White-Box Evaluation of Computer Vision Algorithms through Explicit Decision-Making

Richard Zanibbi[1], Dorothea Blostein[2], James R. Cordy[2]

[1] Dept. Computer Science, Rochester Institute of Technology, Rochester, NY 14623-5608
[2] School of Computing, Queen's University, Kingston, Ontario, Canada, K7L 3N6
rlaz@cs.rit.edu {blostein, cordy}@cs.queensu.ca

Abstract. Traditionally computer vision and pattern recognition algorithms are evaluated by measuring differences between final interpretations and ground truth. These black-box evaluations ignore intermediate results, making it difficult to use intermediate results in diagnosing errors and optimization. We propose "opening the box," representing vision algorithms as sequences of decision points where recognition results are selected from a set of alternatives. For this purpose, we present a domain-specific language for pattern recognition tasks, the Recognition Strategy Language (RSL). At run-time, an RSL interpreter records a complete history of decisions made during recognition, as it applies them to a set of interpretations maintained for the algorithm. Decision histories provide a rich new source of information: recognition errors may be traced back to the specific decisions that caused them, and intermediate interpretations may be recovered and displayed. This additional information also permits new evaluation metrics that include false negatives (correct hypotheses that the algorithm generates and later rejects), such as the percentage of ground truth hypotheses generated (*historical recall)*, and the percentage of generated hypotheses that are correct *(historical precision)*. We illustrate the approach through an analysis of cell detection in two published table recognition algorithms.

Keywords: Performance Evaluation, Document Recognition, Table Recognition, Scripting Languages, Domain-Specific Languages

1 Introduction

Current evaluation methods for computer vision and pattern recognition systems are generally *black box*, based solely on observing algorithm inputs and outputs. For example, black-box analysis is standard practice in the document recognition community [1][8][10][11][14][16][17]. Diagnosis of failures is difficult, since black-box observations provide little insight into the causes of poor recognition results. In response to this lack of information, developers who are searching for sources of recognition errors typically write additional code to produce diagnostic output. This is a laborious and error-prone task.

Significant advances can be achieved through *white box* evaluation. We present a technique that explicitly represents decision points where *interpretations* (recognition results) may be altered, and captures decision outcomes at run-time [20][23]. In this paper we summarize our earlier work, and present new recall and precision-based performance summaries for both complete strategies and individual decisions.

As illustrated in Figure 1, decision points represent the input and output spaces for a decision separately from the function that makes the decision during execution. We represent decision points using the Recognition Strategy Language (RSL), introduced in Section 2. Decision functions themselves may be implemented in any programming language. Section 3 presents new metrics computed using the decision history recorded by RSL programs during execution. An analysis of two published algorithms for table-cell detection [5][7] demonstrates how the technique provides novel insights into algorithm performance, and valuable information for algorithm optimization (Section 4).

2 The Recognition Strategy Language

We represent algorithm decision points as a sequence of operations in the *Recognition Strategy Language* (RSL) [23]. RSL is a scripting language that coordinates the recognition process, maintaining a set of current interpretations while invoking decision functions at run time. In RSL, *what* the algorithm decides is represented separately from *how* the algorithm makes the decisions (see Fig. 1). The six primary decision types are *adaptation* of recognition parameters, *classification* of input regions (e.g. bounding boxes), *segmentation* of input regions, *formation of relations* among regions, *rejection* of parts of an interpretation, and *acceptance/rejection* of complete interpretations.

Each decision point formally defines a decision type, decision function call, and the input and output space of the decision. In Fig. 2, the first decision point takes the set of *Word* regions provided in the input, and then returns one or more sets of *Word* pairs representing horizontal adjacencies (i.e. updating the binary relation *hor_adj*); if

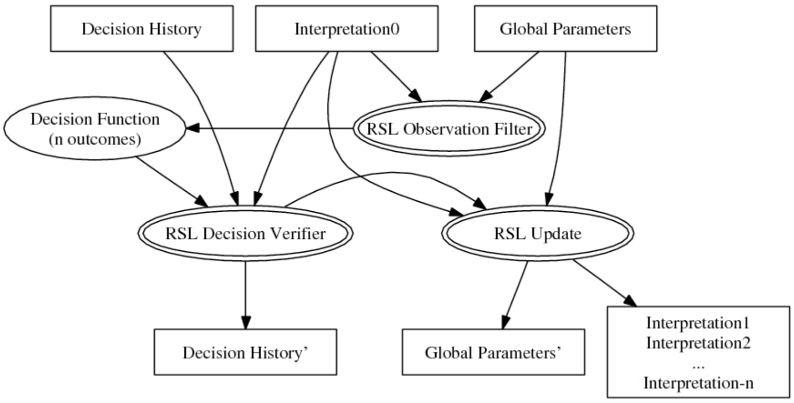


Fig. 1. An RSL decision point. Here a *decision function* produces *n* interpretations for a single input interpretation. Each possible decision outcome contains one or more legal alternatives for the decision type (e.g. valid classes for a classification). The decision-making environment includes a decision history, global parameter set, and RSL book-keeping operations. These operations control the visibility of parameters and interpretations for decision functions, verify outcomes, and update the history, global parameters, and interpretation set

more than one set of *Word* pairs is returned, RSL produces multiple interpretations (see Fig. 1). This decision is made by the external function *selectHorizAdjRegions,* which is passed the *Word* regions and the parameter *sMaxHorDistance* (the prefix *s* indicates ‘static,’ i.e. a constant). The second decision point (the *segment*) operation will be applied to *each* of the interpretations output by the previous decision. While not shown here, RSL provides a simple conditional statement to prevent interpretations from being modified at a decision point [23].

To ensure that decision inputs are explicit, RSL hides all interpretation elements and parameters not given in the decision type definition or decision function call. This allows dependencies between region types, relation types, and parameters to be determined by static analysis of RSL source code [23]. For example, at the second decision point (the *segment* decision) *Word* regions are visible, but the *hor_adj* relation must be indicated using the *observing* keyword to be visible to the external decision function (*segmentHorizAdjRegions). Classify* decisions have ‘neither’ (reject) defined by default. For example, *Cells* may be classified as *Header, Entry,* or neither at the third decision point shown in Fig. 2.

Interpretations are represented as attributed graphs, with nodes representing locations for regions of interest, which may have associated types (e.g. bounding box ROIs labeled as *Words* and/or *Cells*), and edges representing region relationships [23]. For example, the built-in RSL *contains* relation encodes region membership, such as the *Cell* regions that belong to a *Column* region.

Decision functions may be implemented in any programming language, provided they are 'wrapped' in order to accept interpretations and parameters from RSL and return text-based decision records. We have prototyped an RSL implementation using the TXL source transformation language [3].

A sequence of decisions produces an *interpretation tree* that RSL records through the history of decision outcomes (see Fig. 1), and by annotating changes directly within interpretations themselves. Every region and relation hypothesis in an interpretation is annotated with a list of the decisions that accepted or rejected it. For example, the annotation “`H 1 -2 3″` represents acceptance by decision 1, rejection

```
model regions
    *REGION Word Cell Header
           Entry Image
end regions

model relations
    *contains adj_right close_to
end relations

recognition parameters
    sMaxHorDistance 5.0  % mms
    aMaxColSep 20.0      % mms
    aMaxHorSep 25.0      % mms
end parameters
```

```
strategy main
    relate { Word } regions with { hor_adj } using
      selectHorizAdjRegions(sMaxHorDistance)

    segment { Word } regions into { Cell }
      using segmentHorizAdjRegions()
      observing { hor_adj } relations

    classify { Cell } regions as { Header, Entry }
      using labelColumnHeaderAndEntries()

    accept interpretations
end strategy
```

Fig. 2. RSL program recognizing table *Cell* regions from *Word* regions. The *model* section defines legal region and relation types. *Recognition parameters* are constants and variables used by the external decision functions. The RSL *strategy function* ‘main’ defines the sequence of decision points, here using *relate, segment, classify* and *accept* decision types

by decision 2, and reinstatement by decision 3 (e.g., for a *Cell* comprised of *Word* regions). This information is used to compute new performance metrics described in Sections 3-4.

For white-box analysis of an *existing* program, decision points to be observed are coded in RSL, which invoke 'wrapped' decision functions called by the RSL program. If analysis is based on published algorithm descriptions, then re-implementation of the decision functions is needed; this is how we carried out the evaluation of table-cell detection algorithms in Section 4, using TXL to program the decision functions. The granularity of decision points can be chosen freely. To model the system as a handful of coarse components, use a small number of decision functions that perform complex computations. To model the system in greater detail, use a large number of simpler decision functions.

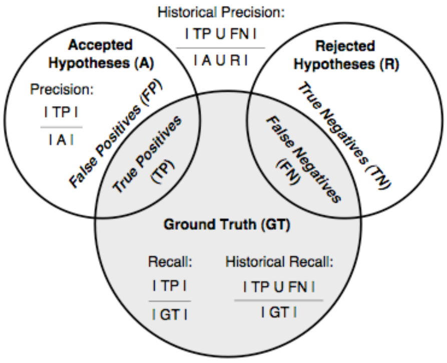
3 Decision-Based Performance: Historical Recall and Precision

The accuracy of individual decisions in an RSL program can be determined using the decision history. The RSL source location of decisions that produce false-positive and false-negative hypotheses can also be determined, as discussed in Section 4. Using existing programming practices, it is difficult to obtain this kind of information, because decision points are not explicit in the code, and intermediate decision information is not available in the output.

Recall and precision are commonly used metrics for evaluating detection tasks. *Recall* is the percentage of ground truth hypotheses present in an interpretation, whereas *precision* is the percentage of accepted hypotheses that match ground truth. We may also use historical versions of these measures, which account for rejected hypotheses [22]. Both conventional and historical recall and precision can be evaluated at any point during algorithm execution. The relationship between conventional and historical recall and precision is illustrated in Fig. 3.

Recall is defined by $|TP| / |GT|$, whereas precision is defined as $|TP| / |A|$. *Historical recall* also takes valid rejected hypotheses into account ($|TP \cup FN| / |GT|$), while *historical precision* takes false negatives into account along with the complete set of generated hypotheses ($|TP \cup FN| / |A \cup R|$). If an algorithm never rejects hypotheses ($R = \emptyset$), then the conventional and historical versions of recall and precision are the same. Historical recall is always greater than or equal to recall: historical recall never decreases during execution, whereas recall can decrease as hypotheses are rejected. Precision measures the accuracy of *accepted* hypotheses, while historical precision measures the accuracy of *generated* hypotheses.

Here we discuss only historical recall and precision. Corresponding historical versions of other black-box evaluation metrics can be defined, for example, weighted recall and precision metrics using areas of region overlap [1][10]; Vector Detection Rate, Vector False Alarm Rate, and Vector Recovery Rate used in the GREC arc segmentation contests [19]; and other performance metrics used in document-image analysis competitions [2].



A	The set of accepted hypotheses
R	The set of hypotheses that were accepted and later rejected. At any given time, a hypothesis is accepted or rejected, but not both
GT	A set of ground-truth declarations
TP	The set of true positives, $A \cap GT$
FN	The set of false negatives, $R \cap GT$

Fig. 3. Recall, Precision, Historical Recall, and Historical Precision

4 Illustration: Evaluating Algorithms for Table-Cell Detection

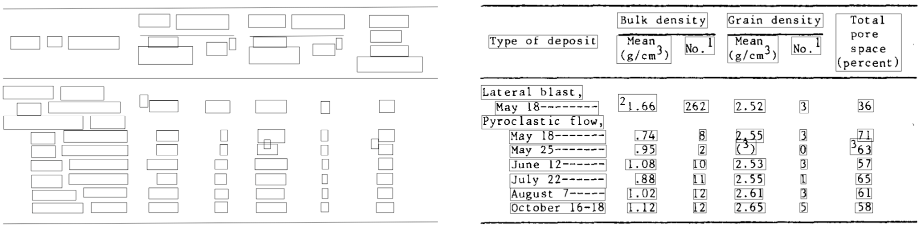
To illustrate the new information provided by white-box analysis, we present an error analysis for cell detection performed by two table-recognition algorithms: Handley [5] and Hu et al. [7], which we refer to as "the Handley algorithm" and "the Hu algorithm" respectively. We chose the Handley [5] and Hu [7] algorithms for our study because they are described in enough detail to permit replication, and exhibit a degree of sophistication.

In this comparison, table-cell detection involves combining *Word* regions to form *Cell* regions, as illustrated in Fig. 4. For our experiments, the input contains *Word* regions, and may also contain *Line* regions if the table is ruled. Our evaluations use test data for which the ground-truth is known; difficulties with defining table ground truth are discussed in [6][12]. Further discussion of table recognition algorithms is presented in [4][9][13][18][21].

The Handley algorithm [5] is a geometry-based approach, making no use of the characters contained in *Word* regions. As illustrated in Fig. 5, the algorithm first hypothesizes that every *Word* region is a *Cell*. *Cell* regions are then merged in a series of steps using weighted projection profiles and cell adjacency relationships.

The Hu algorithm [7] ignores lines, and *Word* regions are classified as alphabetic or non-alphabetic, based on whether more than half the characters in the region are alphabetic. Columns are detected using a hierarchical clustering of the horizontal spans of *Word* regions. Next the table boxhead, stub and rows are located, and then header cells in the boxhead are identified (shown in Fig. 5, Decision 86). Finally cells in the body of the table are identified. We have added a simple step to define text-lines, by projecting *Word* regions onto the Y-axis and defining text-lines at gaps greater than a threshold value. This is necessary because the Hu algorithm is for text files, where text-lines are given.

We re-implemented the Handley and Hu algorithms in RSL and TXL using the published descriptions in [5] and [7] (Handley: 540 lines of RSL, 5000 lines of TXL; Hu: 240 lines of RSL, 3000 lines of TXL, in both cases including comments). For both algorithms, our choice of decision points was informed by the regions and relations that we wished to analyze. The RSL programs may be found elsewhere [20].

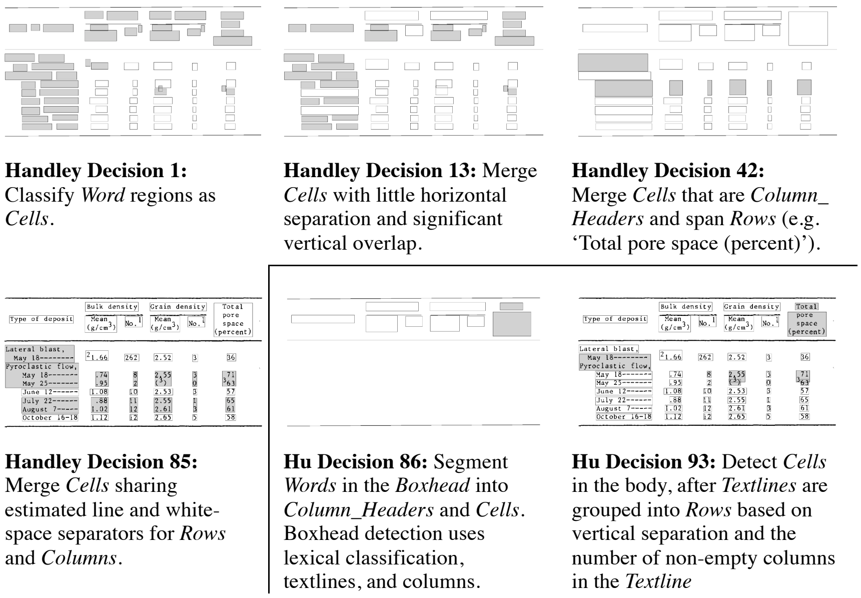


Type of deposit	Bulk density		Grain density		Total pore space (percent)
	Mean (g/cm^3)	No.[1]	Mean (g/cm^3)	No.[1]	
Lateral blast,					
May 18--------	[2]1.66	262	2.52	3	36
Pyroclastic flow,					
May 18-------	.74	8	2.55	3	71
May 25-------	.95	2	([3])	0	[3]63
June 12------	1.08	10	2.53	3	57
July 22------	.88	11	2.55	1	65
August 7-----	1.02	12	2.61	3	61
October 16-18	1.12	12	2.65	5	58

(a) *Word* and *Line* regions (input)

(b) Ground-truth *Cell* regions (ideal output)

Fig. 4. Illustration of the table-cell detection task used in our experiments: *Word* and *Line* regions (a) are analyzed to produce *Cell* regions, which are sets of *Word* regions (b). In our experiments, we manually delimit the *Word* and *Line* regions (a), labeling *Word* regions as alphabetic or other, and we also manually delimit the ground-truth *Cell* regions (b). This table is from page a038 of the University of Washington Database [15]



Handley Decision 1: Classify *Word* regions as *Cells*.

Handley Decision 13: Merge *Cells* with little horizontal separation and significant vertical overlap.

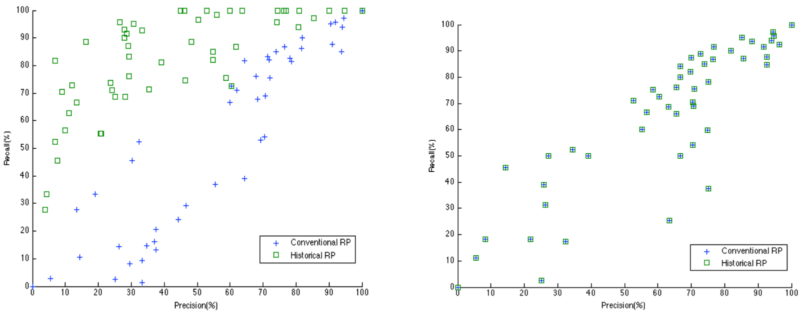
Handley Decision 42: Merge *Cells* that are *Column_ Headers* and span *Rows* (e.g. 'Total pore space (percent)').

Handley Decision 85: Merge *Cells* sharing estimated line and white-space separators for *Rows* and *Columns*.

Hu Decision 86: Segment *Words* in the *Boxhead* into *Column_Headers* and *Cells*. Boxhead detection uses lexical classification, textlines, and columns.

Hu Decision 93: Detect *Cells* in the body, after *Textlines* are grouped into *Rows* based on vertical separation and the number of non-empty columns in the *Textline*

Fig. 5. Intermediate results for some RSL decision points changing accepted *Cell* hypotheses, for the Handley and Hu algorithms given the table in Fig. 5. Gray *Cell* regions are incorrect (false positives), and unshaded *Cell* regions are correct (true positives). Handley Decision 1 accepts 75 cell hypotheses, of which 43 are incorrect. Handley Decision 13 accepts 8 hypotheses (of which 7 are correct) and rejects 16 hypotheses (all of which are incorrect)



(a) Handley Algorithm (b) Hu Algorithm

Fig. 6. Precision vs. Recall of *Cell* regions for the Handley (a) and Hu (b) algorithms (58 tables). Conventional recall and precision is shown by '+', and historical recall and precision by a box. Historical metrics characterize generated *Cell* regions, including those rejected

4.1 Detailed Error Analysis For a Single Table

Intermediate recognition results can be easily captured when a decision-based algorithm representation is used. A visual display of intermediate states is illustrated in Fig. 5, showing the *Cell* hypotheses at selected decision points in the execution of the Handley and Hu algorithms.

From the recorded decision history, we can determine that Decision 42 for the Handley algorithm in Fig. 5 is particularly good: it accepts the only available hypothesis matching ground truth (the upper-right hand header cell), while rejecting two invalid hypotheses. In contrast, Handley Decision 85 (the final interpretation) creates 6 invalid hypotheses, and rejects 12 valid hypotheses. The Hu algorithm uses only two decisions to identify *Cell* regions: Hu Decision 86 identifies header cells, and Hu Decision 93 identifies cells in the table body. The visualizations in Fig. 5 were produced directly from RSL output, which includes decision times that can be matched to specific decision points in the RSL source. This directly locates the part of the source code that caused each recognition error. A decision that 'causes' an error in the output may have been influenced by a poor decision made earlier. Decision histories allow us to also observe which earlier decision points produced the alternatives considered at a decision point.

4.2 Comparing Conventional and Historical Metrics for a Set of Tables

We now compare conventional and historical recall and precision for the Handley and Hu algorithms, for a test set consisting of 58 tables taken from the University of Washington Database [15]. For the final interpretations produced by each algorithm, recall and precision information is shown for individual tables in Fig. 6, and for the test set in Fig. 7. We can see that the Handley algorithm generates far more ground truth cells during execution than are present in the final table interpretations. This is evident from Fig. 6a: the conventional recall for the final interpretations ('+') is generally lower than the historical recall. This is due to over-merging that follows the initial decision to label all *Word* regions as *Cell* regions. The Hu algorithm never discards *Cell* hypotheses, and as a result the historical and conventional recall and precision metrics are identical (Fig. 6b). The conventional metrics for the two algorithms are quite similar; the historical metrics show that the Handley algorithm generates many correct cells that the Hu algorithm does not.

Fig. 8 shows changes in *Cell* recall and precision at each decision manipulating *Cells* in our RSL implementation of the Handley algorithm. In each plot, the center represents no change, with changes in conventional/historical recall shown on the Y-axis, and changes in conventional/historical precision shown on the X-axis. Decision point Op-37 has a positive effect for most tables, as measured by all metrics. In contrast, Op-80 has little effect on historical metrics, and little or negative effect on conventional metrics. Finally, Op-68 was executed for 56 tables, but had no effect.

These new performance summaries visualize patterns in what has been generated and thrown away, for both complete strategies and individual decision points. Along with the ability to identify decision points where specific errors are caused, we believe this information will be useful for machine learning algorithms that tune decision parameters in a recognition strategy, and produce new strategies by combining decision points from multiple strategies (e.g. via genetic algorithms).

5 Conclusion

The Recognition Strategy Language is a scripting language for specifying computer vision algorithms, separating high-level decision points from low-level decision functions expressed in any programming language. This decision-based approach captures intermediate interpretations, permitting white-box evaluation, identifying decisions that cause recognition errors, and analyzing the interaction of decisions. New metrics may be observed, such as *historical recall* and *historical precision*, which measure the set of all hypotheses generated as a decision point is reached. A combination of traditional and historical metrics can then be used to better understand final results and the behavior of individual decisions.

There are some limitations to how RSL may be effectively applied. For heavily numerical/statistical methods, or for decisions that employ significant iteration, it quickly becomes impractical if one records all intermediate values and states; this can be avoided by representing only the inputs and outputs at the outermost level within RSL. Another approach would be to allow the recording of decision outcomes to be modified or disabled using keywords. To improve the speed with which the RSL observation and updating operations are implemented (see Fig. 1), we are also considering producing an interface to 'wrap' the interpretation graphs for direct use by decision functions in various languages. Visible interpretation elements and parameters would be controlled in implementations of the interface for each decision at compile/interpret-time. Finally, we plan to re-implement the RSL compiler to produce Python, which has a number of numerical libraries and interfaces to computer vision libraries (e.g. OpenCV) readily available. This will permit new language features and make RSL useful for a larger group of students and researchers.

RSL is problem-neutral, and a variety of computer vision and pattern recognition algorithms can be represented using RSL. We intend to improve support for feature computations and matrices, develop additional metrics and evaluation methods based on decision histories, and explore constructing hybrid algorithms from existing algorithms implemented in RSL [23], e.g., using genetic algorithms to combine and evolve RSL strategies.

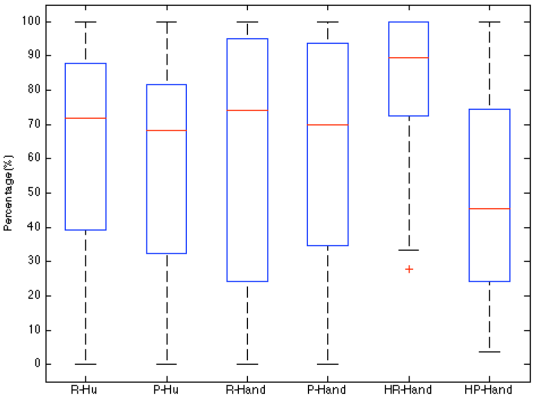


Fig. 7. Distribution of *Cell* detection metrics for Handley and Hu algorithms run on 58 tables. Shown from left to right are Hu recall and precision, Handley recall and precision, and historical recall and precision for the Handley algorithm.

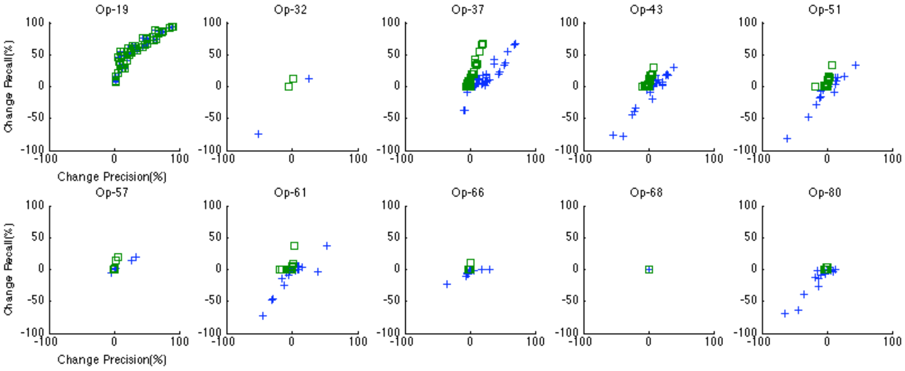


Fig. 8. Changes in precision and recall for *Cell* regions during execution of the Handley Algorithm on 58 tables. '+' show conventional recall vs. precision, and boxes show historical recall vs. precision. Op-19 is an RSL decision labeling all *Word* regions as *Cell* regions. Op-32 is only reached for tables classified as *fully-ruled*; it is executed for two tables in the test set. The following decisions are executed for the remaining 56 tables

Acknowledgments

This research was supported by the Natural Sciences and Engineering Research Council of Canada, and by Xerox Corporation through University Affairs Committee Grants held by R. Zanibbi and D. Blostein. We also thank Andrew Seniuk for help preparing the test data.

References

[1] F. Cesarini, S. Marinai, L. Sarti, and G. Soga, “Trainable Table Location in Document Images,” *Proc. Sixteenth Int'l Conf. Pattern Recognition*, Vol. 3, pp. 236–240, 2002.

[2] Competitions in Document Image Analysis Organized by ICDAR and GREC, http://www.icdar2007.org/competition.html, 2008.

[3] J. Cordy, “The TXL Source Transformation Language,” *Science of Computer Programming*, **61**(3), 190–210, 2006.

[4] D. Embley, M. Hurst, D. Lopresti, G. Nagy, “Table-processing Paradigms: A Research Survey,” *Int’l J. Document Analysis and Recognition*, Vol. 8, pp. 66–86, June 2006.

[5] J. Handley, “Table Analysis for Multi-line Cell Identification,” *Proc. Doc. Rec. and Retrieval VIII (IS&T/SPIE Electronic Imaging)*, Vol. 4307, pp. 34–43, 2001.

[6] J. Hu, R. Kashi, D. Lopresti, G. Nagy, and G. Wilfong, “Why Table Ground-truthing is Hard,” *Proc. Sixth Int'l Conf. Document Analysis and Recognition*, pp. 129–133, 2001.

[7] J. Hu, R. Kashi, D. Lopresti, G. Wilfong, “Table Structure Recognition and its Evaluation,” *Proc. Document Rec. and Retrieval VIII,* SPIE Vol. 4307, pp. 44–55, 2001.

[8] J. Hu, R. Kashi, D. Lopresti, G. Wilfong, “Evaluating the Performance of Table Processing Algorithms,” *Int'l J. Document Analysis and Recognition*, **4**(3), 140–153, 2002.

[9] M. Hurst, “Towards a Theory of Tables,” *Int’l J. Document Analysis and Recognition*, Vol. 8, pp. 123–131, June 2006.

[10] T. Kieninger A. Dengel, “Applying the T-RECS Table Recognition System to the Business Letter Domain,” *Proc. Sixth ICDAR*, pp. 518–522, 2001.

[11] J. Liang, “Document Structure Analysis and Performance Evaluation,” PhD dissertation, Univ. Washington, 1999.

[12] D. Lopresti, “Exploiting WWW Resources in Experimental Document Analysis Research,” LNCS, vol. 2423, Springer, pp. 532–543, 2002.

[13] D. Lopresti and G. Nagy, “A Tabular Survey of Automated Table Processing,” LNCS vol. 1941, Springer-Verlag, pp. 93–120, 2000.

[14] D. Lopresti and G. Wilfong, “Evaluating Document Analysis Results via Graph Probing,” Proc. Sixth Int'l Conf. Document Analysis and Recognition, pp. 116–120, 2001.

[15] Phillips, S. Chen, and R. Haralick, “CD-ROM Document Database Standard,” Proc. Second Int'l Conf. Document Analysis and Recognition, pp. 478–483, 1993.

[16] Phillips and A. Chhabra, “Empirical Performance Evaluation of Graphics Recognition Systems,” IEEE Trans. Pattern Analysis and Machine Intelligence, **21**(9) 849–870, 1999.

[17] A.E. Silva, A. Jorge, L. Torgo, “Design of an End-to-end Method to Extract Information from Tables,” Int’l J. Document Analysis and Recognition, **8**, pp.144–171, June 2006.

[18] X. Wang, “Tabular Abstraction, Editing and Formatting,” PhD dissertation, Univ. Waterloo, Canada, 1996.

[19] L. Wenyin and D. Dori, “A Protocol for Performance Evaluation of Line Detection Algorithms,” *Machine Vision and Applications*, Vol. 9, No. 5/6, pp. 240–250, 1997.

[20] R. Zanibbi, “A Language for Specifying and Comparing Table Recognition Strategies,” PhD dissertation, School of Computing, Queen's Univ., Canada, 2004.

[21] R. Zanibbi, D. Blostein, J. R. Cordy, “A Survey of Table Recognition: Models, Observations, Transformations, and Inferences,” *IJDAR***,** Vol. 7, No. 1, pp. 1–16, Sept. 2004.

[22] R. Zanibbi, D. Blostein, J. R. Cordy, “Historical Recall and Precision: Summarizing Generated Hypotheses,” *Proc. Eighth ICDAR*, vol. 2, pp. 202–206, 2005.

[23] R. Zanibbi, D. Blostein, J. R. Cordy, “Decision-Based Specification and Comparison of Table Recognition Algorithms,” *Machine Learning in Document Analysis and Recognition*, Springer, pp. 71-103, 2008.