SRI International

EVALUATION OF SCENE-ANALYSIS ALGORITHMS

Technical Note 332

August 1984

By: **Kenneth I. Laws**, Computer Scientist

Artificial Intelligence Center

Computer Science and Technology Division

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

ABSTRACT

A software evaluation methodology has been developed at SRI International for evaluating contributions to the ARPA/DMA Image Understanding Testbed. This paper describes the criteria that have shaped the evaluation methodology. Diverse examples of evaluation results are presented for the GHOUGH object detection system from the University of Rochester, the PHOENIX segmentation system from Carnegie-Mellon University (CMU), and the RELAX relaxation package from the University of Maryland.

1. Introduction

This paper describes software evaluation methods developed at SRI International to evaluate contributions to the ARPA/DMA Image Understanding (IU) Testbed. Examples of evaluation results are also presented.

The primary purpose of the 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:

- The establishment of a uniform environment that is as compatible as possible with the environments of research centers at universities participating in the IU research program. Thus, organizations obtaining copies of the testbed can receive a continuing flow of new results derived from ongoing research.

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

Evaluation of contributed scene-analysis techniques has thus received major emphasis in the testbed effort, with development of an efficient evaluation methodology a logically related goal. Software evaluation is difficult and few independent evaluations of IU software have been published. Analysis of an algorithm alone, even if feasible, would neither guarantee correct implementation nor quantify performance on realistic problems. Simple tabulations of pixel classification errors (as in Yasnoff et al. [Yasnoff 77]) would not be meaningful for complex scene-analysis tasks. Comparative evaluation by applying several algorithms or software packages to one set of test scenes (as in Ranade and Prewitt [Ranade 80]) was not practical for testing single algorithms. We at SRI have chosen a more subjective approach based on (1) careful analysis, (2) tests on simple and complex natural scenes, and (3) our own experience in image analysis. This is similar to the method advocated by Nagin et al. [Nagin 79].

This paper describes the methods we developed for evaluating the GHOUGH object detection system [Laws 82a] from the University of Rochester, the PHOENIX segmentation system [Laws 82b] from Carnegie-Mellon University (CMU), and the RELAX relaxation package [Laws 83a] from the University of Maryland. Other software packages have been contributed to the testbed but have not been as extensively evaluated.

2. Evaluation Purpose

There are many reasons for evaluating software packages. Managers, systems personnel, and users all have different perspectives and different requirements. These imply many questions that must be answered by a thorough evaluation effort. Some of the major questions are the following:

- Acquisition — Should the software package be acquired and further evaluated for local applications? What are its capabilities? Can it be extended?

- Implementation — What operating system support is required? How much memory does the package need? How much time does it take to run? Does the implementation correspond to the documented algorithm? Does performance match theoretical predictions? Has the code been well structured and adequately annotated? Is the documentation adequate?

- Application — Is the package suitable for a particular application? Is the user interface adequate? How does the package perform? Can it be integrated with other packages?

We have responded to these questions in our evaluation reports. The first section of each report introduces the package at a management level, discussing the tasks for which the algorithm is suited. Subsequent sections are written for system implementers and for users. The final sections document performance in evaluation tasks and make suggestions for future improvements.

An evaluation effort may have subsidiary effects on the software, the testbed, and the personnel involved:

- Adaptation — The evaluation effort has induced several authors to polish their software before releasing it to the IU testbed. Several contributions had to be translated into the C language before submission; the software thus became available on new classes of systems. Such "packaging" can be a significant step in the life of a software system.

- Validation — The processes of translating, installing, and evaluating contributed software have often led to the discovery of programming bugs and, occasionally, bugs in the algorithms. Where no bugs are found, there is greater assurance that such bugs do not in fact exist.

- Training — We, the evaluating personnel, had to learn to use the software and understand the theory behind it, thus enlarging the knowledgeable user community. We have documented this understanding and are communicating it to others through system demonstrations and discussions with potential users.

- Documentation — Submission of software for evaluation has often spurred initial documentation of the package. Any weaknesses in this documentation were brought to light as we learned to use the package. We have filled in the gaps and have added any necessary overview, literature survey, operating instructions, examples of performance, and suggestions for improvement.

We have also placed notes to future implementers and users in the source code and in the on-line man page documenting the testbed version of the contribution. (These man pages are included in the testbed programmer's manual [Laws 83b].) We believe that such channels of communication among users scattered through space and time are essential for continued development of the software.

- Augmentation — We have generally had to modify the submitted code to use local graphics and user interface routines, to instrument the code with additional displays or printouts of internal variables, and to rewrite portions of the code to eliminate trivial restrictions or to make the package more efficient for particular tasks. Although the dividing line between evaluation and new development is not easily discernible, it is still quite clear that the evaluation effort often leads to improvements in the software. The testbed environment also had to grow to support the contributions, and many ideas from the contributions have been adapted for use in other software.

3. Evaluation Structure

The tasks involved in evaluating a contribution are reflected in the structure of the evaluation report. We have developed this structure for recording and communicating the results of our investigations.

The introduction to a report summarizes the nature of the reviewed software package, the computer languages or system facilities needed to support it, and the contributions of various people in designing, creating, and maintaining it.

The background section, which follows next, describes the package from a management viewpoint. (This is one of the last sections written because it requires knowledge gained from the entire evaluation effort.) First there is a general description of the package, including its purpose, inputs, processing steps, and outputs. Then typical applications and usage scenarios are described, including preconditions and the domain of applicability, relation to preprocessor and postprocessor programs, applications that have been documented in the literature, and potential applications that we or other researchers have suggested.

The background section also discusses potential extensions and related applications. Potential extensions are applications that might be feasible if the package were modified or extended, used in a nonstandard fashion, or incorporated as an element of a larger system. Related applications are generalizations or variants of the standard applications for which other techniques seem to be more appropriate.

A descriptive section then documents the algorithm in detail. We begin with its historical development so as to introduce pertinent terminology and put the major technical issues into perspective. Literature references are cited to give credit where credit is due, to aid researchers in finding the full range of concepts that have been explored, and to furnish managers and implementers with contacts for further inquiries. The section closes with a detailed statement of the algorithm, including further discussion of design options and references to the literature as appropriate.

The next section is a brief implementer's guide describing the structure of the contributed software and the testbed locations of its source files, executable files, on-line documentation, and demonstration files. This is information needed to install and run the

package or to modify and maintain it. We have included here a description of the SRI modifications of the contribution.

A program documentation section then serves as a user's guide to running the package and invoking all the algorithm features. We have given instructions for both interactive and batch (or background) execution, including documentation of all command-line invocation options, interactive commands, controlling variables and flags, and status variables. Sometimes we have also found it necessary to give a detailed description of the program's execution phases, complementing the theoretical description of the algorithm in previous sections. This section of the evaluation report could be omitted whenever existing documents provide adequate and unified documentation of the program.

Our report can now document the evaluation proper. We have divided the evaluation section into two parts: the effects of parameter settings and the performance statistics for representative tasks. A subjective summary may also be included.

The purpose, intended effect, and legal values for each parameter and control variable were specified in the program documentation section. In this evaluation section we probe more deeply, determining the true effect of each parameter on system performance and documenting interactions (either constraints or synergistic effects) with other parameters. The end result is a set of rules for setting the parameters in various processing situations. We also comment on the usefulness of the control features and give suggestions for improving them.

Next we document the system's performance in handling selected scene-analysis tasks. (Task selection is discussed later in this paper.) We describe the test protocols, including the input images and the parameter settings we found optimal for the tasks. We present subjective and objective performance measures and summarize the apparent strengths and weaknesses of the algorithm and the software implementation.

Subjective trials are difficult to document. We ran hundreds of trials on dozens of images. Often a trial designed to investigate one effect would turn up something else as well. It is impractical to illustrate each of these findings in the final report. (Many are of the form "Note how the edge detector found this weak edge but missed that much stronger one.") We have therefore attempted to summarize our findings and present only the relevant information.

In the next report section, we suggest substantial modifications of the algorithm or the implementation. Some of these are of potential, but uncertain, immediate benefit and some are extensions into task areas far beyond those considered by the original author. We also mention known improvements in the contributing institution's continuing software development that have not been incorporated into the more stable testbed version. Many suggestions are derived from the work of other researchers, in which case we supply the appropriate references. Other suggestions arise from our own evaluation effort.

Our evaluation report concludes with a summary of the major technical concepts and of the strengths and weaknesses of the contributed algorithm and software. Appendices may give further information about the task domain, the algorithm, or the software package that, although too detailed for the main body of the report, is not readily available elsewhere.

4. Evaluation Methodology

We have focused our evaluation efforts on the topics of greatest utility. Issues of applicability and of parameter effects and interactions have been given the highest priority; issues of resource utilization have been assigned lower priority because they are dependent on the algorithm implementation and supporting hardware.

Some of the most difficult evaluation issues have to do with the theory underlying the algorithm. While we have attempted to summarize the theoretical basis of each contribution, evaluation of the theory is generally an impractical enterprise. The best we could do was to document other approaches to similar tasks and to note strengths and weaknesses of the algorithm as reported in the literature or found in our own work. For this reason we have included an extensive literature survey with each of our evaluation reports.

To evaluate a contribution fairly, we had to choose particular tasks for it to perform. Some delicacy was needed in making these choices. It would be unfair to evaluate the software in tasks considered unsuitable for it by the author. It would also be pointless to use only tasks that had been well documented in the literature; the essence of evaluation, after all, is the learning of something new. Consequently we tried to select tasks that were well within the contribution's domain and yet of interest to automated cartography and scene analysis.

The GHOUGH object detection system came with instructions for finding a distinctive lake in an aerial scene. We chose as additional tasks the finding of circular objects in aerial scenes and right-angled corners in oblique scenes. The PHOENIX segmentation system came with a test case of segmenting an image of an orange chair against a white background. We chose skyline analysis as a realistic task. The RELAX probabilistic relaxation system was set up for noise cleaning of an infrared image of a tank. We chose gradient edge detection and segmentation of vehicles from roads as additional tasks. In each case, the imagery was rich enough to allow subjective evaluation of performance on auxiliary problems (e.g., nonpurposive segmentation).

For each task we selected suitable imagery, ran dozens of trials to establish optimal parameter settings, and documented the results. If little documentation and operating information came with the package, we spent much of our time learning and recording this information. If documentation was adequate and only a small parameter space had to be explored, we were able to spend more time recording operating characteristics and performance statistics. Economic constraints limited the depth to which any task could be evaluated, but we were able to provide an adequate foundation for future researchers with specific problems.

The first step in evaluating any package was to get it working on a simple test image — usually one provided by the author. This process occasionally took considerable effort; we chose to rewrite the entire I/O structure and user interface for one program, for instance. While this integration effort was clearly essential to the development of the testbed, it did raise a thorny issue: to what extent should we fix perceived deficiencies and to what extent should we simply document them? One rule of thumb was that we would fix or extend the code in any manner required to carry out an evaluation with realistic tasks.

The next step was to test the software rigorously on one or more simple images. The idea was to become familiar with the workings of the program and with the options

available to the user. This step also helped identify software bugs or misunderstandings about the intended functions of the program. We strongly recommend the use of synthetic or well-understood problems as one phase of the evaluation effort.

Investigation of parameter interactions was one of the most difficult evaluation tasks. Analysis of simple imagery permitted us to concentrate on internal variables instead of interactions with a complex environment. Even so, the "search space" of possible interactions was immense. The PHOENIX program, for example, has 14 major threshold values to control image segmentation, in addition to various control strategies and interaction options.

One could navigate this complexity by using an intelligent control system to monitor thousands of runs and incrementally modify parameter settings to optimize some performance criterion. While such a "hill-climbing" approach is feasible [Lenat 82], it would have provided only a superficial understanding of *why* the identified parameter sets were optimal combinations. We chose instead to analyze the program structure, experiment with carefully chosen parameter values, and study the overall execution of each computer run. Often we had to disable features of the program so as to study one feature in isolation.

The final experimental step was to evaluate the software for realistic tasks on "natural" imagery. This proved to be exceedingly difficult because the space of input imagery was impossibly large. If a program could detect circular tanks in one image, for instance, would it be able to detect them at different image resolutions? With different levels of contrast and blur? With strong shadows and highlighting present? With occlusions, unusual edge alignments, or texture effects? Would it be able to distinguish real targets (possibly camouflaged) from decoys and targets already destroyed?

Such questions go beyond the scope of this initial evaluation effort. We tried our best, however, to get a "feel" for each program's capabilities. We varied pertinent imagery variables and carefully noted the effects. Anomalies were checked out by instrumenting the code or by experimentation on simple images. We believe that this intelligent experimentation is at least as useful as extensive statistical validation.

Unfortunately we were not able to devise rigorous performance metrics for tasks such as "target detection." We tuned the analysis system carefully for each problem and reported the best performance that could be obtained. (We tried to avoid tuning the system for each individual image, however. A single parameter set or operating procedure was developed for each task.) The results are necessarily subjective and would vary slightly for other tasks, other imagery, or other experimenters.

5. Evaluation Examples

It is difficult to convey the scope and variety of our evaluation results in a short paper. The PHOENIX report alone is more than 80 pages long, with 25 pages devoted to performance evaluation and suggestions for future development. We shall illustrate the evaluation results by presenting short excerpts from the reports. Different reports will be used to illustrate different points regarding the nature of the evaluation effort.

5.1. Theory

We have documented the theoretical basis and the implementation of each contribution

from several perspectives. We have provided theoretical justifications and mathematical notations where appropriate, and have then related this information to the parameters and commands of the software packages. Sometimes it was quite difficult to extract this information from the technical literature.

The RELAX system, for instance, could be regarded as a general method for local modification of constraint and compatibility information stored in the nodes of a rectilinear graph. The initial label probabilities at the nodes may be derived from image pixel intensities and the final label assignments mapped back to pixel intensities, but the iterative relaxation technique is independent of image-domain considerations. Much of the theoretical work on relaxation has abandoned the rectilinear image plane and has dealt with constraint relations on arbitrary graphs with varying numbers of neighbors for each node.

For the evaluation, we extracted the updating equations actually used in the software package and expressed them in terms commonly found in the theoretical literature. The RELAX package includes both the Hummel-Zucker-Rosenfeld additive updating scheme and the Peleg multiplicative updating scheme. Here is part of our description of the former:

> The goal of the relaxation algorithm is to update the values of the probabilities associated with a node so as to reflect the compatibility of neighboring labels. The $(t+1)$ update of the kth label value is calculated from the previous time (t) update by
>
> $$q_i^{(t)}(\lambda_k) = \frac{1}{m}\sum_j \left\{ \sum_{\lambda'=\lambda_1}^{\lambda_n} r_{ij}(\lambda_k,\lambda') p_j^{(t)}(\lambda') \right\}$$
>
> $$p_i^{(t+1)}(\lambda_k) = \frac{p_i^{(t)}(\lambda_k)\left(1+q_i^{(t)}(\lambda_k)\right)}{\sum_{\lambda=\lambda_1}^{\lambda_n} p_i^{(t)}(\lambda)\left(1+q_i^{(t)}(\lambda)\right)} \quad ,$$
>
> where j indexes the m neighbors of node i; $r_{ij}(\lambda_k,\lambda')$ is the compatibility coefficient for node i with label λ_k and neighboring node j with label λ'; $q_i(\lambda_k)$ can be thought of as the assessment by the neighboring nodes that node i should be labeled λ_k; and $p_i(\lambda_k)$ is the assessment by node i as to its own label. These two assessments are combined to produce an updated probability, $p_i(\lambda_k)$.
>
> The compatibility coefficients may be negative if the labels are incompatible, positive if the labels are compatible, and zero if they are independent. While it is possible to define the compatibility coefficients in terms of conditional probabilities, it is overly restrictive to do so. The compatibility coefficients for the Hummel-Zucker-Rosenfeld rule are based on information theory; mutual information defines the compatibility coefficients and provides a mechanism for calculating them:
>
> $$r_j(\lambda_k,\lambda_l) = \frac{1}{5}\ln\frac{w\sum_i p_i(\lambda_k)p_{i_j}(\lambda_l)}{\sum_i p_i(\lambda_k)\cdot\sum_i p_i(\lambda_l)} \quad ,$$

Table 1: PHOENIX Segmentation Parameter Sets

Parameter	Strict	Moderate	Mild
depth	4	10	20
splitmin	200	40	20
hsmooth	25	9	5
maxmin	300	160	130
absarea	100	30	5
relarea	10	2	1
height	50	20	10
absmin	2	10	30
intsmax	2	3	6
isetsmax	2	3	5
absscore	925	700	600
relscore	95	80	65
noise	50	10	5
retain	4	20	40

where k and l range over the n labels, i ranges over all w nodes of the graph, and j specifies the particular neighbor (e.g., upper-left) of the ith node. For each node i, $r_{ij}(\lambda_k, \lambda_l)$ is equal to $r_j(\lambda_k, \lambda_l)$ clipped to be in the closed interval $[-1, 1]$.

In the report we have discussed the meaning of these terms and methods of estimating or setting the numeric values, as well as the effects of relaxation in various image analysis tasks.

5.2. Analyses

The following are a few results from the performance analyses conducted on the PHOENIX and GHOUGH systems. Space limitations prevent a full description of all of the terms used, but the examples should give some feeling for the level of understanding a thorough evaluation might require.

The PHOENIX segmenter is a moderately complex system with 14 user-settable variables that control the segmentation process itself. The original contribution came with very little guidance about setting these parameters to achieve reasonable segmentations. One of our evaluation tasks was to create such information.

We began by finding a set of "moderate" parameter values that would segment simple scenes of large objects down to the level of major subregions. Then we developed a " strict" set of parameter values for producing a very coarse segmentation by precluding most potential region splits. Finally we developed a "mild" set of parameter values for complete or overly permissive segmentation down to the level of very small regions or regions with small differences from their neighbors.

Each column in Table 1 lists one of these parameter sets. The user need only select the extent of segmentation desired and load the corresponding parameters. We thus reduced

the 14 parameters to a manageable single decision that is relatively independent of the image content. Additional flexibility is possible by switching parameter sets during a segmentation run to control the fineness of segmentation within particular image regions.

It will occasionally be necessary for the user to deviate from the recommended parameter settings. To make this possible, we have evaluated each parameter individually. Here is part of the **maxmin** parameter description:

> **Maxmin** is the minimum acceptable ratio of apex height to higher shoulder for an interval in the histogram. Any interval failing this test is merged with the neighboring interval on the side of the higher shoulder. The test is then repeated on the combined interval. The overall effect on a set of candidate thresholds is to eliminate those that are on the sides or tops of major peaks.
>
> **Maxmin** is a powerful heuristic. With strict smoothing and with all other heuristics disabled, **maxmin** alone is able to produce reasonable segmentations. It is even more powerful when combined with area heuristics. With mild or moderate smoothing, **maxmin** passes clusters of thresholds in the noise regions between major peaks. This is fine if the clusters can be thinned by the **absarea** and **relarea** heuristics, but a poor selection may ensue if they are left for the **intsmax** heuristic.
>
> The problem here is that PHOENIX has no "quality" score for histogram valleys. It assumes that cutpoint bin height is an adequate measure, whereas width and depth relative to the neighboring peaks are also important. PHOENIX can incorporate such knowledge only by smoothing the histogram; the amount of smoothing required depends on how separated the peaks are.

The next step in the PHOENIX evaluation was investigation of a skyline delineation task. One of the test images, *portland*, shows a city skyline against a cloudy sky. After describing segmentation performance on reduced versions of this and other images, we reported the following:

> A test sequence was run on the full-resolution (500x500) *portland* image. Strict and even moderate heuristics were unable to segment the image when only the red, green, and blue feature planes were used; it was necessary to use the mild heuristics. The best approach would be to start the segmentation with mild thresholds and then return to strict or moderate ones for segmenting the subregions. Instead, we avoided such special interference and ran the segmentation to completion with mild heuristics. The full run (which, with the 'v' flag set, generated 19,000 lines of printout) required 33 minutes of CPU time:

Phase	Real	CPU
Histogram	0:04:13	0:02:32
Interval	0:18:12	0:07:27
Threshold	0:10:00	0:03:47
Patch	0:03:51	0:03:30
Collect	0:38:12	0:14:04
Segmentation	1:18:15	0:32:34

The final segmentation into 1182 regions (including nearly every window of every building) was much more successful than the original attempt, but still had difficulty distinguishing a glass-surfaced building from the sky it reflected.

Later test runs showed even better performance when color transforms were used in addition to the three original color planes. On the basis of our experience with these tests, we were able to suggest operating procedures for PHOENIX in similar tasks.

The evaluation of the GHOUGH object detection system was similar. Because GHOUGH had fewer parameters, we were able to spend more time analyzing system performance in realistic tasks. One thrust of this effort was to develop an understanding of specific operational characteristics, as in the paragraphs below.

The requirement of sharp edges does not imply that smooth, continuous object boundaries are essential. The program is quite tolerant of noise in the outline and is able to find irregular, incomplete, or discontinuous shapes. The circle template, for instance, often responds to forest clearings, treetops, road intersections, and curved embankments, as well as to square buildings and to image "hot spots." The irregularities in these image structures spread the vote cluster in the accumulator, but the local maximum may still be above the general noise level.

Shadow edges usually fit the requirement for strong, sharp edges. It is often easier to find a shadow than the object that cast it. This may be a useful cueing technique, but must be used carefully to avoid reporting objects at incorrect locations. A similar problem exists with high-resolution imagery: the position reported for a portion of an object (e.g., the circular top of a storage tank) may not correspond to the position of the whole object.

These characteristics mean that the program is best suited to three tasks: locating industrial parts in high-contrast imagery; counting numerous, obvious, similar objects such as storage tanks, barracks, or microscopic particles; and precisely positioning a template when an approximate location has been cued by the user or another system. Even for these applications, the program must be supervised and its output edited. Other applications will necessitiate further development of the technique.

Sometimes our results were quite unexpected, as when we found that increasing the number of points in the search template definition could actually diminish target location accuracy. This occurred because the template points were entered at discrete points on a Cartesian grid, and close spacing of the points caused severe quantization of the angular relationships among them. (We proposed several possible cures for this problem.)

We were able to quantify system performance in representative tasks. Some of the domain-independent equations are given below. (The terms are fully explained in the evaluation report.) We have attempted to base the formulas on important characteristics of the GHOUGH algorithm, although the coefficients had to be estimated empirically.

$$\begin{aligned}\text{Edge time} &= .00036(\text{window points}) + .0053(\text{edges found}) \\ &\quad + .00019(\text{accumulator entries}) + (\text{additional paging time})\end{aligned}$$

$$\begin{aligned}
\text{Analysis time} &= 10^{-4}(\text{search volume}) \\
&\quad \times (.08 + 2.6\log(1 + \text{accumulator density})) \\
&\quad + (\text{additional paging time}) + .0025(\text{maxima found}) \\
\text{Maxima} &= .023(\text{search volume})^{0.8}(1 + \text{accumulator density})^2 - 1 \\
\text{Noise} &= 2.04(\text{search volume})^{0.09}(1 + \text{accumulator density})^{0.8} - 1
\end{aligned}$$

Such formulas would be very helpful in designing an improved version of GHOUGH. Even more exciting is the possibility of building an expert image-analysis system that would include GHOUGH as a component. The knowledge base of such a system would record predictive formulas and other operating characteristics in a form that could be used by both humans and machines.

Some of the GHOUGH parameters are dependent on image content. These were very difficult to quantify, but we attempted to document the dependencies well enough so that users could adapt our findings to their own imagery. The following is our discussion of GHOUGH performance as a function of the edge detection threshold.

The number and density of edges detected in an image are sigmoid (s-shaped) functions of edge threshold similar to cumulative-frequency histograms. GHOUGH operates best when 10% to 20% of the pixels are classified as edge points, although it will usually work well at any edge density above 6%. Some typical threshold values to achieve specified edge densities are

Scene Type	6%	12%	25%	50%
Cloudy sky	42	35	28	20
Aerial terrain	160	120	80	40
Aerial target area	200	180	120	60
Low-angle urban	260	200	140	90
Forest cover	280	220	160	100
Aerial urban	720	600	480	340

In general, it is better to use too low a threshold: this will increase chances of finding target edges while only slightly raising the noise level, and the edges found are likely to be the most reliable ones. The main drawback is that low thresholds increase the time required to fill the accumulator with votes. A reasonable starting guess is a threshold of 120.

As we experimented with the software packages, we noted a great many characteristics that could be improved. The preceding GHOUGH edge threshold sensitivity, for instance, led us to suggest that an adaptive edge detector be used. Our suggestions have covered everything from the algorithm to the characteristics of larger systems that might incorporate these routines.

5.3. Summaries

We have also tried to summarize our findings, drawing on our experience with other image analysis systems. Each of the reports ends with such a summary. For the PHOENIX system, our observations included the following:

> The PHOENIX segmentation system is one of several existing systems for recursively segmenting digital images. Its major contributions are the optional use of multiple thresholds, spatial analysis for choosing among good features, and a sophisticated control interface. Some of the strengths and weaknesses of the PHOENIX algorithm are listed below.
>
> PHOENIX, like other region-based methods, always yields closed regional boundaries. This is not true of edge-based feature extraction methods, with the possible exception of boundary following and zero-crossing detection. Closed boundaries are the essence of segmentation and greatly simplify certain classification and mensuration tasks.
>
> PHOENIX is a hierarchical or recursive segmenter, which means that even a partial segmentation may be useful. This can save a great deal of computation if efforts are concentrated on those regions where further segmentation is critical. If PHOENIX is to be driven to its limits, other methods of segmenting to small, homogeneous regions may be more economical.
>
> PHOENIX is relatively insensitive to noise. Thresholds are determined by the feature histograms, where noise tends to average out. This contrasts with edge-based methods, where the local image characteristics can be highly perturbed by noise.
>
> PHOENIX has no notion of boundary straightness or smoothness. This may be good or bad, depending on the scene characteristics and the analysis task. It easily extracts large homogeneous regions that may be adjacent to detailed, irregular regions (e.g., a lake adjacent to dock areas or sky over a city); such tasks can be difficult for edge-based segmenters.
>
> PHOENIX tends to miss small regions within large ones because they contribute so little to the composite histogram. It is thus poorly suited for detecting vehicles and small buildings in aerial scenes, although there may be ways to adapt it to this use. It also tends to misplace the boundary between a large region and a small one, thus obscuring roads, rivers, and other thin regions. Boundaries found by edge-based methods are less affected by distant scene properties.
>
> PHOENIX may also fail to detect even long and highly visible boundaries between two similar regions if the region textures cause their histograms to overlap. Edge-based methods are better able to detect local variations at the boundary.
>
> Since perfect segmentation is undefined, PHOENIX must oversegment an image in order to find all regional boundaries that may be of use to any higher-level process. It is left for a segmentation-editing step to merge segments that

have no usefulness for some particular purpose. Without such a step, or indeed even a purpose, it is very difficult to evaluate the segmenter output.

6. Conclusions

Our evaluation efforts have documented a great many suggestions for improving the evaluated software. We have tried to be as quantitative and rigorous as possible, but the results are necessarily subjective. Often we have functioned in the manner of restaurant or theater critics, reporting our impressions of the contributions. These informed opinions, combined with our more rigorous documentation, should provide a good basis for more specific evaluation efforts directed at particular task scenarios and production environments. Our evaluation reports and the SRI testbed environment make the contributed programs available as benchmark systems and as research tools.

Acknowledgment

This research was supported by the Defense Advanced Research Projects Agency under Contract No. MDA903–79–C–0588.

References

[Laws 82a] K.I. Laws, *The GHOUGH Generalized Hough Transform Package: Description and Evaluation,* Technical Note 288, Artificial Intelligence Center, SRI International, Menlo Park, California (December 1982).

[Laws 82b] K.I. Laws, *The PHOENIX Image Segmentation System: Description and Evaluation,* Technical Note 289, Artificial Intelligence Center, SRI International, Menlo Park, California (December 1982).

[Laws 83a] K.I. Laws, *The RELAX Image Relaxation System: Description and Evaluation,* Technical Note 301, Artificial Intelligence Center, SRI International, Menlo Park, California (August 1983).

[Laws 83b] K.I. Laws, *The DARPA/DMA Image Understanding Testbed Programmer's Manual,* Technical Note 298, Artificial Intelligence Center, SRI International, Menlo Park, California (January 1984).

[Lenat 82] D.B. Lenat, W.R. Sutherland, and J. Gibbons, "Heuristic Search for New Microcircuit Structures: An Application of Artificial Intelligence," *The AI Magazine*, Vol. 3, No. 3, pp. 17–33 (Summer 1982).

[Nagin 79] P. Nagin, R. Kohler, A. Hanson, and E. Riseman, "Segmentation, Evaluation, and Natural Scenes," *Proc. IEEE Conf. on Pattern Recognition and Image Processing*, Chicago, Illinois, pp. 515–522 (6–8 August 1979).

[Ranade 80] S. Ranade and J.M.S. Prewitt, "A Comparison of Some Segmentation Algorithms for Cytology," *Proc. 5th Int. Jnt. Conf. on Pattern Recognition*, Miami Beach, Florida, pp. 561–564 (1–4 December 1980).

[Yasnoff 77] W.A. Yasnoff, J.K. Mui, and J.W. Bacus, "Error Measures for Scene Segmentations," *Pattern Recognition*, Vol. 9, pp. 217–231 (1977).