{XB}, e^-_{XC}) \\
&\propto P(e^-_{XB}, e^-_{XC}|x, e^+_X)P(x|e^+_X) \\
&= P(e^-_{XB}|x)P(e^-_{XC}|x)P(x|e^+_X) \\
&= \lambda_B(x)\lambda_C(x)\pi(x) \quad .
\end{aligned}$$

In this way, messages are passed recursively through the graph from the borders towards the interior and then back out to the borders.

The message passing algorithm is initialized as follows. At each root node, the quantity π is replaced by the prior on that node. For each uninstantiated leaf node, the quantity λ is set equal to one, $\lambda = \{1, 1, \ldots, 1\}$. If a node is observed to have its jth value, then λ is set equal to $\{0, 0, \ldots, 1, 0, \ldots, 0\}$, where the jth component is equal to one. Alternatively, observing a node X is equivalent to adding a dummy child node whose sole purpose is to propagate a λ message to X that equals a delta function at its observed value. Termination of the message passing algorithm can be guaranteed in causal polytrees. For details, the reader is referred to Chapter 4 of Pearl [118].

Bibliography

[1] P.A. Chou and G.E. Kopec. A stochastic attribute grammar model of document production and its use in document image decoding. In *Document Recognition II, Proc. of the SPIE*, volume 2422, pages 66–73, 1995.

[2] G.E. Kopec and P.A. Chou. Document image decoding using Markov source models. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 16(6):602–17, June 1994.

[3] Hewlett-Packard Company. Capshare 920 portable e-copier. http://www.capshare.com/product/.

[4] C Technologies AB. C-pen 800. http://www.cpen.com/product/cpen800.shtml.

[5] Jonathan J. Hull and Peter E. Hart. Towards zero-effort personal document management: A new paradigm for users. *Ricoh Technical Report No. 26*, pages 61–68, November 2000. http://www.ricoh.co.jp/rdc/techreport/No26/Ronbun/A07.pdf. See also http://www.ricoh-usa.com/prodshw/solutions/ecabinet.htm.

[6] Kris Popat. Decoding of text lines in grayscale document images. In *Proceedings of the 2001 International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2001)*, Salt Lake City, Utah, May 2001. IEEE. To appear.

[7] Steve Ready *et al.* Xerox PARC Bookscanner. http://www.parc.xerox.com/eml/members/ready/parc_bookscanner.htm.

[8] Hao Chen, Jianying Hu, and Richard W. Sproat. Integrating geometrical and linguistic analysis for email signature block parsing. *ACM Transactions on Information Systems*, 17(4):343–366, 1999.

[9] J. Hu, R. Kashi, D. Lopresti, and G. Wilfong. Table structure recognition and its evaluation. In *Document Recognition and Retrieval VIII, Proc. of the SPIE*, volume 4307, pages 44–55, 2001.

[10] G. Semeraro, F. Esposito, and D. Malerba. Learning contextual rules for document understanding. In *Proceedings of the Tenth Conference on Artificial Intelligence for Applications*, pages 108–15, Los Alamitos, CA, 1994. IEEE Comput. Soc.

[11] Richard Power and Donia Scott, editors. *Using Layout for the Generation, Understanding, or Retrieval of Documents. AAAI Fall Symposium, November, 1999.* Technical Report FS-99-04.

[12] Digital Libraries Initiative. Sponsored by the National Science Foundation and other federal agencies. http://www.dli2.nsf.gov/.

[13] Joint Conference on Digital Libraries, JCDL '01, Roanoke, VA, 24-28 June 2001. Sponsored by ACM and IEEE. http://www.jcdl.org/.

[14] UC Berkeley Digital Library Project. Part of the Digital Libraries Initiative, Phase 2, sponsored by the National Science Foundation and other federal agencies. http://elib.cs.berkeley.edu.

[15] Robert Wilensky and Thomas A. Phelps. Multivalent documents: A new model for digital documents. Technical Report CSD-98-999, University of California, Berkeley, March 1998. See also http://www.cs.berkeley.edu/˜wilensky/MVD.html ?lexical-signature=multivalent+copyeditor+dlib+annotating+enlivened.

[16] Lawrence O'Gorman and Rangachar Kasturi, editors. *Document image analysis.* IEEE Computer Society Press, Los Alamitos, CA, 1995.

[17] ScanSoft Inc. TextBridge Pro Millennium. http://www.scansoft.com/.

[18] Jongwoo Kim, Daniel X. Le, and George R. Thoma. Automated labeling in document images. In *Document Recognition VIII, Proceedings of the SPIE*, volume 4307, pages 111–122, 2001.

[19] Stephen V. Rice, George Nagy, and Thomas A. Nartker. *Optical character recognition : an illustrated guide to the frontier.* Kluwer Academic Publishers, Boston, c1999.

[20] Serge Belongie, Jitendra Malik, and Jan Puzicha. Shape matching and object recognition using shape contexts. Accepted for publication in PAMI.

[21] T.K. Ho, J.J. Hull, and S.N. Srihari. A word shape analysis approach to lexicon based word recognition. *Pattern Recognition Letters*, 13(11):821–826, 1992.

[22] A. L. Spitz. Shape-based word recognition. *International Journal on Document Analysis and Recognition*, 1(4):178–90, May 1999.

[23] Shyh-Shiaw Kuo Kuo and O.E Agazzi. Keyword spotting in poorly printed documents using pseudo 2-d hidden markov models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(8):842–8, August 1994.

[24] R. Cattoni, T. Coianiz, S. Messelodi, and C. M. Modena. Geometric layout analysis techniques for document image understanding: a review. January 1998. ITC-irst Technical Report #9703-09.

[25] Y. Y. Tang, M. Cheriet, Jiming Liu, J. Said, and Ching Y. Suen. Document analysis and recognition by computers. In C.H. Chen, L.F. Pau, and P.S.P. Wang, editors, *Handbook of Pattern Recognition and Computer Vision*, chapter 8. World Scientific, Singapore, second edition, 1999.

[26] H. Bunke and P.S.P. Wang, editors. *Handbook of character recognition and document image analysis.* World Scientific, Singapore, 1997.

[27] H.S. Baird, H. Bunke, and K. Yamamoto, editors. *Structured document image analysis.* Springer-Verlag, Berlin, 1992.

[28] Document Layout Interpretation and its Applications (DLIA99), online proceedings. http://www.science.uva.nl/events/dlia99/.

[29] Document Layout Interpretation and its Applications (DLIA2001). http://www.science.uva.nl/events/dlia2001/.

[30] F. M. Wahl, K. Y. Wong, and R. G. Casey. Block segmentation and text extraction in mixed text/image documents. *Computer graphics and image processing*, 20(4):375–390, 1982.

[31] L.A. Fletcher and R. Kasturi. A robust algorithm for text string separation from mixed text/graphics images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(6):910–18, Nov. 1988.

[32] L. O'Gorman. The document spectrum for page layout analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11):1162–73, November 1993.

[33] H.S. Baird, S.E. Jones, and S.J. Fortune. Image segmentation by shape-directed covers. In *Proc. 10th International Conference on Pattern Recognition*, volume 1, pages 820–5. IEEE Comput. Soc. Press, 1990.

[34] T. Pavlidis and J. Zhou. Page segmentation by white streams. In *Proc. of the First International Conference on Document Analysis and Recognition*, pages 611–619, Los Alamitos, CA, 1991. IEEE Comput. Soc. Press.

[35] A. Antonacopoulos and R.T. Ritchings. Segmentation and classification of document images. In *IEE Colloquium on Document Image Processing and Multimedia Environments*, pages 16/1–7, 1995.

[36] A. Antonacopoulos. Page segmentation using the description of the background. *Computer Vision and Image Understanding*, 70(3):350–69, June 1998.

[37] K. Kise and O. Yanagida. Page segmentation using thinning of white areas. *Systems and Computers in Japan*, 29(3):59–68, March 1998.

[38] H.S. Baird. Background structure in document images. *International Journal of Pattern Recognition and Artificial Intelligence*, 8(5):1013–30, Oct. 1994.

[39] K. Kise, A. Sato, and M. Iwata. Segmentation of page images using the area voronoi diagram. *Computer Vision and Image Understanding*, 70(3):370–82, June 1998.

[40] S. Mao and T. Kanungo. Empirical performance evaluation methodology and its application to page segmentation algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(3):242–256, March 2001.

[41] G. Nagy and S. Seth. Hierarchical representation of optically scanned documents. In *Seventh International Conference on Pattern Recognition*, volume 1, pages 347–9, Los Angeles, CA, 1984. IEEE Comput. Soc. Press.

[42] M. Krishnamoorthy, G. Nagy, S. Seth, and M. Viswanathan. Syntactic segmentation and labeling of digitized pages from technical journals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(7):737–47, July 1993.

[43] A.A. Efros and T.K. Leung. Texture synthesis by non-parametric sampling. In *Proc. of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1033–8, Los Alamitos, CA, 1999. IEEE Comput. Soc.

[44] D.F. Dunn, T.P. Weldon, and W.E. Higgins. Extracting halftones from printed documents using texture analysis. *Optical Engineering*, 36(4):1044–52, April 1997.

[45] Anil K. Jain and Yu Zhong. Page layout segmentation based on texture analysis. *Pattern Recognition*, 29(5):743–70, 1996.

[46] Y.Y. Tang, Hong Ma, Dihua Xi, Xiaogang Mao, and C.Y. Suen. Modified fractal signature (MFS): a new approach to document analysis for automatic knowledge acquisition. *IEEE Transactions on Knowledge and Data Engineering*, 9(5):747–62, Sept-Oct 1997.

[47] D. S. Bloomberg. Textured reductions for document image analysis. In *Document Recognition III, Proc. of the SPIE*, volume 2660, pages 160–74, 1996.

[48] Ross Kindermann and J. Laurie Snell. *Markov random fields and their applications*. American Mathematical Society, Providence, R.I., 1980.

[49] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(6):721–41, Nov 1984.

[50] Rama Chellappa and Anil Jain, editors. *Markov random fields : theory and application*. Academic Press, Boston, 1993.

[51] Jia Li, A. Najmi, and R.M. Gray. Image classification by a two-dimensional hidden markov model. *IEEE Transactions on Signal Processing*, 48(2):517–33, Feb 2000.

[52] K. Etemad, D. Doermann, and R. Chellappa. Page segmentation using decision integration and wavelet packets. In *Proceedings of 12th IAPR International Conference on Pattern Recognition*, volume 2, pages 345–9. IEEE Comput. Soc. Press, 1994.

[53] H. Choi and R. Baraniuk. Multiscale document segmentation using wavelet-domain hidden markov models. In *Document Recognition VII, Proc. of the SPIE*, volume 3967, pages 234–47, 2000.

[54] Jia Li and R.M. Gray. Context-based multiscale classification of document images using wavelet coefficient distributions. *IEEE Transactions on Image Processing*, 9(9):1604–16, Sept 2000.

[55] J. Liang, J. Ha, R.M. Haralick, and I.T. Phillips. Document layout structure extraction using bounding boxes of different entitles. In *Proceedings of the Third IEEE Workshop on Applications of Computer Vision. WACV'96*, pages 278–83. IEEE Comput. Soc. Press, 1996.

[56] W.T. Freeman, E.C. Pasztor, and O.T. Carmichael. Learning low-level vision. *International Journal of Computer Vision*, 40(1):25–47, 2000.

[57] International Organization for Standardization. Information technology – Open Document Architecture (ODA) and interchange format, 1994. ISO/IEC 8613.

[58] International Organization for Standardization. Information processing – text and office systems – Standard Generalized Markup Language (SGML), 1986. ISO 8879:1986.

[59] T. Fruchterman. DAFS: A standard for document and image understanding. In *Symposium on Document Image Understanding Technology, Bowie, MD*, pages 94–100, Oct. 1995.

[60] Xerox Imaging Systems. Xdoc data format: Technical specification. part no. 00-07571-00.

[61] Tim Bray, Jean Paoli, and C.M. Sperberg-MacQueen, Oct. 2000. http://www.w3.org/TR/REC-xml.

[62] A. Lawrence Spitz. Style-directed document segmentation. In *Symposium on Document Image Understanding Technology (SDIUT01)*, pages 195–199, College Park, MD, 2001. UMIACS. Available online at http://lamp.cfar.umd.edu/sdiut01/SDIUT2001-Proceedings/.

[63] F. Esposito, D. Malerba, and F.A. Lisi. Machine learning for intelligent processing of printed documents. *J. of Intelligent Information Systems*, 14(2-3):175–98, March-June 2000.

[64] Chang Ha Lee and Tapas Kanungo. The architecture of TRUEVIZ: A groundTRUth/metadata Editing and VIsualiZing toolkit. In *Symposium on Document Image Understanding Technology (SDIUT01)*, pages 299–319, College Park, MD, 2001. UMIACS. Available online at http://lamp.cfar.umd.edu/sdiut01/SDIUT2001-Proceedings/.

[65] R.J. Boeri. Adobe Acrobat Capture 2.0. *EMedia Professional*, 11(4):82, 84, April 1998.

[66] Yann LeCun, Leon Bottou, Patrick Haffner, and Jeffery Triggs. Overview of the djvu document compression technology. In *Symposium on Document Image Understanding Technology (SDIUT01)*, pages 119–122, College Park, MD, 2001. UMIACS. Available online at http://lamp.cfar.umd.edu/sdiut01/SDIUT2001-Proceedings/.

[67] K.Y. Wong, R.G. Casey, and F.M. Wahl. Document analysis system. *IBM Journal of Research and Development*, 26(6):647–56, Nov 1982.

[68] H.R. Stabler. Experiences with high-volume, high-accuracy document capture. In A.L. Spitz and A. Dengel, editors, *Proc. of IAPR Workshop on Document Analysis Systems*, pages 38–51, Singapore, 1995. World Scientific.

[69] A.L. Spitz. Style directed document recognition. In *Proc. of the First International Conference on Document Analysis and Recognition*, pages 611–619, Los Alamitos, CA, 1991. IEEE Comput. Soc. Press.

[70] J.H. Shamilian, H.S. Baird, and T.L. Wood. A retargetable table reader. In *Proc. of the Fourth International Conference on Document Analysis and Recognition*, volume 1, pages 158–63. IEEE Comput. Soc. Press, 1997.

[71] J. Hu, R. Kashi, and G. Wilfong. Comparison and classification of documents based on layout similarity. *Information Retrieval*, 2(2-3):227–43, 2000.

[72] A. Dengel and F. Dubiel. Logical labeling of document images based on form layout features. In *Proc. Workshop on Document Image Analysis (DIA'97)*, pages 26–31, Los Alamitos, CA, 1997. IEEE Comput. Soc. Press.

[73] A. Kornai and S.D. Connell. Statistical zone finding. In *Proceedings of 13th International Conference on Pattern Recognition*, volume 3, pages 818–22, Los Alamitos, CA, 1996. IEEE Comput. Soc. Press.

[74] A.K. Jain and Yao Chen. Address block location using color and texture analysis. *CVGIP: Image Understanding*, 60(2):179–90, Sept. 1994.

[75] Paul Stefan Williams and Mike D. Alder. Generic texture analysis applied to newspaper segmentation. preprint available at http://ciips.ee.uwa.edu.au/Papers/Conference_Papers/1996/04/Index.html.

[76] P.S. Williams and M.D. Alder. Generic texture analysis applied to newspaper segmentation. In *Proceedings of International Conference on Neural Networks (ICNN'96)*, volume 3, pages 1664–9, New York, 1996. IEEE.

[77] Dacheng Wang and S. N. Srihari. Classification of newspaper image blocks using texture analysis. *Computer Vision, Graphics, and Image Processing*, 47(3):327–52, Sept. 1989.

[78] O.E. Kia and D.S. Doermann. Residual coding in document image compression. *IEEE Transactions on Image Processing*, 9(6):961–9, June 2000.

[79] Qin Zhang, J.M. Danskin, and N.E. Young. A codebook generation algorithm for document image compression. In J.A. Storer and M. Cohn, editors, *Procs. DCC '97, Data Compression Conference*, pages 300–9, Los Alamitos, CA, 1997. IEEE Comput. Soc. Press.

[80] S.J. Inglis and I.H. Witten. Bi-level document image compression using layout information. In J.A. Storer and M. Cohn, editors, *Procs. DCC'96, Data Compression Conference*, page 442, Los Alamitos, CA, 1996. IEEE Comput. Soc. Press.

[81] P. Haffner, L. Bottou, P.G. Howard, and Y. LeCun. DjVu: analyzing and compressing scanned documents for Internet distribution. In *Proc. of the Fifth International Conference on Document Analysis and Recognition - ICDAR '99*, pages 625–8, Los Alamitos, CA, 1999. IEEE Comput. Soc.

[82] G.E. Kopec. Document image decoding in the Berkeley Digital Library. In *Proc. 3rd IEEE International Conference on Image Processing*, volume 2, pages 769–72, New York, 1996. IEEE.

[83] Casey Palowitch and Darin Stewart. Automating the structural markup process in the conversion of print documents to electronic text. In *Proc. of The Second International Conference on the Theory and Practice of Digital Libraries*, 1995. http://www.csdl.tamu.edu/DL95/papers/palowitc/palowitc.html.

[84] D. Niyogi and S.N. Srihari. The use of document structure analysis to retrieve information from documents in digital libraries. In *Document Recognition IV, Proc. of the SPIE*, volume 3027, pages 207–18, 1997.

[85] Eytan Adar and Jeremy Hylton. On-the-fly hyperlink creation for page images. In Frank M. Shipman, Richard Furuta, and David M. Levy, editors, *Digital Libraries '95. Proc. of The Second Annual Conference on the Theory and Practice of Digital Libraries*, 1995. http://www.csdl.tamu.edu/DL95/papers/adar/adar.html.

[86] Claude E. Shannon and Warren Weaver. *The Mathematical Theory of Communication.* University of Illinois Press, Urbana, IL, 1949.

[87] Thomas M. Cover and Joy A. Thomas. *Elements of information theory.* Wiley, New York, 1991.

[88] Lawrence Rabiner and Biing-Hwang Juang. *Fundamentals of speech recognition.* Prentice Hall, Englewood Cliffs, N.J., 1993.

[89] Frederick Jelinek. *Statistical methods for speech recognition.* MIT Press, Cambridge, MA, 1997.

[90] Richard O. Duda and Peter E. Hart. *Pattern classification and scene analysis.* Wiley, New York, 1973.

[91] G.E. Kopec. Multilevel character templates for document image decoding. In *Document Recognition IV, Proc. of the SPIE*, volume 3027, pages 168–77, February 1997.

[92] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing.* The MIT Press, Cambridge, MA, 1999.

[93] H. Abelson and A. diSessa. *Turtle Geometry.* The MIT Press, Cambridge, MA, 1980.

[94] A.C. Kam and G.E. Kopec. Document image decoding by heuristic search. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 18(9):945–50, Sept. 1996.

[95] G.E. Kopec and M. Lomelin. Supervised template estimation for document image decoding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(12):1313–24, December 1997.

[96] Thomas P. Minka, Dan S. Bloomberg, and Kris Popat. Document Image Decoding using iterated complete path search. In *Document Recognition and Retrieval VIII, Proceedings of the SPIE*, volume 4307, pages 250–258, 2001.

[97] Kris Popat, Dan Greene, Justin Romberg, and Dan S. Bloomberg. Adding linguistic constraints to document image decoding: Comparing the iterated complete path and stack algorithms. In *Document Recognition and Retrieval VIII, Proceedings of the SPIE*, January 2001.

[98] G.E. Kopec. Advanced structured document examples. http://elib.cs.berkeley.edu/kopec/.

[99] G.E. Kopec. "General Comparison of Water District Acts", an advanced structured document created using a document-specific image decoder. http://elib.cs.berkeley.edu/kopec/b155/html/home.html.

[100] P.A. Chou. Recognition of equations using a two-dimensional stochastic context-free grammar. In *Visual Communications and Image Processing IV, Proc. of the SPIE*, volume 1199, pages pt.2:852–63, 1989.

[101] H.M. Twaakyondo and M. Okamoto. Structure analysis and recognition of mathematical expressions. In *Proc. of the Third International Conference on Document Analysis and Recognition*, volume 1, pages 430–7, Los Alamitos, CA, 1995. IEEE Comput. Soc. Press.

[102] D. Blostein and A. Grbavec. Recognition of mathematical notation. In H. Bunke and P. Wang, editors, *Handbook of Character Recognition and Document Image Analysis*, pages 557–582, Singapore, 1997. World Scientific.

[103] R.J. Fateman, T. Tokuyasu, B.P. Berman, and N. Mitchell. Optical character recognition and parsing of typeset mathematics. *J. of Visual Communication and Image Representation*, 7(1):2–15, March 1996.

[104] G.E. Kopec, P.A. Chou, and D.A. Maltz. Markov source model for printed music decoding. In *Document Recognition II, Proc. of the SPIE*, volume 2422, pages 115–25, 1995.

[105] G.E. Kopec and P.A. Chou. Automatic generation of custom document image decoders. In *Proc. of the Second International Conference on Document Analysis and Recognition*, pages 684–7, Los Alamitos, CA, 1993. IEEE Comput. Soc. Press.

[106] G.E. Kopec. Document image decoding in the UC Berkeley digital library. In *Document Recognition III, Proc. of the SPIE*, volume 2660, pages 2–13, 1996.

[107] A.C. Kam and G.E. Kopec. Separable source models for document image decoding. In *Document Recognition II, Proc. of the SPIE*, volume 2422, pages 84–97, 6-7 Feb. 1995.

[108] A.C. Kam and G.E. Kopec. The iterated complete path algorithm. In *Proc. of the Third International Conference on Document Analysis and Recognition*, volume 2, pages 1088–91. IEEE, 14-16 Aug. 1995.

[109] G.E. Kopec, P.A. Chou, and D.A. Maltz. Markov source model for printed music decoding. *Journal of Electronic Imaging*, 5(1):7–14, Jan. 1996.

[110] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation.* Addison-Wesley, Reading, MA, 1979.

[111] Dick Grune and Ceriel J.H. Jacobs. *Parsing techniques : a practical guide.* Ellis Horwood, New York, 1990.

[112] György E. Révész. *Introduction to formal languages.* McGraw-Hill, New York, 1983.

[113] Eugene Charniak. *Statistical language learning.* MIT Press, Cambridge, MA, 1993.

[114] Jesse F. Hull. Recognition of mathematics using a two-dimensional trainable context-free grammar. Master's thesis, MIT, Cambridge, MA, June 1996.

[115] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon limit error-correcting coding and decoding: Turbo codes. 1. In *Proc. of ICC' 93*, volume 2, pages 1064–70, New York, 1993. IEEE.

[116] Brendan J. Frey. *Graphical models for machine learning and digital communication.* The MIT Press, Cambridge, MA, 1998.

[117] Chris Heegard and Stephen B. Wicker. *Turbo coding.* Kluwer Academic Publishers, Boston, 1999.

[118] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference.* Morgan Kaufmann, San Francisco, 1988.

[119] Y. Weiss and W.T. Freeman. On the optimality of solutions of the max-product belief propagation algorithm in arbitrary graphs. *IEEE Transactions on Information Theory*, 47(2):736–44, Feb 2001.

[120] T. A. Tokuyasu. unpublished.

[121] M. Tomita. Parsing 2-dimensional language. In *International Workshop on Parsing Technologies, Pittsburgh, PA, USA, 28-31 Aug. 1989*, pages 414–24. Carnegie Mellon Univ, 1989.

[122] Alan Conway. Page grammars and page parsing. a syntactic approach to document layout recognition. In *Proc. of 2nd International Conference on Document Analysis and Recognition - ICDAR '93*, pages 761–4, Los Alamitos, CA, 1993. IEEE Comput. Soc. Press.

[123] F.R. Kschischang and B.J. Frey. Iterative decoding of compound codes by probability propagation in graphical models. *IEEE Journal on Selected Areas in Communications*, 16(2):219–30, February 1998.

[124] R.J. McEliece, D.J.C. MacKay, and Jung-Fu Cheng. Turbo decoding as an instance of pearl's belief propagation algorithm. *IEEE Journal on Selected Areas in Communications*, 16(2):140–52, February 1998.

[125] D. Miller, K. Rose, and P.A. Chou. Deterministic annealing for trellis quantizer and hmm design using baum-welch re-estimation. In *Proc. of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP '94)*, volume 5, pages V/261–4. IEEE, 1994.

[126] UC Berkeley Digital Library Project Source Code. http://elib.cs.berkeley.edu/src.

[127] Irfan Skiljan. Irfanview32 freeware graphic viewer. http://www.irfanview.com/.

[128] Sun Microsystems Inc. et al. Java Advanced Imaging API. http://java.sun.com/products/java-media/jai/.

[129] A.C. Kam and G.E. Kopec. Heuristic image decoding using separable source models. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '94)*, volume 5, pages V/145–8, 19-22 April 1994.

[130] Carson Cumbee. Line by line script identification. In *Proc. of the Symposium on Document Image Understanding Technology (SDUIT01)*, pages 23–29, College Park, MD, 2001. UMIACS.

[131] R. Phelps, T.A.and Wilensky. Multivalent documents. *Communications of the ACM*, 43(6):82–90, June 2000.

[132] Xerox Research Centre Europe MLTT. Finite state networks. http://www.xrce.xerox.com/research/mltt/fst/fsnetwork.html.

[133] Azaria Paz. *Introduction to probabilistic automata.* Academic Press, New York, 1971.

[134] Kevin Murphy. Learning finite automata. Technical Report 96-04-017, Santa Fe Institute, 1996.