SRI International

THE RELAX IMAGE RELAXATION SYSTEM: DESCRIPTION AND EVALUATION

Technical Note 301

August 1983

By:

Kenneth I. Laws, Computer Scientist
Grahame B. Smith, Computer Scientist

Artificial Intelligence Center
Computer Science and Technology Division

Program Development by:

Russell C. Smith
Joseph A. Pallas

The University of Maryland

SRI Project 1009

The work reported herein was supported by the Defense Advanced Research Projects Agency under Contract No. MDA903-79-C-0588.

SRI International®

333 Ravenswood Ave. • Menlo Park, CA 94025
(415) 326-6200 • TWX: 910-373-2046 • Telex: 334-486

Foreword

The primary purpose of the Image Understanding (IU) Testbed is to provide a means for transferring technology from the DARPA-sponsored IU research program to DMA and other organizations in the defense community.

The approach taken to achieve this purpose has two components:

(1) The establishment of a uniform environment that will be as compatible as possible with the environments of research centers at universities participating in the IU program. Thus, organizations obtaining copies of the Testbed can receive a flow of new results derived from ongoing research.

(2) The acquisition, integration, testing, and evaluation of selected scene analysis techniques that represent mature examples of generic areas of research activity. These contributions from participants in the IU program will allow organizations with Testbed copies to immediately begin investigating potential applications of IU technology to problems in automated cartography and other areas of scene analysis.

The IU Testbed project was carried out under DARPA Contract No. MDA903-79-C-0599. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the United States government.

This report describes the RELAX relaxation package contributed by the University of Maryland and presents an evaluation of its characteristics and features.

Andrew J. Hanson
Testbed Coordinator
Artificial Intelligence Center
SRI International

Abstract

RELAX is a system of routines that modifies the probabilities associated with labels attached to the elements of a two-dimensional array. These modifications reflect the compatibility of each element's labels with those of its neighbors. The initial probability assignments are usually derived from local property values in the neighborhood of each pixel. The final assignments may be used for object detection or segmentation, or may be mapped back to image intensities to achieve noise suppression, enhancement, or segmentation.

The relaxation package was contributed to the ARPA/DMA Image Understanding Testbed at SRI by the University of Maryland. This report summarizes applications for which RELAX is suited, the history and nature of the algorithm, details of the Testbed implementation, the manner in which RELAX is invoked and controlled, the type of results that can be expected, and suggestions for further development.

Table of Contents

Section 1

Introduction

The RELAX package is an interactive system of routines for mapping digital image data into a probability network, modifying the probabilities to reflect local constraints, and mapping the information back to the luminance domain. It is currently configured for image enhancement and object detection, but has many other applications.

Code modules and test data for the RELAX system were provided by the Computer Vision Laboratory at the University of Maryland (UM). The UM relaxation routines are configured as a set of stand-alone programs collectively called GPSPAR: General-Purpose Software Package for Array Relaxation. This package was originally written in the C language by Russel C. Smith and Joseph A. Pallas at the University of Maryland.

The current RELAX program is a command interpreter that interactively invokes the GPSPAR routines. This version of RELAX was constructed for the ARPA/DMA Image Understanding Testbed at SRI International by Kenneth Laws. The underlying command interpreter is the CI subroutine provided to the Testbed by Carnegie-Mellon University (CMU).

Many of the user-interface and image access routines were also contributed by CMU. Particular credit is due to Steven Shafer for the CI command interpreter and related string manipulation routines, to David Smith for the image access software, and to David McKeown, assisted by Jerry Denlinger, Steve Clark, and Joe Mattis, for the Grinnell display software. Kenneth Laws at SRI adapted this C-language software for Testbed use and interfaced it with the University of Maryland contributions.

No changes were required in the relaxation algorithm itself. The information in this document should thus be considered supplementary to the material cited in the UM references.

This document includes both a user's guide to the RELAX system and an evaluation of the algorithm. Section 2 explains the nature and use of the system in the context of typical applications. Section 3 surveys the historical development of the technique and presents the current algorithms in detail. Section 4 describes the Testbed implementation of this package and suggests some possible improvements. Section 5 instructs the user in the mechanics of using the RELAX software. Section 6 documents the performance that may be expected in various circumstances and presents the results of evaluation tests. Section 7 outlines a number of suggestions for improving the algorithm. Section 8 is a very brief summary. Appendix A shows how to invoke the RELAX routines in the manner of the original GPSPAR package submitted by UM.

Section 2

Background

This section presents a management view of the RELAX program. The relaxation algorithm is briefly sketched. Typical applications and potential applications requiring further development of the algorithm are discussed, and related applications for which other algorithms are better suited are noted.

2.1. General Description

The RELAX package for digital image enhancement and analysis is based on a class of algorithms for iteratively modifying vectors of probability values associated with the pixels of a two-dimensional array. This competitive-cooperative relaxation process strengthens compatible relationships and suppresses incompatible ones.

The RELAX algorithms also illustrate methods of propagating global interpretations and constraints through a network by local updating of the node interpretations. Such operations show promise for implementation on parallel arrays of processors and other advanced architectures.

An extensive literature connects the basic relaxation methods with numerous application areas. Much of the literature discusses relaxation on arbitrary graph structures rather than rectilinear data grids. The RELAX package, however, is aimed specifically at image-based applications.

The user starts with an array of values or a digital image, typically a luminance image or the output of an image-processing operator. The value at each pixel (or a set of values from a neighborhood of each pixel) is converted to a vector of probabilities; each probability reflects the likelihood that the pixel should be assigned a particular semantic label. This conversion process depends on the user's goals and the pixel classes that are relevant to those goals. In some formulations of relaxation, usually using different rules for adjusting the vectors at each pixel, the vectors can be regarded as representing fuzzy-set memberships [Zadeh65, Kandel78] rather than probabilities. In this report, for simplicity, we shall regard the vectors as probability vectors and and call the vector-valued image a probability image.

Next the user selects a relaxation method, a neighborhood size, and a set of compatibility coefficients. The compatibility coefficients are typically generated automatically from the initial probability image. This will be discussed in more detail in Sections 3 and 5.

The user then initiates one or more "relaxation steps," adjusting the probability vector at each pixel in accordance with the compatibility relationships and the probability vectors at each of its neighboring pixels. The definition of "neighbor" is supplied by the user; it must be the same for each pixel.

The resulting probability vectors are typically mapped back to the luminance domain

so that the user may observe the effect of the relaxation. This is not strictly necessary; a halting criterion based on the probability domain may be employed. The final probability image may or may not be mapped back to a luminance image, depending on the needs of the user.

Relaxation is a philosophically attractive procedure that seeks a globally consistent interpretation through local processing. Relaxation is still in the early stages of development and needs further research to determine the nature and range of its future applications.

2.2. Typical Applications

The RELAX program may be used in any application requiring noise suppression or feature reinforcement. The results of an image operation, such as edge detection, can be "smoothed" and detection reinforced. These effects, useful in themselves, may be precursors to further analysis.

The RELAX package is primarily adapted to specific applications by the mapping functions that convert luminance images to probability images. Very few such mappings are currently available with the package. Those that have been provided are suitable for the following purposes:

* Requantization--Reduction of the number of gray levels in an image typically introduces visible false edges in areas of smooth gradient. Relaxation may be used to pull pixels near the quantization threshold into the next higher or lower gray level. This will reduce false contours and act as a segmentation technique if the relaxation tends to group pixels that are within the same imaged object. (Adding a random dither signal to the image prior to requantization would also reduce false contouring, but would degrade the image and any subsequent segmentation of it.)

* Histogram Sharpening--There have been several schemes for iteratively replacing pixel values by some function of neighboring values in order to sharpen the peaks of the image histogram [Rosenfeld78, Peleg78b, Bhanu82]. Repeated applications can be used to merge smaller histogram peaks into larger ones until only a set number remain. (This differs from requantization in that the resulting gray levels need not be equally spaced.) Histogram sharpening is sometimes used as a precursor to image segmentation or compression.

* Smoothing--Relaxation can be used to smooth image regions to reduce noise artifacts. The smoothing can be done without blurring region edges if adjacent regions are mapped fairly well into different *a priori* labels. (Edge-preserving smoothing without such conditions requires special-purpose techniques that test region homogeneity before applying the compatibility correction. A median filter works this way.)

* Edge Enhancement--Relaxation can be used to sharpen region boundaries while smoothing the interiors. (Here, too, special-purpose algorithms that include decision logic might have better success than a linear summation of compatibility constraints.) Relaxation can also be applied to a gradient image to enhance extended discontinuities and suppress noise spikes.

* Linear-Feature Enhancement—This is essentially the same as edge enhancement, although more sophisticated feature detectors and classification operators might be involved.

* Detection—It is a small step from enhancing a feature to detecting it. Relaxation can help by reinforcing detection of groups of similar pixels while suppressing detection of isolated noise points. Other methods may be more advantageous for detecting objects larger or smaller than a few pixels.

* Pixel Classification—The source class to which a pixel is assigned may be adjusted by using the classifications and arrangement of its neighbors. Iteration of this process can reduce the effect of texture on classification. The RELAX package supplied by the University of Maryland contained a demonstration of two-class segmentation by thresholding, using an infrared image of a military tank. Segmentation by pixel classification into multiple classes using relaxation is described by Eklundh *et al.* [Eklundh80]. The anomaly detection experiments documented in Section 6 are also related to pixel classification.

2.3. Potential Extensions

The following applications might be feasible if the RELAX package were modified, used in a nonstandard fashion, or integrated into a more sophisticated system.

* Clustering—Clusters of points in a metric space can be detected by allowing each point to "gravitate" toward its neighbors. This is the spatial analogue of histogram sharpening. It requires a graph-based relaxation algorithm instead of the image-based RELAX updating algorithm.

* Semantic Labeling--Relaxation can be used to derive consistent sets of names or interpretations for regions in a scene. This also requires a graph-based relaxation method. A similar application is the identification of mixed pixels, noise regions, and border slivers in segmented images.

Such applications have been described in numerous papers, and there have been numerous other applications of relaxation techniques to image processing [Rosenfeld77b, Rosenfeld82, Rosenfeld83].

2.4. Alternative Approaches

This section describes applications that are similar to RELAX applications, but which differ in some fundamental fashion. While the difficulties with applying RELAX might be overcome, other techniques would often be more appropriate.

* Noise Suppression—Despite the applicability of relaxation to smoothing, the updating algorithms in the RELAX package have no underlying model of image and noise characteristics. Image noise can be more effectively removed by filtering techniques based on the noise statistics.

Background

* Restoration—Similarly, blur and other degradations are best removed by techniques that model the degradation process. The RELAX package can be used for limited classes of image enhancement, but usually at the cost of introducing less visible degradations elsewhere.

As a rule, relaxation methods work best in those applications requiring "gravitational" or "fluid flow" solutions, such as histogram sharpening or image smoothing. They might also be useful for enhancement applications that can be cast as "reverse fluid flow" problems. They generally do not work as well as model-based restoration or analysis methods when there exist underlying models of the scene and the image formation process.

Section 3

Description

This section presents the history of relaxation for image processing and a detailed statement of the algorithms used in the RELAX package. The historical information is intended to clarify the major issues in iterative image processing and to guide the reader to the relevant literature.

3.1. Historical Development

Relaxation methods are iterative procedures designed to seek adequate solutions to problems that defy analytic analysis. Relaxation advances toward a solution state (*e.g.*, a fully segmented image) step by step, instead of solving for the optimal fixed-point solution (as is common in the image restoration literature). Either the user stops the process or an automatic halting criterion is invoked when the remaining errors are sufficiently small. The operation is similar to hill climbing, combinatorial optimization [Kirkpatrick83], or stochastic approximation approaches.

Relaxation methods have long been used in physics and engineering, particularly in computational fluid dynamics, aerodynamics, and thermodynamics. Systems of ordinary and partial differential equations are commonly solved by approximate numerical methods. Some finite-element techniques propagate local constraints through a field in a single pass; other techniques are iterative.

Relaxation techniques may have entered the image-understanding literature through constraint satisfaction networks used to label line drawings [Guzman68, Waltz72, Haralick79]. The early procedures assumed all possible labelings for each adjacent line pair in the scene, then eliminated incompatible label pairs. Convergence was very rapid, but these methods had no mechanism for handling probabilities or uncertain evidence.

Constraint networks were later generalized to many image-understanding and expert-system applications [Montanari74, Hart77, Rosenfeld77b], particularly to the consistent labeling of scene regions [Yakimovsky73, Barrow76, Freuder76, Faugeras81]. This movement merged with the development of iterative techniques for texture segmentation and identification [Troy73], image region growing and merging [Brice70, Yakimovsky76, Zucker76], image smoothing and enhancement [Davis77a, Lev77], histogram modification [Rosenfeld77a], edge detection [Eberlein76, Schachter76] linear-feature enhancement [Riseman77, Zucker77, VanderBrug77]. curve segmentation [Davis77b], shape matching [Davis77c], and other applications.

The result, generally called relaxation labeling, is a set of iterative probabilistic approaches that consolidate many applications in the above areas. Probabilistic labeling, continuous relaxation, and stochastic labeling are other names for these techniques. All involve the application of local constraints to vary the weights (or degrees of belief) of semantic labels attached to the nodes of a graph. Iteration of the local adjustments, either in sequence or in parallel, are presumed to drive the

network closer to a global optimum or to a fixed point of the relaxation (*i.e.*, a state unaffected by further iterations).

The original relaxation labeling algorithm was proposed by Hummel, Zucker, and Rosenfeld at the University of Maryland [Rosenfeld76, Hummel78] and was further developed by a number of other researchers [Peleg78a, Zucker78a, Zucker78b, Yamamoto79, Haralick80, Hummel80, O'Leary80]. This method makes use of additive updating formulas. A later approach, based on multiplicative updating, was also developed at the University of Maryland [Kirby80, Peleg80a].

Many researchers have offered evaluation and discussion of the limits of relaxation processing [Kirby80, Kitchen80, O'Leary80, Richards80, Fekete81, Diamond82, Nagin82, Haralick83]. Faugeras and Berthod have suggested an alternative relaxation labeling philosophy [Faugeras80a, Faugeras80b], and this in turn has been criticized [Hummel80].

The papers cited above generally discuss relaxation in the abstract, although numerous applications have been developed [Rosenfeld82, Rosenfeld83]. The original papers on the Hummel-Zucker-Rosenfeld and Peleg methods [Rosenfeld76, Peleg80a] still constitute a good introduction to the philosophy of relaxation.

3.2. Algorithm Description

While the RELAX program provides a stand-alone implementation of relaxation, there are other ways of implementing it. Often the relaxation algorithm is so integrated with other techniques that it would be difficult to isolate the "relaxation part" of a procedure. Relaxation is a philosophy; no one algorithm embodies its alternate formulations. The two procedures included in the RELAX program (Hummel-Zucker-Rosenfeld, or HZR, and Peleg) are representative of the algorithms used in most applications of relaxation.

3.2.1. General Approach

Most image-based relaxation procedures comprise the four stages listed below; in rare circumstances one of the stages may be skipped. The stages are essentially input, construction of the relaxation operator, relaxation *per se*, and output. Each stage significantly influences the effectiveness of the overall relaxation procedure.

The four stages of the relaxation process are as follows:

- *Image-to-Probability Mapping*

  An operator is applied to the image array to convert the luminance values (or local property values, *etc.*) to probability vectors. Each pixel is assigned a vector representing an arbitrary set of semantic labels. Numeric values of the vector elements may be probabilities, likelihoods, or other measures of belief in the applicability of the corresponding labels at that image point. It is this mapping that adapts the relaxation paradigm to a specific application. The RELAX package currently contains illustrative mapping functions for image smoothing and edge detection applications. The user will generally have to supply new mapping routines for these or other applications.

Description

- *Compatibility Computation*

  Coefficients of compatibility between labels on neighboring pixels are computed, usually, from the weighted frequencies of label adjacencies in the original probability image. These coefficients essentially define the local relaxation operations that will be applied to the probability image. The RELAX package contains one routine for estimating HZR coefficients and another for Peleg coefficients; the values may also be supplied manually.

- *Relaxation Updating*

  The compatibility coefficients and updating rule are used to modify the probability vector at each pixel in turn. The values in each vector are adjusted by a weighted sum or product of values at neighboring pixels to enhance compatible combinations and suppress incompatible ones. The user may call for one or more passes of the relaxation operator through the probability image; current methods do not have halting criteria to determine when the updating should terminate.

- *Probability-to-Image Mapping*

  Relaxation methods may be designed to derive one or more numbers per pixel; these may be image luminances, edge probabilities, object likelihoods, segmentation maps, or other interpretations. The user must supply a mapping procedure to convert the multivariate probability image to this form. Routines currently in the RELAX package can produce binary and gray-level luminance images useful for edge or object detection as well as for image enhancement.

We shall now present these stages in detail.

3.2.2. Image-to-Probability Mapping

Luminance images used as input to a relaxation procedure must first be converted into probability image form. The method supplied in the RELAX package *imgprb* command is described here. It is a linear mapping of image brightness to a vector of probability values representing our belief that the pixel belongs in particular luminance "level slices." This mapping might be suitable for image binarization or requantization, noise suppression, and object detection applications. It is provided only as an example and as a mechanism for verifying the correct operation of the relaxation updating software; other mapping functions will be needed for other applications.

Among the arguments to *imgprb* are the low and high gray-level values and the number of labels to use. Pixel values are clipped to lie within the specified range and are then mapped to vectors of probability values between 0.0 and 1.0. The mapping function depends on the number of labels, as discussed below.

For binary output, the label probabilities may be thought of as positive and negative support for the hypothesis that a bright object is the pixel source. These are represented by two floating-point numbers per pixel, $p[0]$ and $p[1]$, where the indices represent the two classes or pixel labels. The first value, $p[0]$, represents

our belief that the pixel is from a light "object" area; the second, $p[1]=1.0-p[0]$, our belief that it is from a dark background population. The lowest pixel value in the specified range thus maps to $p[0]=0.0$ and $p[1]=1.0$; the highest to $p[0]=1.0$ and $p[1]=0.0$. Linear interpolation is used for intermediate pixel values.

If the user requests more than two labels, the labels may be thought of as level slices and the vector elements as probabilities that a pixel should be assigned to one of these levels. The lowest pixel value maps to $p[0]=1.0$, the highest to $p[$*maximum label*$]=1.0$. Linear interpolation between the bracketing labels is used for intermediate pixel values. At most two labels will have nonzero probabilities with this conversion scheme.

3.2.3. Compatibility Computation

A compatibility coefficient must be specified for each combination of a label at the central pixel with each label at each neighboring pixel. The values of the coefficients depend on the updating method to be used and also, as a rule, on the initial probability image data.

The neighborhood of a pixel is usually taken to be the set of pixels in a small surrounding square, with the size of the square selectable by the user. Neighborhoods restricted to only horizontal or vertical neighbors or that include nonadjacent pixels are sometimes required; the *defnbr* routine allows such arbitrary neighborhoods to be defined.

Compatibility coefficients used for the Hummel-Zucker-Rosenfeld relaxation scheme are different from those used for its Peleg counterpart. In the additive HZR scheme [Hummel78, Peleg78a, Yamamoto79], coefficients are negative if the labels are incompatible, zero if they are independent, and positive if the labels are compatible. Coefficient values are restricted to the range $[-1.0, +1.0]$. They typically range from -0.1 to about 0.5 when computed with the RELAX *hcompat* routine.

In the multiplicative Peleg scheme [Peleg80a, Haralick83], all the coefficients are nonnegative. Coefficients are less than unity if the labels are incompatible, unity if they are independent, and greater than unity if the labels are compatible. They typically range from near zero to about 5.0 when computed with the RELAX *pcompat* routine.

Compatibility is a loosely defined term, and no definition to date has been entirely satisfactory [Haralick83]. The HZR compatibility coefficients are based on information theory [Peleg78a, Yamamoto79], the corresponding Peleg coefficients on conditional probabilities [Peleg80a]. Both methods utilize the *a priori* probabilities of labels at an arbitrary pixel, as measured in the initial probability image, to estimate the compatibilities.

Suppose we have a graph whose nodes are each to be labeled with one of the possible labels $\lambda_1, \lambda_2, \ldots, \lambda_k, \ldots, \lambda_n$. Further suppose that some measurement associated with each node has allowed us to state the *a priori* probability of that node being labeled with each of the possible labels. For node i we have $p_i(\lambda_k)$, $k=1, \cdots, n$, as the probability that node i has label λ_k.

If we were to label the ith node with the label λ_k, and if the graph showed that the ith node and the jth node are adjacent, then we need to determine whether the label λ_k on the ith node is compatible with the labels on the jth node. We specify

this compatibility with the coefficient $r_{ij}(\lambda_k,\lambda_l)$, which states the compatibility of label λ_k on the ith node with the label λ_l on the jth node.

For the HZR scheme, compatibilities may be calculated from the quantity

$$r_j(\lambda_k,\lambda_l) = \frac{1}{5}\ln\left\{\frac{w\sum_i p_i(\lambda_k)p_{i_j}(\lambda_l)}{\sum_i p_i(\lambda_k)\cdot\sum_i p_i(\lambda_l)}\right\} \qquad \begin{matrix} k=1,\cdots,n \\ l=1,\cdots,n \end{matrix}$$

where i ranges over all w nodes of the graph and j specifies the particular neighbor of the ith node. For each node i, the compatibility with its jth neighbor is then

$$r_{ij}(\lambda_k,\lambda_l) = \begin{cases} -1 & \text{if} \quad r_j(\lambda_k,\lambda_l) < -1 \\ r_j(\lambda_k,\lambda_l) & \text{if } -1 \le r_j(\lambda_k,\lambda_l) \le +1 \\ +1 & \text{if } +1 < r_j(\lambda_k,\lambda_l) \end{cases}$$

Note that the RELAX package currently uses the same values of the compatibility coefficients for all values of i, *i.e.*, for each pixel postion to be updated.

For the Peleg scheme, the compatibilities may be calculated from the quantity

$$r_j(\lambda_k,\lambda_l) = \frac{w\sum_i p_i(\lambda_k)p_{i_j}(\lambda_l)}{\sum_i p_i(\lambda_k)\cdot\sum_i p_i(\lambda_l)} \qquad \begin{matrix} k=1,\cdots,n \\ l=1,\cdots,n \end{matrix}$$

where i ranges over all w nodes of the graph and j specifies the particular neighbor of the ith node. For each node i, the compatibility with its jth neighbor is then

$$r_{ij}(\lambda_k,\lambda_l) = r_j(\lambda_k,\lambda_l)$$

Note that these Peleg compatibilities are in the range $[0,w]$.

It may sometimes be desirable to use ensemble statistics to compute the compatibilities. Only experience with a particular application allows coefficients to be chosen rather than calculated by formula.

3.2.4. Updating Formulas

The relaxation updating computations can now be presented in more detail.

The goal of the relaxation algorithm is to update the values of the probabilities associated with a node so as to take into account the compatibility of neighboring labels. A number of different schemes have been proposed to do this updating. The earliest was the HZR scheme [Rosenfeld76], in which the $(t+1)$ update of the probability values is calculated from the (t) values by the following rule:

Description

For node i,

$$p_i^{(t+1)}(\lambda_k) = \frac{p_i^{(t)}(\lambda_k)[1+q_i^{(t)}(\lambda_k)]}{\sum_{\lambda=\lambda_1}^{\lambda_n} p_i^{(t)}(\lambda)[1+q_i^{(t)}(\lambda)]} \qquad k=1,\cdots,n$$

$$q_i^{(t)}(\lambda_k) = \frac{1}{m}\sum_j \left\{ \sum_{\lambda'=\lambda_1}^{\lambda_n} r_{ij}(\lambda_k,\lambda')p_j^{(t)}(\lambda') \right\} \qquad k=1,\cdots,n$$

where j indexes the m neighbors of node i, and $r_{ij}(\lambda_k,\lambda')$ is the compatibility coefficient for node i with label λ_k and neighboring node j with label λ'.

The Peleg relaxation scheme [Peleg80a], also included in this package, uses the updating rule

$$p_i^{(t+1)}(\lambda_k) = \frac{1}{m}\sum_j \frac{p_i^{(t)}(\lambda_k)q_{ij}^{(t)}(\lambda_k)}{\sum_{\lambda=\lambda_1}^{\lambda_n} p_i^{(t)}(\lambda)q_{ij}^{(t)}(\lambda)} \qquad k=1,\cdots,n$$

$$q_{ij}^{(t)}(\lambda_k) = \sum_{\lambda'=\lambda_1}^{\lambda_n} r_{ij}(\lambda_k,\lambda')p_j^{(t)}(\lambda') \qquad k=1,\cdots,n$$

where j indexes the m neighbors of node i. (Actually, a slightly more general form of the Peleg scheme is implemented in the RELAX system; see Section 5.2 for details.)

In both of these schemes, $q_i(\lambda_k)$ (or $q_{ij}(\lambda_k)$) can be thought of as the neighboring node's assessment (by node j in the case of $q_{ij}(\lambda_k)$) that node i should be labeled λ_k; $p_i(\lambda_k)$ is the assessment by node i that its own label should be λ_k. These two assessments are combined to produce an updated probability.

Other forms of the updating rule based on optimizing measures that are functions of terms like the above p's and q's, have been employed [Faugeras80a, Faugeras80b, Hummel80]. While the development of these forms rests on a firmer foundation than that of the rules above, these newer rules also have defects [O'Leary80, Peleg80b]. Substantial theoretical work needs to be done to comprehend the nature of relaxation and to lay the groundwork for eliminating rule deficiencies in the future.

The foregoing description has been couched in terms of the probability, likelihood, certainty, or favorability of assigning a particular label to a node. It is useful to think of the associated numeric value as the probability that the node has the label, but there are theoretical problems with this interpretation. We shall adopt the convenience of referring to such values as probabilities—but it should be kept in mind that, strictly speaking, this may not be correct.

Relaxation is an updating rule for improving the initial assignment of labels by enforcing compatibility with neighboring labels. Unfortunately, compatibility is a poorly-understood notion. One does not know when a rule will converge, or, if it does, what its rate of convergence will be [Zucker78a]. Nevertheless, relaxation has been successfully applied to a number of different problems. Relaxation

captures the ideas that we should be able to label objects so that they are compatible with their neighbors, and that we should be able to do this through local processing [Ullman79]. Relaxation requires a solid theoretical base [Hummel78, Hummel80] to define its domain of applicability.

3.2.5. Probability-to-Image Mapping

When the relaxation system is used for object cueing, the matrix of $p[0]$ values may be the desired output. In other cases, it may be desirable to map the probability vectors back to luminance gray levels so that the output can be displayed. This mapping is typically the inverse of that used to convert a luminance image to probability form. The RELAX package *prbimg* routine is the inverse of the *imgprb* mapping described earlier.

Inversion of the image-to-probability mapping is complicated by the fact that the "probability" vectors for a pixel are not always normalized and need not sum to 1.0. The inverse mapping function supplied by the University of Maryland uses different resolutions of this problem in the two-label and multilabel cases.

The *prbimg* routine will convert a two-label probability image to a gray-scale image whose values quantify the "strength of belief" in the $p[0]$ hypothesis represented by the stronger of the two probabilities. Thus, $p[0]$ values are converted directly to pixel values; $p[1]$ values, if stronger, are subtracted from 1.0 before conversion to pixel values. The gray-level interpolation inherent in this procedure produces an image quantized to an arbitrary number of gray levels, usually 256.

A multilabel probability image will be converted to a gray-scale image with values representing the label, or luminance-level slice, with the highest probability. Thus, a strongest $p[maximum\ label]$ maps to the highest pixel value and a strongest $p[0]$ maps to the lowest; intermediate labels map to intermediate gray levels. There is no interpolation between gray levels, so the output image is quantized to the number of labels used.

Section 4

Implementation

This section documents the SRI Testbed implementation of RELAX. It is intended as a guide for system maintainers and for programmers modifying the RELAX system. The terms used in this section are either defined elsewhere in this report or come from the supporting operating systems. The SRI Testbed uses the EUNICE operating system, which is a Berkeley UNIX[1] emulator for VAX computers using DEC's VMS operating system. All of the relaxation software will also run on a pure UNIX system.

The RELAX package is currently compiled as an interactive driver program. The driver invokes other programs for most of its work, although it does call subroutines directly to enable conversion between intensity and probability formats. The computational algorithm is very little changed from the original University of Maryland version, but the command interpreter, help system, and supporting image-access and display utilities are all new.

The main program and related files are in directory */iu/tb/src/relax*. Major subdirectories are

cvlhdrlib	- probability image subroutines;
defcom	- *defcom* source code;
defnbr	- *defnbr* source code;
demo	- shell script for the tank demo;
help	- help system text files;
imgprb	- *imgprb* source code;
prbimg	- *prbimg* source code;
relax	- *relax* main program source code;
relaxpar	- *relaxpar* source code (used by *setup*);
src/hummel	- *hcompat* and *hrelax* source code;
src/peleg	- *pcompat* and *prelax* source code.

Compiled versions of these main programs are kept in */iu/tb/bin*. The Hummel and Peleg operators are not kept in compiled form, since they are intended to be customized for each application.

Cvlhdrlib is a library of subroutines for manipulating the floating-point probability files. (It is expected that someday this function will be absorbed by the Testbed image-accessing code.)

The *imgprb* and *prbimg* programs simply parse their command-line arguments and pass the information to subroutines. The RELAX driver likewise parses commands given to it and invokes the same subroutines. These subroutines are currently stored in directory */iu/tb/lib/visionlib*, specifically in the *relaxlib* subdirectory.

[1]UNIX is a trademark of Bell Laboratories.

Implementation

Source code and help files for the CI driver are in */iu/tb/lib/cilib*. For extensive documentation, type "man ci" or run "vtroff -man /iu/tb/man/man3/ci.3c". The CI driver uses command-line parsing routines in *cilib/cmuarglib* and in */iu/tb/lib/sublib/asklib;* both of these may someday be replaced by the Testbed argument-parsing routines in *sublib/arglib*.

Other utility routines contributed by CMU have been distributed to */iu/tb/lib/dsplib/gmrlib*, */iu/tb/lib/imglib*, and */iu/tb/lib/sublib*, and are documented for the **man** system in */iu/tb/man/man3*. Some of these have been modified or rewritten for the Testbed environment: the image access code, for instance, reads Testbed image headers in addition to CMU image headers. Other routines in these directories were developed at SRI.

To recompile one of the relaxation routines, *e.g.*, *imgprb*, just connect to the appropriate source directory and type "make". You may type "make -n" to see what will happen if you do this. Additional options are documented in the header of the makefile. To compile and install the entire system, run the *make* program in */iu/tb/src/relax*. For more flexible maintenance options, see the header sections of the corresponding *makefile* files.

The *hcompat* and *hrelax* programs and their Peleg equivalents are normally compiled interactively by using the *setup* command of the RELAX package. Implementation of this capability requires that the source file locations of these files be known to the RELAX program. This program must therefore be modified and recompiled any time the relaxation source files are moved.

RELAX demonstrations have been set up in subdirectories *relax* and *tank* of */iu/testbed/demo*. Just connect to the appropriate directory and run the *demo* command. Afterwards you may want to run the *cleanup* script stored in that directory to delete the relaxation output files.

The UM code represents an interesting style of programming and usage. Two points should be noted:

* Each routine is compiled as a separate program. Commands are passed to a command interpreter or to the UNIX shell to invoke the programs in the proper sequence. One benefit is that any routine can be altered without recompiling and linking the entire system. (Another possibility, not implemented here, is to pipe the routines together so that each feeds its output to the next. This would eliminate many of the intermediate files now being created by the system.)

* One of the steps in a relaxation sequence can be the construction and compilation of a special-purpose relaxation operation. This is currently done by the *setup* routine, which uses an *include* file built by *relaxpar* to tailor the *hcompat* and *hrelax* programs (or their Peleg equivalents) to the desired neighborhood definition. Such operators can run faster than general-purpose ones that use run-time evaluation of conditionals to control execution logic.

We have attempted to retain this style of programming while still packaging the relaxation system in a form similar to that of other major software systems on the Testbed. We have retained the concept of separate compilation so that the shell script in Appendix A will execute properly on the Testbed. Those who prefer such a system are free to

make use of it.

We have also written an interactive driver package, known as RELAX, for invoking the routines in a more structured environment; this allows for easy incorporation of customized syntax, argument defaulting, global status variables, help functions, and other "intelligent" session control features. (Very few such features are now implemented, but the possibilities can be seen in other Testbed software using the CI driver mechanism.)

Various problems were encountered in integrating the original package with the IU Testbed and in documenting the result. Sometimes it was easier to change the user interface slightly than to document an inconsistency. Among the changes made in the original system are the following:

* *Name Changes*

  The program to convert images to a probability format was originally named *init*, and the program to convert them back again was named *display*. We have changed these names to *imgprb* and *prbimg*, respectively. The main programs now invoke subroutines to do most of the work; we have called these *imgprbsub* and *prbimgsub*.

* *Conversion to Testbed Formats*

  The original *imgprb* and *prbimg* routines accepted images no wider than 512 pixels. We have removed this restriction. The pixels were also limited to 8 bits, or 256 gray levels. We have extended this range to the 36-bit pixels currently handled by the Testbed image access software. Testbed images of unusual pixel lengths, *e.g.*, 3 bits, are supported directly, as opposed to the UM practice of padding them into 8-bit fields with a "significant bits per pixel" specification to recover the dynamic range information.

* *Generalization to Multiple Labels*

  The original version of *imgprb* assumed that only two labels were to be used, although the rest of the package did accept more than two labels. We have extended the mapping algorithm as described in the previous section. The mapping to multiple labels was chosen to be the inverse of the mapping back from multiple labels that was already implemented in *prbimg*.

  Prbimg accepted a "number of labels" argument, but then ignored it, since this information could be more reliably obtained from the header of the probability file. The demonstration shell script supplied with the original package erroneously omitted this argument in calls to the routine. This has been fixed by eliminating the interactive argument.

Implementation

* *Defaulted Arguments*

  The main programs are able to count the number of arguments passed to them and to substitute defaults for any missing arguments. We have retained this capability in the RELAX driver program. The invoked subroutines, however, must be supplied with a full set of arguments. We have adopted the convention that a negative minimum or maximum range specification passed to *imgprbsub* or *prbimgsub* will denote that the full dynamic range of the picture file is to be used. The user should specify non-negative values if stretching and clipping are desired.

* *Input Clipping*

  The *imgprb* routine originally stretched imagery to the specified gray-level range, but did not clip to this range. The mapping to a probability image was fine if the true image range was specified, but would generate probabilities that were negative or greater than unity for gray levels outside this range. The result of the inverse *prbimg* mapping on such a file was a sawtooth function. We have corrected this by clipping all probability values to the 0.0-1.0 range.

* *Word Size Conversion*

  The original package was written for a PDP-11 computer, for which the C language uses 16-bit integers. Our installation is on a VAX 11/780 that uses 32-bit integers. This difference is important in the writing and reading of neighborhood and compatibility file headers, since the I/O statements specified the number of bytes to be transferred. We have rewritten these sections to use the C *sizeof()* construct, which is guaranteed to be valid on any type of machine. The relaxation data files, however, cannot be transferred between machines with different integer sizes unless further standardization is implemented.

The need for additional changes became apparent during our software evaluation effort. Many of the needed improvements have to do with the command interpreter or the package philosophy rather than the relaxation algorithms. We suggest the following implementation changes:

* *Initialization Capability*

  The program should be able to run a startup file. This would allow the user to customize the package to his own preferences and tasks. Additional flexibility should be built into the command driver so that it could take advantage of initial profile information to set default neighborhood sizes and file names.

Implementation

* *Redefinition of Probabilities*

  The *imgprb* two-label mapping should be reversed to match its multilabel mapping interpretation. The *prbimg* inverse mapping should offer an option as to whether two-label probability files will be mapped to gray levels or to a binary output. Possibly a separate routine should be provided for each mapping and inverse mapping function instead of combining many functions in one routine.

* *Defcom Improvement*

  The *defcom* routine for interactively defining compatibility coefficients is very tedious to use. It is often easier to construct a file containing the coefficients and then pipe it into *defcom*, using the command interpreter's "<" facility for acquiring command input directly from a text file. If coefficients are provided in this manner and a neighborhood file is not specified, an inconsistency arises: the neighborhood size must be included at the start of the piped coefficients, even though they would not be needed if the coefficients were entered interactively. This should be changed so that the routine reading the coefficients does not expect to read the neighborhood size as well.

* *Run-Time Argument Passing*

  The command interpreter's "<" mechanism for invoking script files should accept arguments. The UNIX shell languages provide a good model for the type of argument macro expansion capability that is required.

* *Automatic Checkpointing*

  If automated iteration is ever added to the RELAX package, there should be some easy method of saving a checkpoint output every few iterations. Relaxation is such an expensive technique that a user should not have to start again from scratch if the system crashes or if processing has gone past the optimum point and begun to degrade the image.

* *File Name Flexibility*

  Part of the checkpointing problem is due to the current "hard-wiring" of the names *prb.img* and *compat.dat* into several of the routines. This causes the *hrelax* and *prelax* routines to overwrite the *prb.img* file, making it difficult to repeat an iteration (*e.g.*, with different parameters) or to recover after too many iterations. It is also difficult to remember exactly which iteration or processing sequence generated the current *prb.img* file. Although the UNIX/EUNICE hierarchical file system and the EUNICE multiple file-version facility alleviate some of these problems, the best solution is to allow arbitrary file names to be passed to the processing routines. An intelligent system for constructing default file names could also be helpful.

Implementation

* *Subroutine Implementation*

The RELAX routines are currently implemented as stand-alone main programs that can also be invoked from the RELAX driver. This differs from the subroutine-based implementation that is common to all other Testbed software. Although the main-program approach works, it is difficult to integrate with the rest of the Testbed. Our directory hierarchies and archiving techniques are based on libraries of subroutines rather than on clusters of main programs. It would be simpler to maintain a system that has a uniform programming philosophy.

As an example, consider the compilation of the *hrelax* routine. Ordinarily, each main program in the Testbed is accompanied by a *makefile* script that "remembers" all the *include* files and libraries needed in compiling the routine. If *hrelax* is to be created by the *setup.csh* script provided by UM, the compilation command must be in that script. If it is to be created by the RELAX program, the compilation command must be compiled into that code. Thus, changes in the structure or location of the *include* files and libraries must now be implemented by changes in several types of source code. While this is not difficult, it is certainly more trouble than updating a single type of file used consistently throughout the system.

The chief reason for using main programs rather than subroutines is that optimized relaxation operators are compiled for each specific task. This exchanges extra compilation time for reduced execution time—which seems reasonable, given the length of time currently required to run a relaxation step. Separately compiled subroutine modules could be linked into the driver package at run time, although this UNIX capability is not easily accessible or commonly used.

The shell languages do provide convenient mechanisms for sequencing programs, but they are better suited for batch execution than for interactive use. Furthermore, they are general-purpose and hence lack the focused environment, vocabulary, defaults, and help facilities that a dedicated command driver can offer. Command interpreters can invoke either main programs or subroutines, but are somewhat easier to write for the subroutine case.

Another benefit of subroutine-based implementation is control of interprocess communication. Programs invoked by shell scripts can communicate only via files. (Communication by means of shell environment variables is also possible, but not commonly done.) File passing is rather awkward, since it requires each main program to open, read, and parse every file. It also tends to clutter the environment with superfluous files; these can be deleted by commands at the end of a shell script, but must be removed by hand if the session is interactive or is aborted.

A more traditional subroutine-based programming style would allow communication by means of global variables and passed arguments as well as files. Display devices would also be under better control, since the device status can be remembered and maintained by the driver program, and each routine need not reallocate or reinitialize the display to be sure of getting a usable configuration.

In addition, the data files could be opened once, passed around, then closed

or deleted. Permanent files need be created only for permanent data, and the user need not specify the file names as arguments to every routine.

A final advantage of the subroutine approach is that there is less system overhead. The current approach requires that a UNIX (or EUNICE) process be created to run each command, since this is the only way to invoke a main program. The overhead in creating a process is much greater than that of calling a subroutine, and, in fact, may require several seconds of real time.

* *Operator Libraries*

As mentioned above, one advantage of the University of Maryland approach is the run-time compilation of customized image operators in order to reduce total execution time. The actual speedup achieved in the current RELAX package is rather small, but the technique could be extended for larger gains. Perhaps the greatest speedup could be achieved by using replicated in-line code instead of iterative constructs. For research purposes, of course, such optimizations are seldom worthwhile.

The run-time compilation of such routines is not a problem. It can be done as a separate program step, possibly invoked interactively by interrupting a session (with ^Y), compiling, and then resuming. It can also be done by using the "system" subroutine to call the compiler from within another program.

Since compilation is an expensive step, it might make sense to keep the most useful operators in a library of executable files. Nothing in the current RELAX package prevents this from being done, but neither are there any features to facilitate it. At the very least, a naming convention should be devised so that the operator names can be remembered.

* *Speedup*

The current relaxation updating algorithms are exceedingly slow. (They take about three CPU minutes for one pass of a 3×3 operator through a 128×128 image.) This is probably due to an inefficient scheme for accessing the floating-point label probabilities. Further investigation is needed to determine the requisite time for this type of processing.

Section 5

Program Documentation

This section constitutes a user's guide to the RELAX package as it is implemented on the SRI Image Understanding Testbed. As with any reference manual, it has occasionally been necessary to refer to terms before they are defined and discussed in detail. The first-time reader may find a preliminary scan through the section helpful. Additional information is available online, as described below.

5.1. Interactive Usage

There are typically five steps in applying relaxation to an image:

* Compilation of the updating routine
* Image-to-probability mapping
* Estimation of compatibility coefficients
* Relaxation updating iterations
* Probability-to-image mapping.

Display steps are usually interspersed so that one can watch the progress of the enhancement. Other techniques are sometimes required, such as edge detection or manual entry of neighborhood and compatibility data. All of these processes can be invoked from within the RELAX package.

The RELAX package currently takes no command-line arguments; just type "relax" and begin an interactive session. The following sample session displays some of the capabilities of the underlying CI driver language.

```
relax

  RELAX, Version 1.0

>
```

This invokes the program. You need not specify the full directory path name for the executable file if the path is given in your UNIX *.cshrc* shell startup file. (If you have no startup file, you may have to specify */iu/tb/bin/relax* or some other full path name.) The system then responds and waits for commands.

```
> *

      defcom        pcompat
      defnbr        prbimg
      erase         prelax
      hcompat       quit
      hrelax        setup
      imgprb        help
      insert
```

An "*" command lists all available commands. The *help* command is provided by the CI driver; all others are specifically related to the relaxation system. Typing "help" will give further information on the CI command interpreter and the help subsystem.

```
> help

Command names, variable names, and help topics
may be abbreviated to any unique prefix.  Several
instructions may appear on the same line, separated
by semicolons.  Use ^O to abort typeout and ^Z
to exit.

  command [ arg ... ]
    Execute a command with the specified arguments.

  foo*
    List commands that begin with "foo".
    Use just "*" to list all commands.

  foo* =
    List the names and values of variables
    that begin with "foo".

  variable [ param ... ]
    Display a variable value.  Some variables
    may require subscripts or parameters.

  variable [ param ... ] = value
    Assign a variable value.  The equals sign (=)
    may appear anywhere.

  ? topic  <or>  help topic  <or>  help *
    Print help message related to topic.  If topic
    is ambiguous, list the names of matching topics.

  push_level
    Creates a new level of the CI driver with the same
    commands available as there were at the preceding
    level.  The user may execute any combination of
    commands before quitting.

  ! command
    Fork a shell and execute the UNIX command.  If
    no command is given, just create an interactive
    shell process.

  < commandfile
    Read commands from "commandfile".

  : comment
    Accept a comment line; take no action.
```

As an example of the online help facility, we can print out the contents of the *setup.txt* file in the relaxation system help directory.

```
> help setup

setup {h|p} nlabels [ncols nrows]

  Setup is used to create programs for relaxation operations
  on images.  Both a compatibility operator and a relaxation
  operator will be compiled.  Either Hummel-Zucker-Rosenfeld
  (h) or Peleg (p) relaxation formulas may be chosen; the
  corresponding programs will be hcompat and hrelax or pcompat
  and prelax.  The default is "h".  You may also specify the
  number of class labels (default 2) to be used and the size
  of the relaxation neighborhood (default 3 x 3).
```

Running the *setup* command with default arguments produces a *param* file that is compiled into the *hrelax* and *hcompat* programs. The *param* file is then deleted and the executable programs are left in the current directory.

```
> setup

  Creating the Hummel-Zucker-Rosenfeld relaxation package ...

    Creating the "param" file ...
    Compiling hcompat ...
    Compiling hrelax ...
    Removing the param file ...
    Setup completed.
```

A UNIX directory listing command shows the new executable programs and a demonstration command file that was created previously.

```
> !ls

hcompat.exe     hrelax.exe      tank.cmd
  [continuing]
```

Here we use the command file to run a relaxation sequence. If the file were not in this directory, RELAX would search for it in directory */iu/tb/src/relax/demo*. RELAX then echos the commands and performs the indicated actions just as if they had been typed from the terminal.

```
> <tank.cmd
```

The image is first converted to two-label probability form. It has no pixel values below 13 or above 49, so stretching and clipping are requested. (This speeds convergence and improves the appearance of the output image.)

```
> : Convert the image to probability form.
> imgprb /iu/tb/pic/tank/bw.img 2 13 49
```

Next the compatibility coefficients are needed. They are computed from the pixel relationships in the original image. They could also be entered by hand with the *defnbr* and *defcom* commands. The program echos the *hcompat* request, with the default file names filled in.

```
> : Compute the compatibility coefficients.
> hcompat
  hcompat prb.img compat.dat
```

The original image is now to be reconstructed and displayed in an upper-left area of the screen. This reconstruction, which shows the input after stretching and clipping, also serves as a check on the *imgprb* and *prbimg* processes.

```
> : Display the (stretched) original image.
> erase
> prbimg output0.img
> insert output0.img 100 348
```

Now eight relaxation steps are to be run. Each will produce an output image; these images will be displayed along with the original in a 3 × 3 pattern.

```
> : Display eight relaxation steps.
> hrelax
  hrelax prb.img compat.dat

> prbimg output1.img
> insert output1.img 224 348

> hrelax
  hrelax prb.img compat.dat

> prbimg output2.img
> insert output2.img 348 348

> hrelax
  hrelax prb.img compat.dat

> prbimg output3.img
> insert output3.img 100 224

> hrelax
  hrelax prb.img compat.dat

> prbimg output4.img
> insert output4.img 224 224

> hrelax
  hrelax prb.img compat.dat

> prbimg output5.img
> insert output5.img 348 224

> hrelax
  hrelax prb.img compat.dat

> prbimg output6.img
> insert output6.img 100 100

> hrelax
  hrelax prb.img compat.dat

> prbimg output7.img
> insert output7.img 224 100

> hrelax
  hrelax prb.img compat.dat

> prbimg output8.img
> insert output8.img 348 100

End of command file.
```

The quit command is now given to return the user to the operating-system level.

```
> quit
```

5.2. Commands

The following commands are currently available within the RELAX system:

defcom compatfile relaxtype nlabels [neighborfile]
Defcom is an interactive program used to install manually computed compatibility coefficients for each neighbor of a point. The arguments specify the file to which the coefficients are to be written, whether a Hummel-Zucker-Rosenfeld ("h") or Peleg ("p") relaxation is desired, the number of labels in the relaxation process, and, optionally, a file that specifies a nonstandard neighborhood.

defnbr neighborfile ncols nrows
Defnbr is an interactive program used to define a nonstandard neighborhood for each point. The "neighborfile" argument specifies the file to which the neighborhood definition is to be written. The number of columns and rows that contain the neighborhood must also be specified. The program will ask which is to be considered the center point and whether each neighbor is to be considered as part of the neighborhood.

erase
Erase the display. Note that the display is not allocated or locked, so it will remain clear only if no one else transmits data to it.

hcompat prbfile compatfile [neighborfile]
Computes the Hummel-Zucker-Rosenfeld compatibility coefficients for the probability file (default *prb.img*) and stores them in a compatibility file (*compat.dat*). You may specify a neighborhood file as created by *defnbr*.

The *hcompat* program must have been compiled previously: see *setup*. If you would like to specify the compatibility coefficients by hand, see *defcom*.

hrelax prbfile compatfile [neighborfile]
Performs one Hummel-Zucker-Rosenfeld relaxation operation on a probability file (default *prb.img*), using compatibility coefficients in compatfile (*compat.dat*). You may specify a neighborhood file as created by *defnbr*.

The *hrelax* program must have been compiled previously: see *setup*.

imgprb inimg nlabels minval maxval
Convert an image to a probability format with the specified number of labels. The image will be clipped (stretched) using minval and maxval as the outer gray-level limits: omit these or specify -1 if the full input range is to be used. The file *prb.img* will be produced as output. At present it is a floating-point data file, not a true picture file.

insert picname mincol minrow
Insert the picture into the display at the specified lower-left pixel position.

pcompat prbfile compatfile [neighborfile]

Computes the Peleg compatibility coefficients for the probability file (default *prb.img*) and stores them in a compatibility file (*compat.dat*). You may specify a neighborhood file as created by *defnbr*.

The *pcompat* program must have been compiled previously: see *setup*. If you would like to specify the compatibility coefficients by hand, see *defcom*.

prbimg outname minval maxval

Convert a probability file to a luminance image format with the specified output range. Omit minval and maxval, or specify -1 to use the full (8-bit) dynamic range of the output image. The input file is assumed to be *prb.img*.

prelax prbfile compatfile [n neighborfile] [s nsets size1 ...]

Performs one Peleg relaxation operation on a probability file (default *prb.img*), using compatibility coefficients in compatfile (*compat.dat*). You may specify a neighborhood file as created by *defnbr;* precede it with the letter n.

You may also specify the grouping of label values into sets. Previously, in Section 3.2, we stated the Peleg updating rule as

$$p_i^{(t+1)}(\lambda_k) = \frac{1}{m}\sum_j \frac{p_i^{(t)}(\lambda_k) q_{ij}^{(t)}(\lambda_k)}{\sum_{\lambda=\lambda_1}^{\lambda_n} p_i^{(t)}(\lambda) q_{ij}^{(t)}(\lambda)} \qquad k=1, \cdots, n$$

where the denominator is the sum over the set of all labels. Actually, this sum can be limited to a set of labels, rather than summing over all labels. The sum is carried out over the set to which the label λ_k belongs.

$$p_i^{(t+1)}(\lambda_k) = \frac{1}{m}\sum_j \frac{p_i^{(t)}(\lambda_k) q_{ij}^{(t)}(\lambda_k)}{\sum_{\lambda \varepsilon [\lambda_k]} p_i^{(t)}(\lambda) q_{ij}^{(t)}(\lambda)} \qquad k=1, \cdots, n$$

Label sets are specified by the command option [s nsets size1 ...], where nsets specifies the number of different sets and size1, size2, ..., the number of labels in each set. The sets must consists of consecutive labels, *e.g.*, [s 3 2 4 3] specifies that there are 3 sets of labels; the first set contains the 2 labels 0, and 1; the second set labels 2, 3, 4, and 5; and the third set labels 6, 7, and 8.

Although the added flexibility of this label set grouping is available, its use is not recommended. The numbers computed by this method are no longer probabilities and the behavior of the process cannot be predicted[1].

The *prelax* program must have been compiled previously: see *setup*.

[1]Azriel Rosenfeld, personal communication, 1983.

quit n
Quit the current level of the CI driver. At the top level this will leave RELAX. The argument n may be provided to quit more than one level. Specify -1 or some large number to abort all levels of the driver and exit from the program.

setup {h|p} nlabels [ncols nrows]
Setup is used to create programs for relaxation operations on images. Both a compatibility operator and a relaxation operator will be compiled. Either Hummel-Zucker-Rosenfeld (h) or Peleg (p) relaxation formulas may be chosen; the corresponding programs will be *hcompat* and *hrelax* or *pcompat* and *prelax*. The default is "h". You may also specify the number of class labels (default 2) to be used and the size of the relaxation neighborhood (default 3 × 3).

5.3. Batch Execution

The RELAX program offers two methods of invoking prestored commands. The first is the interactive invocation of CI command files, as illustrated earlier. The second is to drive the entire RELAX session from an operating-system script. A UNIX shell script invoked by, *e.g.*, "runrelax picture.img 10 195", might look like the following:

```
# This is file "runrelax".
#
# Argument $1 is the gray-level image.
# Arguments $2 and $3 are the optional low and
#   high clipping values.

relax <<!

  : Make 3 x 3 neighborhood, two label,
  : HZR coefficient computation and
  : relaxation programs.
  setup h 2

  : Display the original image, $1.
  erase
  insert $1 100 348

  : Transform the picture into a probabilistic image.
  imgprb $1 prb.img 2 $2 $3

  : Compute the compatibility coefficients.
  hcompat prb.img compat.dat

  : Apply the relaxation operator.
  hrelax prb.img compat.dat

  : Display the smoothed image.
  prbimg prb.img output1.img
  insert output1.img 224 348
!

echo "Finished."
```

Note that the shell substitution mechanism can be used. This is an advantage over interactive use, in which no substitution of variables is currently implemented.

To save the typed terminal output, you should pipe the standard output into a file. The UNIX method for doing this is to add *>session.log* to the *relax* command within the script or to the UNIX command line that invokes the script. You may also use the UNIX *script* or *tee* commands to route the typed output simultaneously to a file and to your terminal. (UNIX output buffering generally prevents you from interacting with a program while the script is being created. The EUNICE operating system does not have this limitation.)

The actual submission of this shell script is described in the UNIX programmer's manual. You should run it in foreground mode if you want to interact with the program. If you run it in background mode, be sure to pipe the output into a log file so that it won't appear on your terminal. You can monitor the log file during execution, using the *cat* or *tail -f* [formerly *tra*] commands to make sure everything is running smoothly, although the UNIX buffering mechanism prevents you from monitoring in real time. You can also halt the process or reconnect it to your terminal if you wish.

Section 6

Evaluation

This section documents the performance of the Hummel-Zucker-Rosenfeld and Peleg relaxation algorithms on a variety of imagery and in several scene analysis tasks. Although we have chosen realistic tasks, our goal was not the full exploration of relaxation techniques for these applications. Rather, we have chosen tasks with simple mapping functions to and from the probability domain so that we could better assess the actions of the probability-updating functions.

We could devise objective performance measures for particular tasks [Fekete81], but the relationship of such measures to subjective scene analysis performance would be difficult to quantify. Linear-feature extraction, for instance, must be rated differently when we desire only prominent features (*e.g.*, for stereo image matching) or closed boundaries of regions than when we desire extraction of all detectable discontinuities.

We could also compute performance measures for restoration of images to which we have added blur or noise, but we would need models of the imagery and degradations occurring in realistic scenarios. If such models were available, other methods of image restoration would almost certainly be more effective than heuristic relaxation. Furthermore, any comparison of relaxation output with ground truth would necessarily depend on the function used for mapping the initial image to the probability domain.

We have therefore chosen a subjective evaluation procedure. We compare the relaxation output with the probability input after each has been mapped back to the luminance domain. This type of comparison is valid for almost any task. It measures whether the relaxation operation has made the image more useful for the intended application.

The particular scene analysis tasks we have selected are discussed below. For any task and type of imagery, we still must choose the

- Image-to-probability and inverse mappings
- Size (and shape) of the compatibility neighborhood
- Method of computing compatibility coefficients
- Relaxation scheme (HZR or Peleg)
- Number of relaxation iterations.

We shall explore these choices in the following subsections and then summarize our conclusions.

6.1. Task Selection

The RELAX package supplied by the University of Maryland contained a demonstration of noise suppression on an infrared image of a military tank. The nature of the image, a single bright tank centered against a dark background, made this also a demonstration of object detection and of segmentation in a noisy image. We have

used this image (and the implied tasks) extensively in learning to run the RELAX package and to understand its functioning. Two other test tasks we have selected are edge linking and anomaly detection.

Edge linking is the enhancement of linear features in an image by strengthening connected edge elements and suppressing isolated or conflicting ones. A set of edge operators is first passed over the image; they return, for each pixel, the probabilities that scene edges at various orientations are present. (We shall treat *no-edge* as a label also.) The relaxation stage strengthens the probabilities of edge labels that link "nose to tail" with the edges at neighboring pixels while suppressing other edge pair orientations.

Anomaly detection is the identification or enhancement of isolated blobs against a fairly uniform background. An image operator first classifies pixels as either background or anomalous according to the statistics of the background and the gray level of each pixel. Relaxation then reinforces the classification of a pixel if its neighboring pixels are similarly classified. This two-label operation may also be viewed as a noise-suppression application.

6.2. Edge Linking

The first step in edge linking is to calculate the initial probabilities that scene edges at various orientations pass through each pixel. We convolve the image with four Sobel-type masks that detect *northwest*, *north*, *northeast*, and *east* edges. Because we consider opposing gradient directions (or contrasts) to be equivalent, there are four orientations rather than eight.

The 3 × 3 convolution operators are:

0	2	1	1	0	−1	1	2	0	1	2	1
−2	0	2	2	0	−2	2	0	−2	0	0	0
−1	−2	0	1	0	−1	0	−2	−1	−1	−2	−1
northwest			*north*			*northeast*			*east*		

Five labels are then attached to each pixel: *no-edge*, *northwest*, *north*, *northeast*, and *east*. The probabilities for each of these labels are calculated from the outputs of the four convolution operators, $Op(dir)$. For label *north*, the probability at a particular pixel is calculated as

$$Prob(north) = \frac{Op(north)}{\sum_{all\ dir} Op(dir)} \cdot \frac{\max_{all\ dir} Op(dir)}{\max_{\substack{all\ dir,\\ all\ image\\ pixels}} Op(dir)},$$

and similarly for $Prob(northwest)$, $Prob(northeast)$, and $Prob(east)$. The first term relates the $Op(north)$ response to the response of the other edge operators; the second compares the operator response at that pixel with responses over the entire image.

The probability of *no-edge* is calculated as

Evaluation

$$Prob(no\text{-}edge) = 1 - \frac{\max_{all\ dir} Op(dir)}{\max_{\substack{all\ dir,\\ all\ image\\ pixels}} Op(dir)}.$$

Together these probabilities constitute the initial probability image. Similar schemes have been reported in the literature [Riseman77, Zucker77].

The next step is to calculate the compatibility coefficients. We have generally used the RELAX package *hcompat* and *pcompat* routines for that purpose. Our aim was to evaluate fully automated relaxation rather than to develop optimal compatibility matrices for this one application; the former is the usual approach in the relaxation literature.

Figures 1-13 illustrate the results of our edge-linking efforts. Each figure consists of sixteen pictures. The upper-left picture is the original image. The other three pictures across the top of the figure and the leftmost picture on the second row are the *northwest*, *north*, *northeast*, and *east* edge maps. In these images, black represents zero probability of an edge, white a unity probability.

The remaining pictures are binary images. The second from the left in the second row is the inverse mapping of the original probability image. It is produced from the four edge maps by displaying, for each pixel, the label with the highest probability; *no-edge* maps to black while all other labels map to white. The remaining ten pictures are obtained by successive iterations of the relaxation procedure. They represent the results of 1, 2, ..., 10 iterations of relaxation. Figure 2 is the exception: the ten pictures are the results of 10, 20, ..., 100 iterations.

Figures 1 and 2 illustrate a fairly typical relaxation application. The first few iterations of relaxation show a strong edge-linking effect. Later iterations seem to do little except smooth or blur the enhanced structures. This dual nature of relaxation has been analyzed by Richards [Richards80]. Figure 3 shows that the Peleg relaxation scheme exhibits essentially the same behavior.

Figure 4 illustrates the HZR method on aerial imagery. The improvement in the displayed output is rather dramatic, although it may be partially due to the method of output mapping.

Figures 5 and 6 illustrate the effect of larger neighborhoods. The 7×7 neighborhood increases edge spreading for the HZR method and decreases it for the Peleg method. Computation time is much greater for large neighborhoods, of course.

Figure 7 shows that strong edge structure does not guarantee effective compatibility coefficients if the edges appear equally in many orientations and relationships. This is opposite to the effect of most enhancement operators.

Figure 8 shows the result of "enhancing" the edges in an image that has no perceptual edge structure. The procedure for computing compatibility coefficients registers any regularities in the image, not just strong responses from the edge detectors. This, combined with the relaxation blurring effect, gives a surprisingly good noise-suppression or object detection operator.

Figures 9 through 12 show that the exact values of the compatibility coefficients are not critical. Figure 9 was made with coefficients rounded to one decimal place, Figure 10 with these same numbers doubled. (Doubling the coefficients will have no effect on Peleg relaxation.) Figure 11 was generated using compatibility coefficients

derived from the image in Figure 12; Figure 12 was likewise generated using compatibility coefficients derived from the image in Figure 11. In each case the tampering had relatively little effect.

Figure 13 shows what happened when we supplied the coefficients by hand to see if a careful choice of the values could produce faster relaxation effects. Although these coefficients were intuitively derived, they perform very badly. For a pixel and the neighbor above it, the coefficients we used for the Hummel-Zucker-Rosenfeld scheme are

$$\begin{aligned}
Coeff(north, northwest) &= 0.50 \\
Coeff(north, north) &= 1.00 \\
Coeff(north, northeast) &= 0.50 \\
Coeff(north, east) &= -0.50 \\
Coeff(north, no\text{-}edge) &= -0.25 \\
Coeff(northeast, northwest) &= -0.50 \\
Coeff(northeast, northeast) &= -0.75 \\
Coeff(northeast, east) &= -0.50 \\
Coeff(northeast, no\text{-}edge) &= 0.50 \\
Coeff(east, east) &= -0.75 \\
Coeff(east, no\text{-}edge) &= 0.75
\end{aligned}$$

For a pixel and its northeast neighbor we used

$$\begin{aligned}
Coeff(north, northwest) &= -0.50 \\
Coeff(north, north) &= 0.25 \\
Coeff(north, northeast) &= 0.25 \\
Coeff(north, east) &= -0.25 \\
Coeff(north, no\text{-}edge) &= 0.50 \\
Coeff(northeast, northwest) &= -0.5
\end{aligned}$$

(The other required compatibility coefficients can be derived from these by using symmetry considerations.) We also tried variations of the above, *e.g.*, reversing the signs of some coefficients; there were no significant changes in results. The lesson seems to be that manual selection of coefficients is exceedingly difficult. A better policy is to generate coefficients using *hcompat* or *pcompat* and to modify the values only slightly.

6.3. Anomaly Detection

Anomaly detection is a two-label problem: each pixel is part either of the background or of an anomaly. We assume that the background comprises the majority of the image and that it can be modeled as a constant gray level plus Gaussian noise. Anomalies are regions that diverge significantly from this model. The relaxation process should strengthen the probability of an *anomaly* label for dark or light groups of pixels while suppressing the *anomaly* label for isolated pixels that are equally bright.

The background is modeled by the mean gray level and standard deviation for the whole image. Then, for each pixel in turn, we calculate the deviation of the pixel value from the background value (*i.e.*, from the image mean). The probability that

the pixel is a background pixel is then taken to be the probability that a deviation this large or larger can occur by chance. The probability that the pixel is part of an anomaly is taken to be 1.0 minus this value.

It is somewhat easier to present the mathematical formulas in the reverse order. Let us suppose that a pixel is t standard deviations from the image mean. We calculate the probability of the label *anomaly* as

$$Prob\,(anomaly) = \int_{-t}^{t} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}\,dx,$$

$$Prob\,(background) = 1 - Prob\,(anomaly).$$

These initial probabilities are used to calculate the compatibility coefficients by means of the *hcompat* or *pcompat* routine. They also serve as input to the relaxation process. After a number of relaxation iterations, the pixels for which the *anomaly* label has the higher probability are displayed as white points in a binary image.

Figures 14-16 are the results of our efforts at anomaly detection. Each figure is a pair of images that document the results for one example. The four pictures in the top image, from left to right, are the

- Original scene.
- Anomaly probabilities mapped to gray levels 0 to 255.
- Binary image formed by inverse mapping the two-label probability image with the *prbimg* routine.
- Binary result of one iteration of relaxation.

The bottom image shows, from left to right, the results of 2 through 5 iterations of relaxation.

Figure 14 begins with the original gray-level image of a composite road scene. Figure 15 is similar, but derived from a version of the road scene that has been "cleaned" to remove the background road profile and make the vehicles (anomalies) more distinct. (This technique is part of the SRI road tracker [Quam78].) Figure 16 is the Peleg method applied to the cleaned road scene. In each case, relaxation enhances the background noise structure until it swamps the anomaly signal.

The lesson from this sequence is that a good initial image does not guarantee a good result. The compatibility coefficients derived by *hcompat* or *pcompat* do not necessarily encode our own notions of image "structure." Relaxation-based anomaly detection may be feasible, but not by using the straight-forward approach attempted here.

6.4. Summary

For any task and type of imagery, the user must choose the

- Image-to-probability and inverse mappings
- Size (and shape) of the compatibility neighborhood
- Method of computing compatibility coefficients

* Relaxation scheme, HZR or Peleg
* Number of relaxation iterations.

We can now give some guidance on these matters as well as on the question of whether to use relaxation at all.

We have presented some simple image-to-probability and inverse mappings. Each mapping is designed for a specific application, and the success of relaxation processing depends critically on the quality of the initial mapping. While we can say little about the initial imagery or the mapping function, we can suggest some constraints on their combination—the probability image.

The probability image is the source of both the compatibility coefficients and the relaxation output. Any repetitive structure in this image will be reflected in the coefficients and enhanced in the output image. It is therefore essential that the probability image have the following characteristics:

* The signal, or desired characteristics, should dominate the noise.
* The signal must be spatially correlated within the chosen neighborhood; the noise, however, should be uncorrelated.
* The signal must contain some spatial relationships and not others; the noise should contain all relationships equally.

Relaxation will be successful to the extent that these conditions are met.

The user should specify a neighborhood just large enough to guarantee the above conditions. An application with large contiguous regions, such as land use classification, might benefit from the noise immunity of a large neighborhood. Generally, though, a 3×3 neighborhood is the best choice. If the signal does not dominate the noise locally, it is unlikely to dominate it within larger neighborhoods. If it does dominate on the average, relaxation provides a mechanism for propagating the signal a short distance into those regions where the signal is weak. Larger neighborhoods would increase penetration distance, but at the cost of greatly increasing the computation time.

Different compatibility coefficients are needed by different relaxation methods, but the user is faced with similar choices in computing them. They may be computed separately for each image, jointly for an ensemble of images, or theoretically for some image model. The same set of constraints applies: the sampled or modeled population must have biased second-order statistics rather than an equiprobable selection of spatial relationships. This typically prohibits training on an ensemble of randomly oriented images.

The RELAX *hcompat* and *pcompat* routines look for combinations of neighboring labels that occur more (or less) frequently than that expected if the label assignments were statistically independent. These estimates are calculated under the assumption that the image is a homogeneous data set, ignoring the fact that the structure in one part of the image may be significantly different from that of other parts. It has been suggested that this disadvantage may be alleviated by carrying out the calculation over windows of the image [Nagin82]. However, the problem of estimating coefficients from small samples worsens as the sample size decreases. If we have many labels, a large neighborhood, and a small sampling area, the number of samples with similar pixel configurations will be small and the estimated compatibility coefficients will be unreliable.

An alternative method for obtaining compatibility coefficients is for the user to

estimate them. For many labels and a large neighborhood, manual entry of the coefficients is a daunting task. Moreover, except for obvious cases of compatibility, it is difficult to arrive at good coefficient values by guessing. Figure 13 shows the hopeless results we obtained when we guessed what we believed to be reasonable values for the coefficients. A major difficulty was our reluctance to allow independent label combinations. Generally we assigned values that forced combinations to be either compatible or incompatible rather than independent. If manually entered values of the coefficients are to be used, we suggest considering those the package can provide as a guide to assigning reasonable values; slight modifications may then be beneficial.

Which relaxation scheme should be used? It does not seem to matter. One method sometimes does a little better than the other, but both work or fail together. The principal distinction we observed was that the relaxation smoothing effect could be reduced for the Peleg method by using a larger neighborhood.

The last question that needs to be answered is how many iterations of relaxation should be performed. There appear to be two aspects to the relaxation process. The first three or four iterations often show moderate enhancement, while later ones are often dominated by blurring. Little change occurs after ten iterations. Only by looking at the output can one ascertain the optimum halting point, but approximately four iterations seems to be a good rule of thumb.

Relaxation is essentially an enhancement operator, with the structure to be enhanced derived from the image rather than from any model of either the imagery or the application. What is the cost of this signal enhancement? One iteration of relaxation, using a 3×3 neighborhood over a 128×128 image, took approximately three CPU minutes on a VAX 11/780—a considerable cost if image smoothing is the desired result. These costs increase linearly with the number of image pixels, the number of pixels in the neighborhood, and the number of iterations.

Evaluation

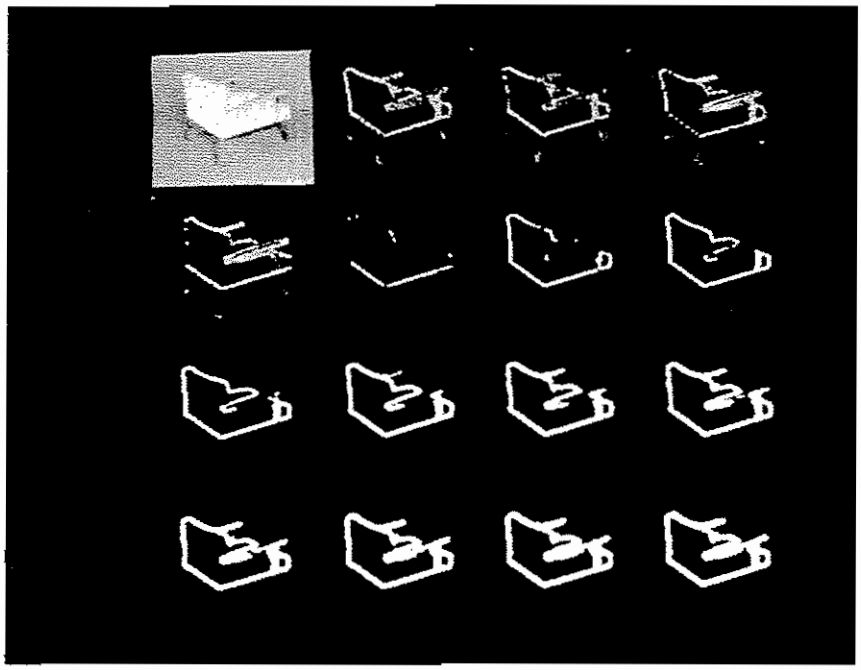

Figure 1. HZR relaxation using a 3×3 neighborhood and *hcompat* compatibility coefficients. See text for explanation of the figure format. Edge linking during the first iterations is supplanted by blurring or spreading.

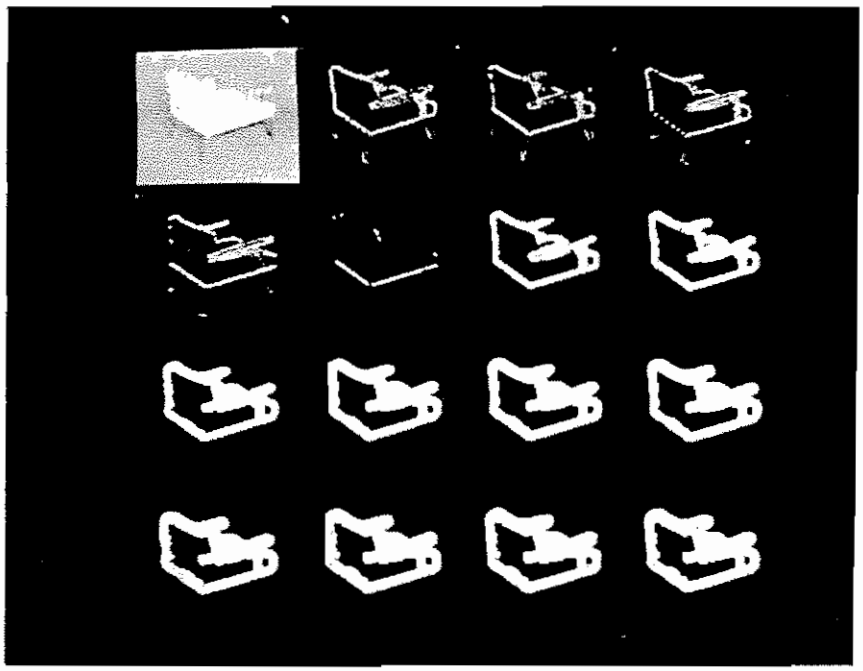

Figure 2. Further iterations of the example shown in Figure 1. Iterations 10, 20, ..., 100 are shown. These additional iterations only spread the edges.

Evaluation

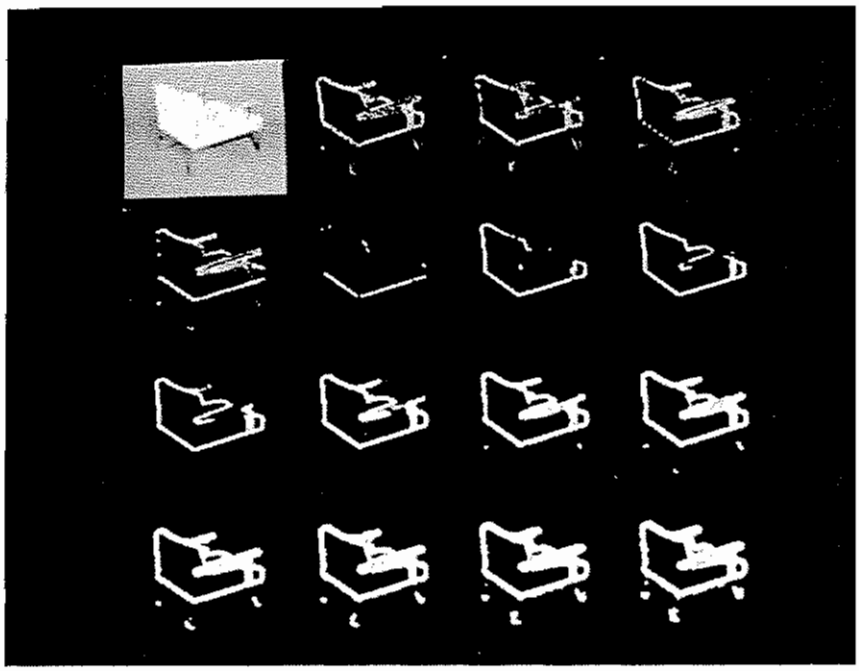

Figure 3. Peleg relaxation using a 3 × 3 neighborhood and *pcompat* compatibility coefficients. Results are similar to those obtained using the HZR scheme. See Figure 1 for comparison.

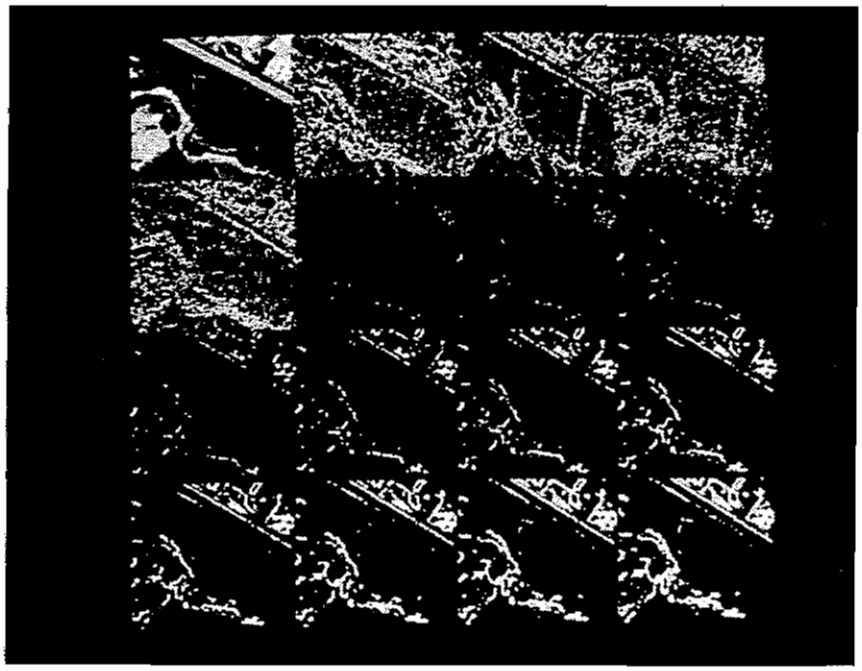

Figure 4. An aerial image with HZR relaxation using a 3 × 3 neighborhood and *hcompat* compatibility coefficients. Relaxation extracts structure from noisy edge data.

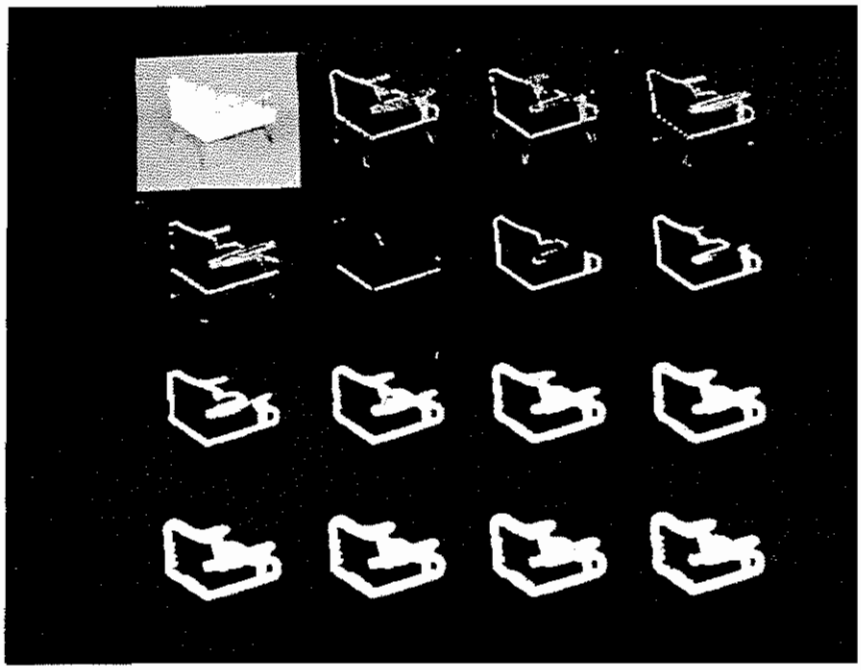

Figure 5. HZR relaxation using a 7 × 7 neighborhood and *hcompat* compatibility coefficients. The use of a larger neighborhood in the HZR relaxation scheme generally increases edge spreading.

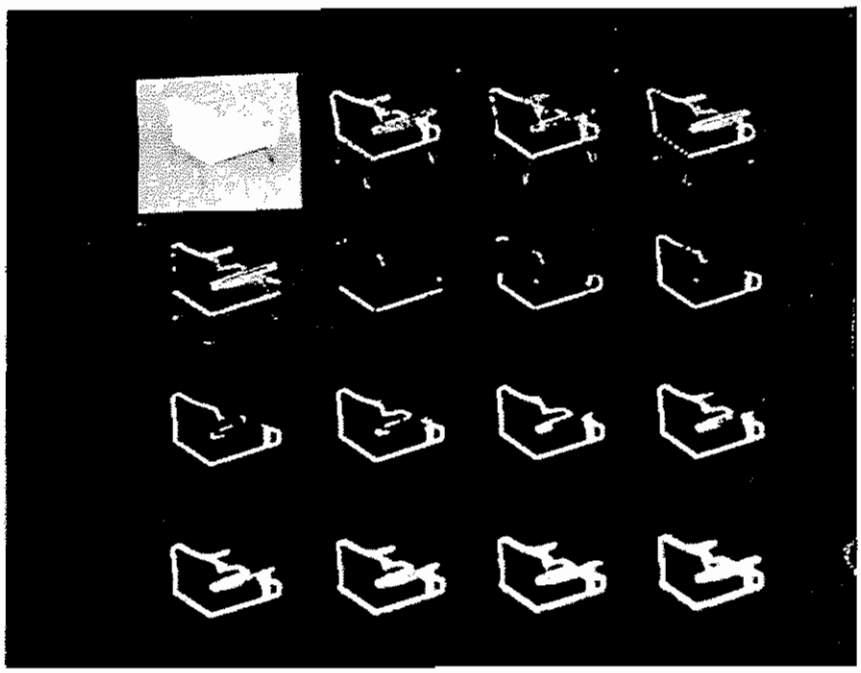

Figure 6. Peleg relaxation using a 7 × 7 neighborhood and *pcompat* compatibility coefficients. The larger neighborhood reduces the edge spread instead of increasing it.

Evaluation

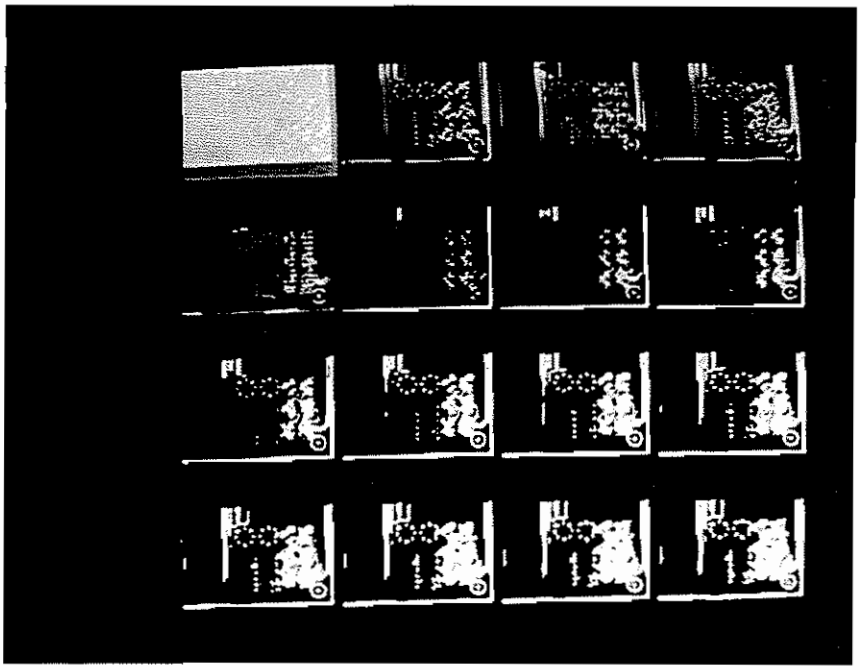

Figure 7. A printed-circuit board image with HZR relaxation using a 3 × 3 neighborhood and *hcompat* compatibility coefficients. Both this example and Figure 1 have sharp edges, but this example lacks bias in the edge orientations.

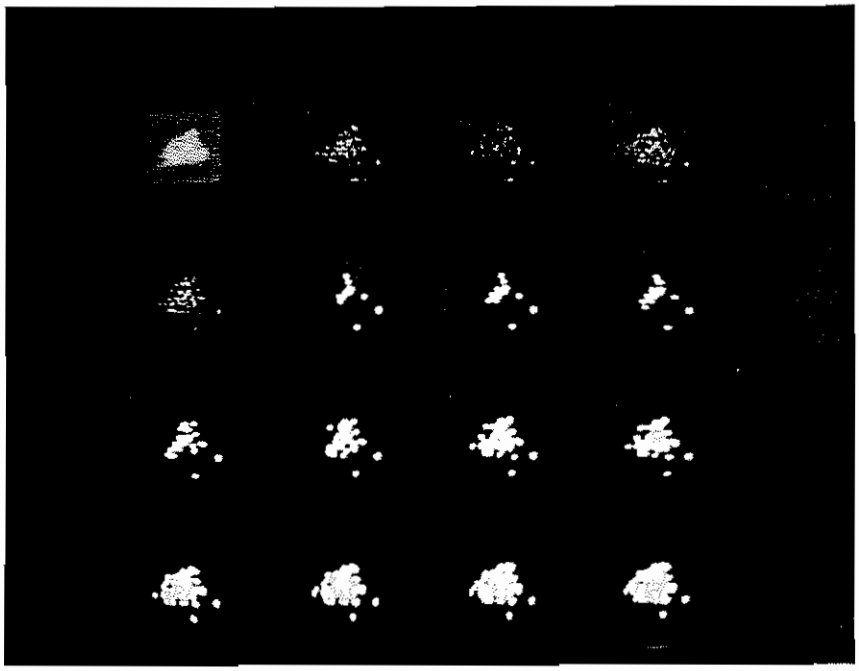

Figure 8. A noisy tank image with HZR relaxation using a 3 × 3 neighborhood and *hcompat* compatibility coefficients. Although the edge-detection method is inappropriate here, it works quite well for smoothing or object detection.

Evaluation

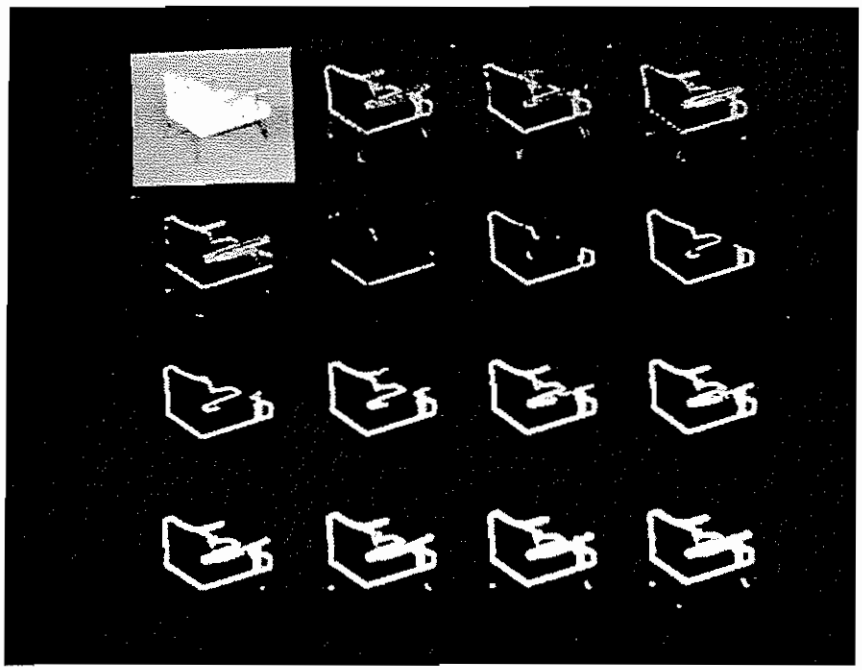

Figure 9. HZR relaxation using a 3 × 3 neighborhood and *hcompat* compatibility coefficients rounded to one decimal place. Comparison with Figure 1 shows that exact values of the coefficients are not important.

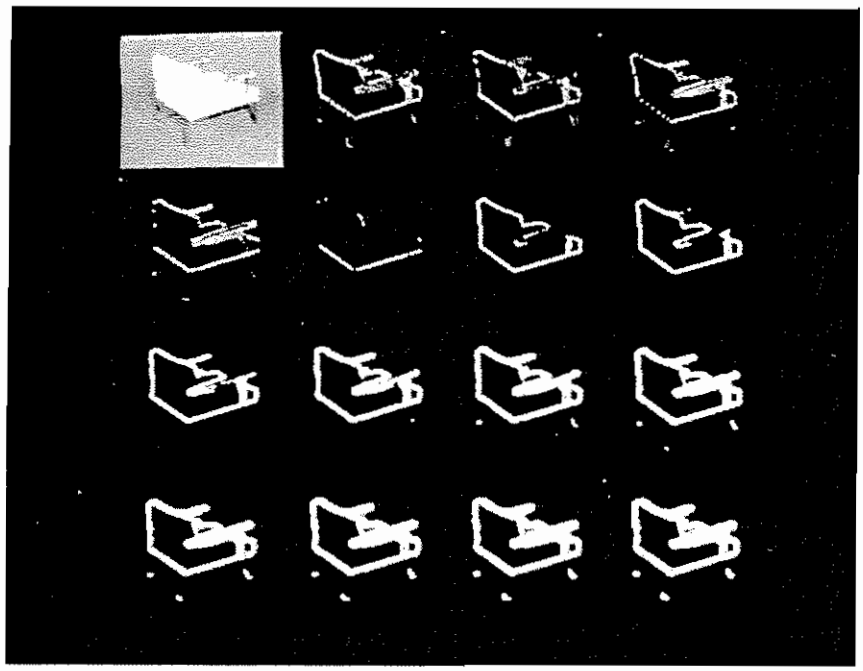

Figure 10. HZR relaxation using a 3 × 3 neighborhood and *hcompat* compatibility coefficients that are twice those for Figure 9. Only the relative values of the compatibility coefficients are important.

Evaluation

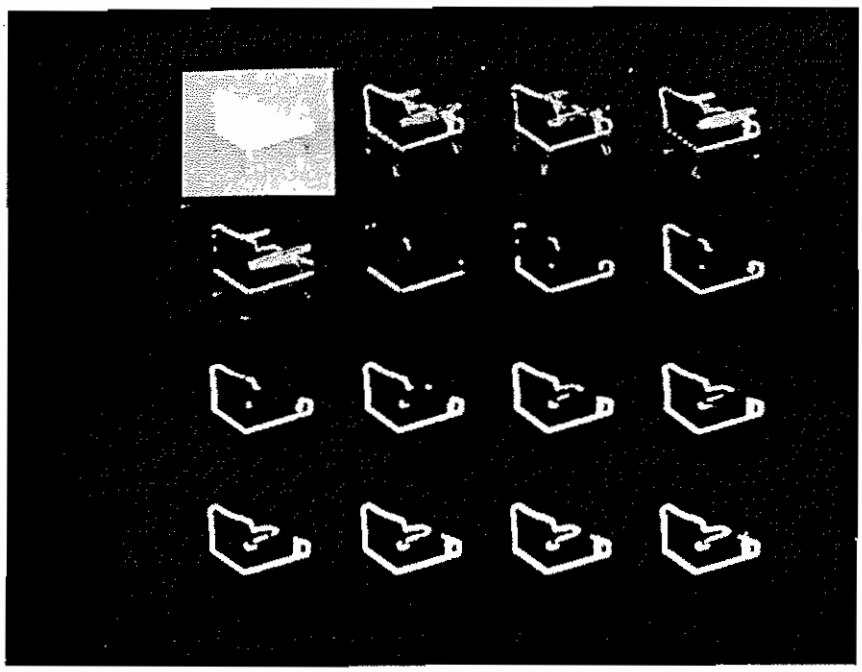

Figure 11. HZR relaxation using a 3 × 3 HZR and compatibility coefficients derived from the aerial scene in Figure 12. Useful compatibilities may be derived from dissimilar imagery.

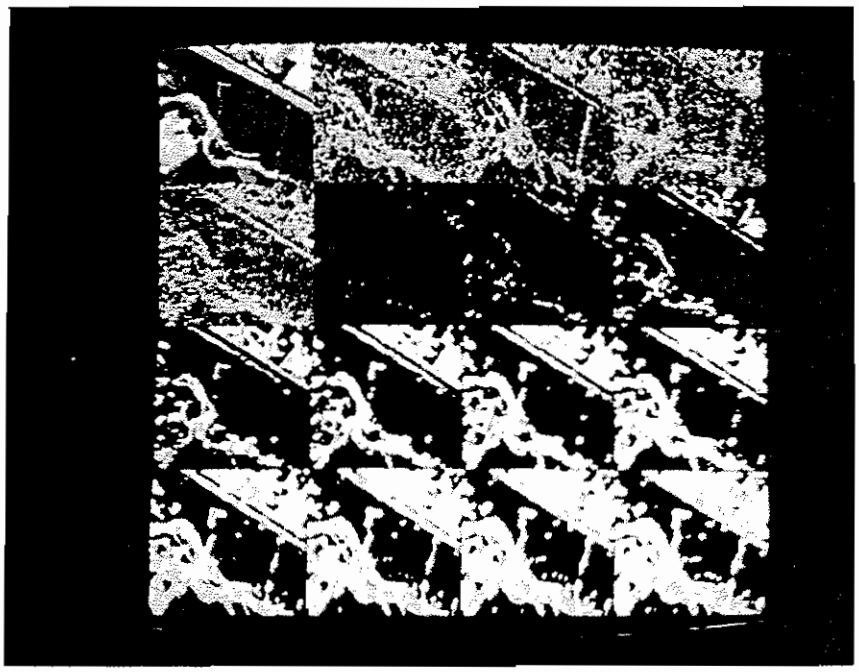

Figure 12. HZR relaxation using a 3 × 3 neighborhood and compatibility coefficients derived from the chair scene in Figure 11. Useful compatibilities may be derived from dissimilar imagery.

Evaluation

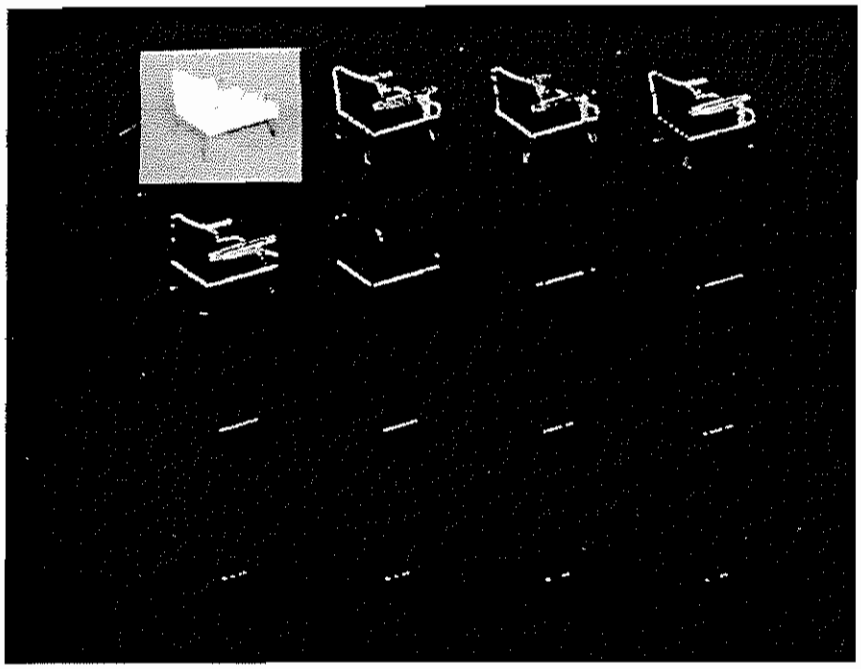

Figure 13. HZR relaxation using a 3 × 3 neighborhood and manually selected compatibility coefficients; see text for details. Although these coefficients were intuitively determined, they perform very badly.

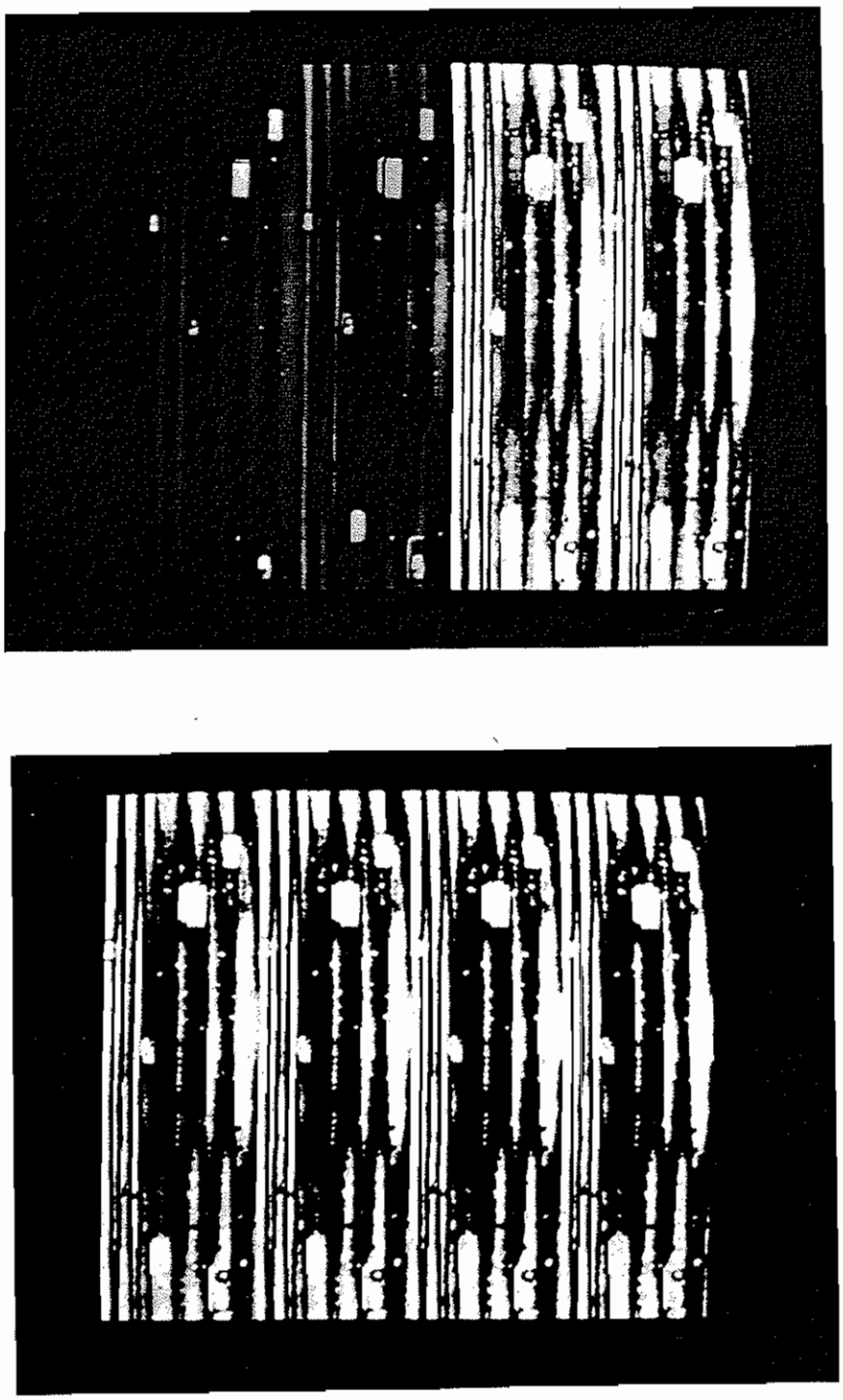

Figure 14. Anomaly detection by HZR relaxation using a 3 × 3 neighborhood and *hcompat* compatibility coefficients. Top image: the original scene, the anomaly probabilities, and binary outputs of iterations 0 and 1. Bottom image: iterations 2 through 5. Spatially correlated noise is enhanced along with the signal.

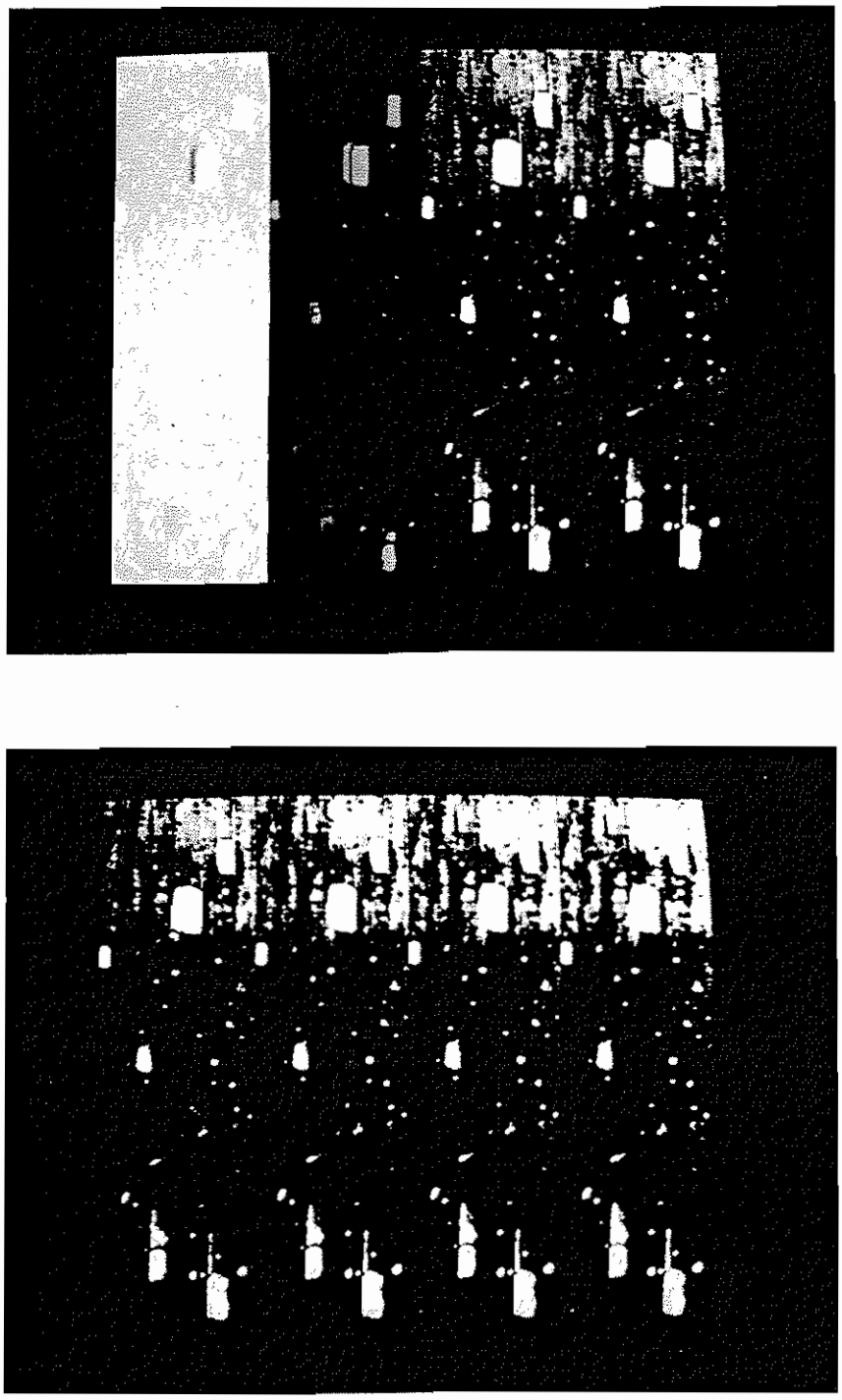

Figure 15. Anomaly detection by HZR relaxation using a 3 × 3 neighborhood and *hcompat* compatibility coefficients. These results are obtained from a "cleaned" version of the original scene used in Figure 14.

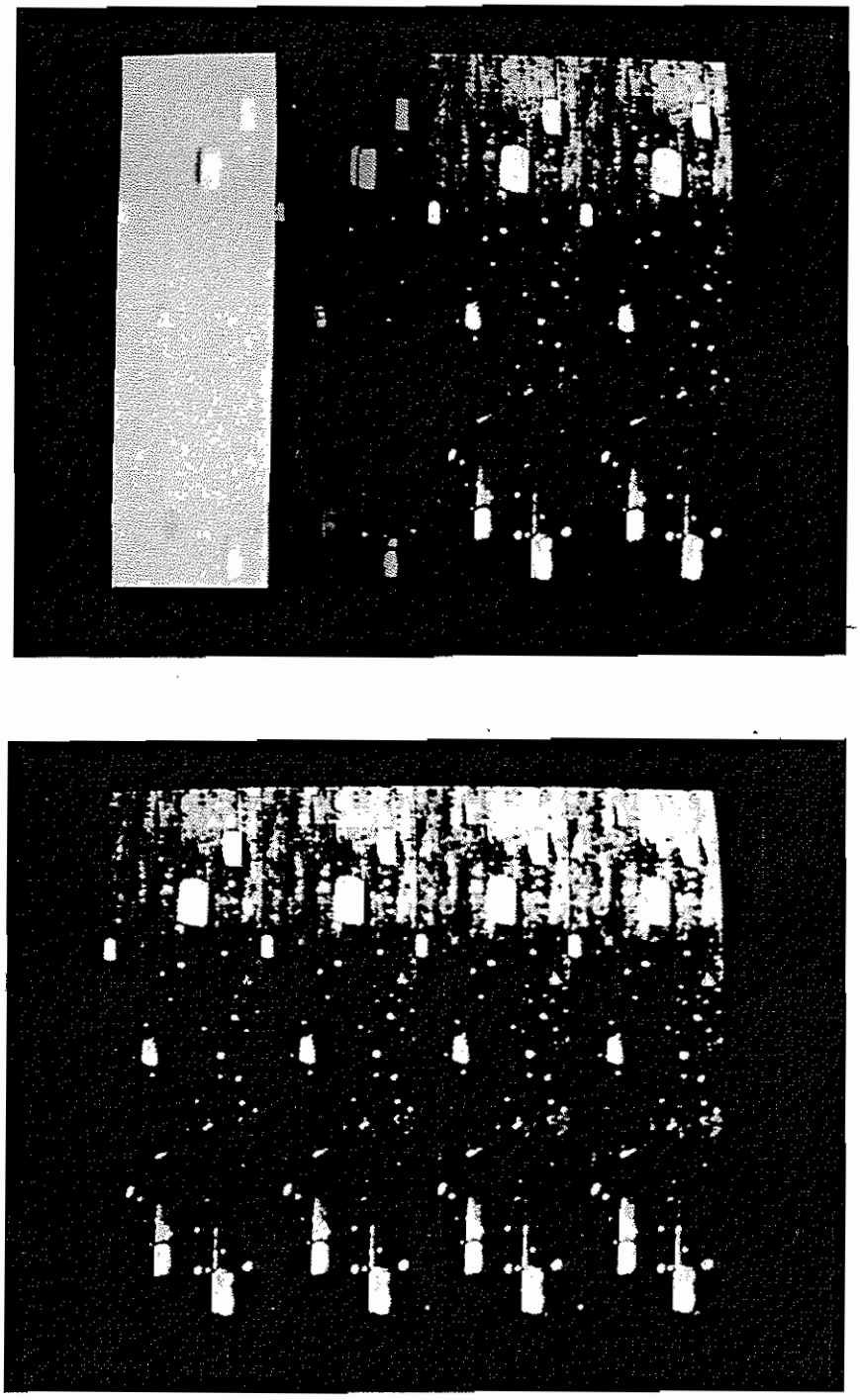

Figure 16. Anomaly detection by Peleg relaxation using a 3 × 3 neighborhood and *pcompat* compatibility coefficients. Comparison with Figure 15 shows that HZR and Peleg methods produce similar results.

Section 7

Suggested Improvements

The process of evaluation has turned up several ways to improve or extend the current RELAX implementation. Comments about existing features have been made at the appropriate junctures throughout this document. The following are additional suggestions for substantial modifications or needed research.

* *Improved Coefficient Entry and Editing*

  Manual entry of compatibility coefficients is currently very awkward, and once entered the coefficients cannot be displayed or altered. The Testbed *view* program was developed to display files of coefficients, but a more flexible display-and-editing capability is needed within the relaxation package itself.

  One way to reduce the burden on the user is to use symmetry or other constraints to reduce the number of coefficients that must be typed in. Another is to allow entry of important individual coefficients, with all others defaulting to the central value (0.0 or 1.0). (This approach could be extended to the relaxation updating formula, with only important terms actually being entered into the computation.) In any case, a coefficient query and correction capability would be very useful.

* *Ensemble Coefficient Extraction*

  The current *hcompat* and *pcompat* routines extract compatibility coefficients from one probability image. There may be applications for which average coefficients from an ensemble of similar images are desired. A simple program could be written to extract coefficients from such an ensemble, or to combine coefficient matrices derived from individual images.

* *Supervised Learning*

  Relaxation enhancement is often unpredictable when the compatibility coefficients are derived from noisy images. One could argue that "cleaned" or "ground truth" images should be used for deriving the coefficients. This would build signal statistics into the compatibility coefficients directly instead of depending on desired signals to dominate the noise in the initial probability image. The result should be faster convergence and better signal enhancement [Peleg78a, Eklundh80].

Suggested Improvements

- *Adaptive Coefficients*

  Since relaxation can be used to enhance signals and suppress noise, it may be useful for producing cleaned images for the estimation of compatibility coefficients. In the limit, new coefficients could be extracted during (or after) each application of relaxation updating. This would either produce faster enhancement or faster degradation of the image.

- *Use of Decision Logic*

  The Hummel-Zucker-Rosenfeld or Peleg updating formulas in the RELAX package may be well suited for enhancement and smoothing applications. Many other uses of iterative image modification would require nonlinear decision logic in the updating algorithms. The decision logic might make use of the image histogram, the statistics of objects already found in the image, or other global information.

- *Use of Joint Neighborhood Constraints*

  The current probability updating formulas use only pairwise relationships to compute a new pixel label probability. Other types of iterative image operators often look for patterns in the neighborhood as a whole. There may be relaxation applications for which joint neighborhood relationships must similarly be modeled. See Peleg [Peleg80a] for a discussion of the conditional independence assumption.

- *Adaptive Neighborhood Definition*

  A particular use of joint neighborhood constraints and decision logic is to decide, for each neighborhood, which pixels belong to the same region as the central pixel. Only those pixels would be used in updating the central-pixel label probabilities. This should speed convergence in segmentation applications.

- *Halting Criteria*

  The RELAX package currently offers no way to ascertain how many iterations of relaxation updating are sufficient for any given task. We have suggested that three or four iterations are usually optimum for enhancement applications, but there are no image-dependent rules for determining when improvement has stopped and blurring has taken over. More research is necessary in this area. See Fekete *et al.* [Fekete81] for an approach based on examining the rates of change and the entropies of the probability vectors at each iteration.

Suggested Improvements

* *Further Research*

We have attempted to evaluate relaxation as a technique rather than make an exhaustive study of its application to a particular task. If relaxation seems promising for a specific task, however, such a thorough evaluation may be required. As relaxation techniques are still in their infancy, further research is needed to determine where and how they may be best applied.

Section 8

Conclusions

Relaxation is a procedure for enhancing the signal, or features, found in an image by an imperfect enhancement, detection, or classification operator. It is a very general technique and has been used in a variety of image-processing applications.

The approach works when a label at one image pixel is constrained by neighboring labels. The relaxation procedure discovers and exploits these relationships to produce a more consistent labeling. Where an initial label is strongly believed, it tends to be unchanged by relaxation updating. Where it is uncertain, relaxation tends to propagate either the neighborhood information or its own biases into the classification. This results in either enhancement or smoothing, depending on the nature of the compatibility coefficients.

There are three basic components to relaxation: mapping the original image to a probability domain, estimating the compatibility coefficients, and applying the updating formula. The updating formulas in the RELAX package are simple, standard, and nearly equivalent, so that only the first two components are of concern.

Each application domain requires a different mapping to the probability image format. The mapping should be such that (1) the desired signal dominates other image components; (2) the signal is locally correlated and occurs in only certain neighboring combinations; (3) the noise is locally uncorrelated and appears in all possible combinations.

Compatibility coefficients can be provided by hand, although they are exceedingly difficult to derive for most applications. The automated coefficient extraction routines in the RELAX package work well if the mapping constraints above are met, but produce surprising results otherwise. If the noise or unwanted signal in an image is spatially correlated, it will be enhanced. If the desired signal takes on all possible local relationships, it will not be enhanced. Enhancement using image-based compatibility coefficients can improve on a good initial image, but will not redeem an incompetent detection operator.

Relaxation methods for solving "gravitational" or "fluid flow" problems such as histogram sharpening, requantization, image smoothing, and classification-map improvement have been reported in the literature. Relaxation can be used for model-independent enhancement, but is often more costly and less effective than model-based enhancement or restoration when appropriate models are available.

The RELAX package provides a mechanism for exploring the relaxation philosophy in image-based applications. Relaxation techniques are still in an early stage of development, and more research is needed into both theoretical foundations and domains of applicability.

Appendix A

The GPSPAR Relaxation Package

This appendix documents the control language used by the original University of Maryland contribution to the Testbed, the GPSPAR relaxation package. This is a set of stand-alone programs that may be invoked in sequence, either interactively or using a UNIX shell script.

The capabilities and user interfaces of the GPSPAR programs are essentially identical to those documented for the interactive RELAX driver. (We have changed the names of some of the programs during integration into the Testbed; for instance *prbimg* was originally known as *display*. The shell script is just another method of invoking these programs. A sample script is shown below.

```
# This program will do everything a person sitting
# at a terminal would do to:
#
#    Set up compatibility coefficient creation and
#    relaxation programs using the Hummel-Zucker-Rosenfeld
#    formulas,
#
#    Create a two label probabilistic image from the
#    gray level "tank" picture (or other image) using
#    the problem-specific program "imgprb",
#
#    Compute the compatibility coefficients from this
#    image using the program "hcompat" that was produced
#    by "setup",
#
#    Perform eight iterations of relaxation using "hrelax"
#    as well as converting each resulting probabilistic
#    image into a gray level image, output.img, using the
#    "prbimg" program.


# Erase the screen.
erase


# Make 3 x 3 neighborhood, two label, HZR coefficient
# computation and relaxation programs.

csh /iu/tb/bin/setup.csh h 2


# Transform the picture into a probabilistic image.
# $1 is the image name, $2 and $3 are the optional
# low and high pixel range specifications.

if ($#argv < 1) then
  imgprb /iu/tb/pic/tank/bw.img prb.img 2 13 49
else
  imgprb $1 prb.img 2 $2 $3
endif
```

```
# Recreate a gray level image from this.  This first
# "output" image will have gone through essentially
# the same steps as the other new gray level images
# to be created.  The output is stretched to fill
# 8-bit pixels.

prbimg prb.img output0.img
show output0.img -t 100 348


# Compute the compatibility coefficients from the
# probabilistic image.
hcompat prb.img compat.dat


# Perform eight iterations of relaxation on the image.
# After each iteration display a gray level image
# representation.

hrelax prb.img compat.dat
prbimg prb.img output1.img
show output1.img -i -t 224 348

hrelax prb.img compat.dat
prbimg prb.img output2.img
show output2.img -i -t 348 348

hrelax prb.img compat.dat
prbimg prb.img output3.img
show output3.img -i -t 100 224

hrelax prb.img compat.dat
prbimg prb.img output4.img
show output4.img -i -t 224 224

hrelax prb.img compat.dat
prbimg prb.img output5.img
show output5.img -i -t 348 224

hrelax prb.img compat.dat
prbimg prb.img output6.img
show output6.img -i -t 100 100

hrelax prb.img compat.dat
prbimg prb.img output7.img
show output7.img -i -t 224 100

hrelax prb.img compat.dat
prbimg prb.img output8.img
show output8.img -i -t 348 100

#   done
echo "Finished."
```

References

[Barrow76] H.G. Barrow and J.M. Tenenbaum, *MSYS: A System for Reasoning about Scenes*, Technical Note 121, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, April 1976.

[Bhanu82] B. Bhanu and O.D. Faugeras, "Segmentation of Images Having Unimodal Distributions," *Pattern Analysis and Machine Intelligence*, Vol. PAMI-4, No. 4, pp. 408-419, July 1982.

[Brice70] C.R. Brice and C.L. Fennema, "Scene Analysis Using Regions," *Artificial Intelligence*, Vol. 1, No. 3, pp. 205-226, Fall 1970.

[Davis77a] L.S. Davis and A. Rosenfeld, *Noise Cleaning by Iterated Local Averaging*, Technical Report TR-520, Computer Science Center, University of Maryland, College Park, April 1977.

[Davis77b] L.S. Davis and A. Rosenfeld, "Curve Segmentation by Relaxation Labeling," *IEEE Trans. on Computers*, Vol. C-26, No. 11, pp. 1053-1057, November 1977.

[Davis77c] L.S. Davis, "Shape Matching Using Relaxation Techniques," *Proc. IEEE Conf. on Pattern Recognition and Image Processing*, Troy, New York, pp. 191-197, June 6-8, 1977.

[Diamond82] M.D. Diamond, N. Narasimhamurthi, and S. Ganapathy, "A Systematic Approach to Continuous Graph Labeling with Application to Computer Vision," *Proc. National Conf. on Artificial Intelligence*, Pittsburgh, Pennsylvania, pp. 50-54, August 18-20, 1982.

[Eberlein76] R.B. Eberlein, "An Iterative Gradient Edge Detection Technique," *Computer Graphics and Image Processing*, Vol. 5, No. 2, pp. 245-253, 1976.

[Eklundh80] J.O. Eklundh, H. Yamamoto, and A. Rosenfeld, "A Relaxation Method for Multispectral Pixel Classification," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. PAMI-2, No. 1, pp. 72-75, January 1980.

[Faugeras80a] O.D. Faugeras and M. Berthod, "Scene Labeling: An Optimization Approach," *IEEE Conf. on Pattern Recognition and Image Processing*, Chicago, Illinois, pp. 318-326, August 6-8, 1979. Also in *Pattern Recognition*, Vol. 12, No. 5, pp. 339-347, 1980.

[Faugeras80b] O.D. Faugeras, "An Optimization Approach for Using Contextual Information in Computer Vision," *Proc. 1st Annual National Conf. on Artificial Intelligence*, Stanford, California, pp. 56-60, August 18-21, 1980.

[Faugeras81] O.D. Faugeras and K. Price, "Semantic Description of Aerial Images Using Stochastic Labeling," *IEEE Trans. on Pattern Anal. and Machine Intelligence*, Vol. PAMI-3, No. 6, pp. 633-642, November 1981.

[Fekete81] G. Fekete, J.O. Eklundh, and A. Rosenfeld, "Relaxation: Evaluation and Applications," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. PAMI-3, No. 4, pp. 459-469, July 1981.

[Freuder76] E.C. Freuder, *A Computer System for Visual Recognition Using Active Knowledge*, Technical Report AI-TR-345, Artificial Intelligence Laboratory, Massachussetts Institute of Technology, Cambridge, Massachussetts, June 1976.

[Guzman68] A. Guzman-Arenas, *Computer Recognition of Three-dimensional Objects in a Visual Scene*, Technical Report AI-TR-228 (MAC-TR-59), Artificial Intelligence Laboratory, Massachussetts Institute of Technology, Cambridge, Massachussetts, (AD-692-200), December 1968.

[Haralick79] R.M. Haralick and L.G. Shapiro, "The Consistent Labeling Problem: Part 1," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. PAMI-1, No. 2, pp. 173-184, April 1979.

[Haralick80] R.M. Haralick, J.C. Mohammed, and S.W. Zucker, "Compatibilities and the Fixed Points of Arithmetic Relaxation Processes," *Computer Graphics and Image Processing*, Vol. 13, No. 3, pp. 242-256, July 1980.

[Haralick83] R.M. Haralick, "An Interpretation for Probabilistic Relaxation," *Computer Vision, Graphics, and Image Processing*, Vol. 22, No. 3, pp. 388-395, June 1983.

[Hart77] P.E. Hart and R.O. Duda, *PROSPECTOR–A Computer-Based Consultation System for Mineral Exploration*, Technical Note 155, Artificial Intelligence Center, SRI International, Menlo Park, California, October 1977.

[Hummel78] R.A. Hummel and A. Rosenfeld, "Relaxation Processes for Scene Labeling," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-8, No. 10, pp. 765-768, October 1978.

[Hummel80] R.A. Hummel and S.W. Zucker, "On the Foundations of Relaxation Labeling Processes," Technical Report TR-80-7, Computer Vision and Graphics Laboratory, McGill University, Montreal, July 1980. Summarized in *Proc. 5th Int. Jnt. Conf. on Pattern Recognition*, Miami Beach, Florida, pp. 50-53, December 1-4, 1980.

[Kandel78] A. Kandel and W.J. Byatt, "Fuzzy Sets, Fuzzy Algebra, and Fuzzy Statistics," *Proc. IEEE*, Vol. 66, No. 12, pp. 1619-1639, December 1978.

[Kirby80] R.L. Kirby, "A Product Rule Relaxation Method," *Computer Graphics and Image Processing*, Vol. 13, No. 2, pp. 158-189, June 1980.

[Kirkpatrick83] S. Kirkpatrick, C.D. Gelatt, Jr., and M.P. Vecchi, "Optimization by Simulated Annealing," *Science*, Vol. 220, No. 4598, pp. 671-680, May 13, 1983.

[Kitchen80] L. Kitchen, "Relaxation Applied to Matching Quantitative Relational Structures," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-10, No. 2, pp. 96-101, February 1980.

[Lev77] A.Lev, S.W. Zucker, and A. Rosenfeld, "Iterative Enhancement of Noisy Images," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. 7, No. 8, pp. 435-442, June 1977.

[Montanari74] U. Montanari, "Networks of Constraints: Fundamental Properties and Applications to Picture Processing," *Information Sciences*, Vol. 7, pp. 95-132, 1974.

[Nagin82] P.A. Nagin, A.R. Hanson, and E.M. Riseman, "Studies in Global and Local Histogram-guided Relaxation Algorithms," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. PAMI-4, No. 3, pp. 263-277, May 1982.

[O'Leary80] D.P. O'Leary and S Peleg, *Analysis of Relaxation Processes: The Two-node, Two-label Case*, Computer Vision Laboratory Technical Report TR-967, University of Maryland, College Park, November 1980.

[Peleg78a] S. Peleg and A. Rosenfeld, "Determining Compatibility Coefficients for Curve Enhancement Relaxation Processes," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-8, No. 7, pp. 548-555, July 1978.

[Peleg78b] S. Peleg, "Iterative Histogram Modification, 2," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-8, No. 7, pp. 555-556, July 1978.

[Peleg80a] S. Peleg, "A New Probabilistic Relaxation Scheme," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. PAMI-2, No. 4, pp. 362-369, July 1980.

[Peleg80b] S. Peleg, "Monitoring Relaxation Algorithms Using Labeling Evaluations," *Proc. 5th Int. Jnt. Conf. on Pattern Recognition*, Miami Beach, Florida, pp. 54-57, December 1-4, 1980.

[Quam78] L.H. Quam, *Road Tracking and Anomaly Detection in Aerial Imagery*, Technical Note 158, Artificial Intelligence Center, SRI International, Menlo Park, California, March 1978.

[Richards80] J.A. Richards, D.A. Landgrebe, and P.H. Swain, "Overcoming Accuracy Deterioration in Pixel Relaxation Labeling," *Proc. 5th Int. Jnt. Conf. on Pattern Recognition*, Miami Beach, Florida, pp. 61-65, December 1-4, 1980.

[Riseman77] E.M. Riseman and M.A. Arbib, "Computational Techniques in the Visual Segmentation of Static Scenes," *Computer Graphics and Image Processing*, Vol. 6, No. 3, pp. 221-276, June 1977.

[Rosenfeld76] A. Rosenfeld, R.A. Hummel, and S.W. Zucker, "Scene Labeling by Relaxation Operations," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-6, No. 6, pp. 420-433, June 1976.

[Rosenfeld77a] A. Rosenfeld and L.S. Davis, *Iterative Histogram Modification*, Technical Report TR-519, Computer Science Center, University of Maryland, College Park, April 1977.

[Rosenfeld77b] A. Rosenfeld, "Iterative Methods in Image Analysis," *Proc. IEEE Conf. on Pattern Recognition and Image Processing*, Troy, New York, pp. 14-18, June 6-8, 1977.

[Rosenfeld78] A. Rosenfeld and L.S. Davis, "Iterative Histogram Modification," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-8, No. 4, pp. 300-302, April 1978.

[Rosenfeld82] A. Rosenfeld, "Picture Processing: 1981," *Computer Graphics and Image Processing*, Vol. 19, No. 1, pp. 35-75 (esp. p. 59), May 1982.

[Rosenfeld83] A. Rosenfeld, "Picture Processing: 1982," *Computer Vision, Graphics and Image Processing*, Vol. 22, No. 3, pp. 339-387 (esp. pp. 368-369), June 1983.

[Schachter76] B.R. Schachter, A. Lev, S.W. Zucker, and A. Rosenfeld, *An Application of Relaxation Methods to Edge Reinforcement*, Technical Report TR-478, Computer Science Center, University of Maryland, College Park, August 1976.

[Troy73] E.B. Troy, E.S. Deutsch, and A. Rosenfeld, "Gray-Level Manipulation Experiments for Texture Analysis," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-3, No. 1, pp. 91-98, January 1973.

[Ullman79] S. Ullman, "Relaxation and Constrained Optimization by Local Processes," *Computer Graphics and Image Processing*, Vol. 10, No. 2, pp. 115-125, June 1979.

[VanderBrug77] G.J. VanderBrug, "Experiments in Iterative Enhancement of Linear Features," *Computer Graphics and Image Processing*, Vol. 6, No. 4, pp. 25-42, April 1977.

[Waltz72] D.L. Waltz, *Generating Semantic Descriptions from Drawings of Scenes with Shadows*, Ph.D. Dissertation, Technical Report AI-TR-271, Artificial Intelligence Laboratory, Massachussetts Institute of Technology, Cambridge, Massachussetts, (AD-754-080), August 1972.

[Yamamoto79] H. Yamamoto, "A Method of Deriving Compatibility Coefficients for Relaxation Operators," *Computer Graphics and Image Processing*, Vol. 10, No. 3, pp. 256-271, July 1979.

[Yakimovsky73] Y. Yakimovsky and J.A. Feldman, "A Semantic Based Decision Theory Region Analyzer," *Proc. 3rd Int. Jnt. Conf. on Artificial Intelligence*, pp. 580-588, August 1973.

[Yakimovsky78] Y. Yakimovsky, "Boundary and Object Detection in Real World Images," *J. ACM*, Vol. 23, No. 4, pp. 599-618, October 1976.

[Zadeh65] L.A. Zadeh, "Fuzzy Sets," *Inform. Control*, Vol. 8, pp. 338-353, 1965.

[Zucker76] S.W. Zucker, "Region Growing: Childhood and Adolescence," *Computer Graphics and Image Processing*, Vol. 5, No. 3, pp. 382-399, September 1976.

[Zucker77] S.W. Zucker, R.A. Hummel, and A. Rosenfeld, "An Application of Relaxation Labeling to Line and Curve Enhancement," *IEEE Trans. on Computers*, Vol. C-26, No. 4, pp. 394-403 (end 922-929), April 1977.

[Zucker78a] S.W. Zucker, E.V. Krishnamurthy, and R.L. Haar, "Relaxation Processes for Scene Labeling: Convergence, Speed, and Stability," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-8, No. 1, pp. 41-48, January 1978.

[Zucker78b] S.W. Zucker and J.L. Mohammed, "Analysis of Probabilistic Relaxation Labeling Processes, *Proc. IEEE Conf. on Pattern Recognition and Image Processing*, Chicago, Illinois, pp. 307-312, May 31 - June 2, 1978.