Proc. SDIUT'95, Symposium on Document Image Understanding Technology, Bowie MD, pp. 70–83, October 1995

A Complete Environment for Ground-Truthing and Benchmarking Page Segmentation Algorithms

Luc Vincent **Berrin Yanikoglu**
Xerox Desktop Document Systems
9 Centennial Drive, Peabody MA 01960, USA
lucv@xis.xerox.com berrin@xis.xerox.com

Abstract

We describe a new approach for the automatic and objective evaluation of page segmentation (zoning) algorithms. Unlike techniques that rely on OCR output, our method is region-based: the segmentation output, described as a set of regions together with their types, ordering relationships, child-parent relationships, etc, is compared to a pre-stored set of ground-truth regions, and an assessment of segmentation quality is derived. In order for this comparison not to be fooled by slight variations in region shapes, it is achieved using region maps in which ON-pixels are painted according to the region(s) they belong to. From the "ground-truth map" and the "segmentation map", error maps are derived, in which segmentation mistakes and their locations are encoded. Error maps enable an accurate assessment of segmentation quality regardless of the number and intricacy of the regions on the page. Misclassifications, splittings and merging of various types, are among the zoning mistakes that are derived from these error maps.

Our segmentation benchmarking system can be finely tailored to a user's needs: region types, subtypes, attributes, etc, can be customized, the weights of errors of different types are user-defined, and several method are provided for combining errors into a global quality measure. This approach is also of great interest for benchmarking specific aspects of segmentation, such as headline detection, text/image separation, reverse-video text detection, etc.

The ground-truth data files needed by the system can be easily generated and updated using GroundsKeeper, *an X-window based tool we developed.* GroundsKeeper *allows one to view a document image, manually draw regions (rectangular or arbitrarily shaped) on top of it, and specify information about each region (type, attributes, etc). Just like the benchmarking system itself,* GroundsKeeper *is completely customizable: region types, sub-types, attributes, etc, are specified by the user in a startup file.*

1 Benchmarking Page Segmentation: Introduction

Page segmentation is the process by which a document page image is decomposed into its structural and logical units, such as images, rulings, paragraphs, headlines, tables, line-art regions, etc. This process, often refered to also as "zoning", "layout analysis", or "page decomposition", is critical for a variety of document image analysis applications. First and most importantly, commercial OCR packages need page segmentation to understand the layout of the page, read the text in the correct order (without attempting to read the halftones and graphics), and ultimately be able to *recompose* the document in a word processing environment. Similarly, reading machines for the blind need accurate page segmentation to be able to deal with multicolumn documents and read the text in an adequate order. Even digital copiers and related software/hardware systems rely on document segmentation techniques to correctly identify the various types of regions present on a page (e.g., text, continous tone images, halftones, line-art, ...), and render them appropriately [8].

A very large number of page segmentation algorithms have been developed in the past couple of decades, some described in literature [9] and some prorietary [20]. These algorithms use many different aproaches to the problem, some of which include: rule-based systems [4], use of connected component bounding-boxes [5], use of background and "white streams" information [12, 2, 1], etc. Some of these algorithms are specifically designed to work on particular types of document, whereas others are meant to be completely general. Furthermore, some algo-

rithms are designed for very specific sub-tasks of what is generally refered to as page segmentation: for example, some algorithms may simply detect the graphics in a document page image, others may only care about the text. Some algorithms generate coarse segmentations (e.g., at the galley level) while others decompose pages down to the paragraph level, or even down to the line level. For some applications, a page segmentation algorithm may need to distinguish between halftones and continuous tones, for others, the distinction is irrelevant.

Faced with such a diversity of methods and such a wide variety of goals, a question arises: how good is a particular segmentation algorithms at performing a particular type of page segmentation? How should we assess the quality of a given algorithm for a given task? The accuracy of a page segmentation system, is unfortunately very difficult to evaluate objectively. Typically, assessing the quality of a system involves running it on a large number of document images and "eyeballing" the results, a very tedious and subjective process.

For all these reasons, systems that make it easier to automatically and objectively evaluate the adequacy of a zoning algorithm for a given segmentation task are becoming more and more important. Obviously, a good segmentation benchmarking tool, would be very helpful in deciding between commercial systems for a given application. But beyond that, such a benchmarking environment would also enable developers of page segmentation systems to regularly test their algorithms against large databases of documents, thereby speeding up the process of testing new ideas and fine-tuning existing segmentation algorithms.

Two main approaches have been proposed in literature for benchmarking page segmentation. The first one, developed at UNLV, is purely text-based [17, 6, 7], whereas the approach we have chosen is region-based instead [15, 21]. These two approaches are briefly described below, and we explain why we believe our region-based approach to offer significant advantages over the UNLV system.

The rest of the paper is organized as follows: in Section 2, we describe our ground-truthing methodology, the tool we developed to easily create segmentation ground-truth files, as well as the file format used to represent the greound-truth information. In section 3, we describe the benchmarking itself: the use of region maps by the system is first described, then we deal with error identification and combination. Finally, we show how the system can be customized to virtually any user's need, and conclude.

1.1 Text-Based Approaches to the Evaluation of Page Segmentation Algorithms

The Information Science Research Institute (ISRI) of the University of Las Vegas (UNLV) has designed an interesting technique to evaluate the quality of page segmentation algorithms working within OCR software packages: schematically, this method works as follows:

1. Run page segmentation (including region ordering) *and* character recognition on a document page, and output the result as an ASCII string;

2. Calling c_i the cost of an elementary insertion operation in a string, and c_m the cost of a block move, use string matching algorithms derived from [19] to determine the number of insertions n_i and the number of block moves n_m that are needed to turn this output string into the ideal *ground-truth* string while minimizing the associated page cost $C = n_i \times c_i + n_m \times c_m$;

3. From this number, *subtract* the cost of errors that are purely due to OCR mistakes, as opposed to segmentation mistakes; this is done by running the OCR system on the same page, using *manual zoning* information, and subtracting the resulting error cost (expressed in numbers of insertions times cost c_i of an insertion) from the page cost C.

The big advantage of this technique is that it is purely text-based, and therefore does not require the page segmentation sub-system to specifically output zoning results in any particular format. In addition, although its underlying string-matching algorithms are rather elaborate, the overall approach is fairly straightforward: for one thing, ground-truth files for use with this system are very easy to create. Therefore, the UNLV zoning evaluation system has been relatively well accepted by the document recognition community.

Nonetheless, this system has a few severe limitations:

- Although this may not be true any longer, the UNLV segmentation quality metric only takes *insertions* and *block moves* into account, *deletions* are assumed to have zero cost. In practice, this means that an algorithm will not be penalized for finding extra text regions where there are none, or worse, for detecting images as text.

- With this method, only one zone order, or a small set of zone orders, are considered acceptable. If a zoning algorithm produces a perfect segmentation, but does not output regions in one of these acceptable orders, it will be penalized. This is sometimes unfair, since in many cases, region order is not uniquely defined (see Section 2.1). For example, how should such text regions as captions, headers, footers, or insets, be ordered? Clearly a range of options should be considered equally correct.

- Similarly, this approach requires running the OCR system twice: once with automatic segmentation and once with manual zoning, so that errors purely due to the OCR can be subtracted. The underlying assumption here is that OCR accuracy is unchanged regardless of scanning order. This is often not true in practice: a messy segmentation generally results in poorer OCR quality (for example, if the segmentation splits or joins two galleys, hyphens are incorrectly paired, so the OCR engine can rely on fewer words to be lexically verified...). Therefore, this benchmarking approach tends to give lower segmentation scores than it would if the OCR engine was perfect.

- The method requires the zoning algorithm being benchmarked to be part of of an OCR system. Without OCR, there is no way to benchmark segmentation! Specifically, this means that the segmentation of documents in "exotic" languages (not supported by the OCR engine) cannot be benchmarked. More annoying, the segmentation of document images containing no text cannot be evaluated with this system.

- Along the same lines, the UNLV segmentation benchmarking system can only deal with text regions: the accuracy measure produced only reflect accuracy on text zones!

- Lastly, the output of the UNLV system is merely a set of numbers (number of insertions, block moves, and perhaps deletions). It therefore provides very little information on the types of mistakes that were actually made (Were regions split or merged? Were images mistaken for text? Were headline regions messed up more than regular text regions? etc). Such information would be very valuable to the segmentation algorithm developer.

1.2 Our Region-Based Approach to Automatic Segmentation Benchmarking

For all the reasons listed above, we started working on a region-based approach to page segmentation benchmarking. As will be shown in the rest of the paper, the system we developed is more complex and cumbersome to use than a text-based system, but offers much more flexibility, and avoids most of the shortcomings of the UNLV approach.

In our system, the output of the segmentation algorithm to be benchmarked is modeled as a set of regions, a region being a polygon (representing the outline of a page zone) together with such attributes as a type (text, image, etc) and a parent zone. This set of segmentation regions is matched against a pre-stored set of *ground-truth regions,* and such problems as erroneous region merging and splitting, misidentified region types, or extra "noise" regions are flagged.

A tool, *GroundsKeeper,* was developed to help the creation of ground-truth files. This tool, described in Section 2.2, allows the user to bring up a document image, draw a set of regions (enclosing image features such as halftones, paragraphs, captions, etc), specify ordering and "attachement" relationships between regions, and save the result in a simple ASCII format called *RDIFF (Region Description Information File Format).* This format is described in Section. 2.1. As will be emphasized later, *GroundsKeeper* can be customized to a user's needs.

During the actual benchmarking, the set of ground-truth region represented in the RDIFF file is matched against a set of *segmentation regions.* This region matching step is actually performed using *sets of pixels* instead of polygonal representations. In other words, each region is modeled as the set of ON-pixels its associated polygon contains: this way, the shape of region polygons is not taken into account, only their ON-pixel contents matters. Errors can therefore be accurately flagged, and insignificant differences between ground-truth and segmentation polygons are automatically ignored. This overall scheme is efficiently implemented through the use of *region maps* and *error maps,* as described in Section 3.

As we stress later, both the ground-truthing tool (GroundsKeeper) and the benchmarking tool (currently called *cluzo*) are completely customizable. At ground-truthing time, region types, subtypes, and attributes are user-defined. Additionally, in the benchmarking, some region types can be ignored

while some can be made equivalent to other (example: ignore all `halftone` regions, and make `caption` type equivalent to `regular-text`), the types of errors that are important can be specified by the user, together with the error weighting method. All this information is stored in `init` files used by the ground-truthing and benchmarking tools.

2 Creation of Ground-Truth Files

2.1 Representation of Ground-Truth Information

To represent ground-truth information as well as segmentation results, the RDIFF format is used. For simplicity, this is an ASCII format, containing some header information such as the name of the associated document image, the resolution, the page size, and then a list of regions. Each region has an associated type and sub-type, may refer to a "parent" region, and may have any number of attributes. Region shape itself is encoded as a simple polygon (list of coordinates).

Providing the complete specifications of the RDIFF format would be beyond the scope of this paper. Let us however point out a few interesting features of this format:

- RDIFF is a Tag-Value based format, meaning that each entry in the format starts with a tag, the rest of the entry being the value of this tag. The advantage of this approach is its flexibility: an RDIFF reader typically only knows about the tags that are relevant to its purpose, and ignores the rest. In addition, even as new tags are added with each new release of the format, the new RDIFF files remain backwards compatible with previouly written RDIFF readers.
- Region shape is currently encoded using a polygon. In most applications, this is sufficient. However, if it became important to support region with holes, or disconnected regions, the RDIFF format could easily be expanded. New region representations, using for example lists of polygons, or run-length encoding, could be added, and a new tag would be used to specify the representation scheme chosen for each region (the default scheme being the current one: single polygon).
- Note that there are no restrictions on region location: regions can overlap, be included into one another, etc.

An example of RDIFF file is given below. Fig. 3 shows the *GroundsKeeper* window with the corresponding set of regions overlayed on the document image they were created from.

```
BEGIN_IMAGE
FILENAME /data/tif/h/halftone21.tif
IMAGE_WIDTH 2560
IMAGE_HEIGHT 3300
IMAGE_XRES 300
IMAGE_YRES 300
END_IMAGE

BEGIN_SUMMARY
INIT_FILE_VERSION 1.8 1995/04/18
TOTAL_REGIONS 25
TEXT_REGIONS 25
IMAGE_REGIONS 0
V_RULE_REGIONS 0
H_RULE_REGIONS 0
ORDER_FLAGS ANY
END_SUMMARY

BEGIN_REGIONS

R_TYPE IMAGE
R_SUBTYPE HALFTONE
R_NUMBER 1
R_ATTACHMENT 0
R_PARENT 0
R_DRAW_TYPE Box
R_NAME Zone1
R_ATTRIBUTES SHADED_BACKGROUND
REGION_POLYGON 301 692 71 681

... ...

R_TYPE TEXT
R_SUBTYPE CAPTION
R_NUMBER 9
R_ATTACHMENT 1
R_PARENT 0
R_DRAW_TYPE Box
R_NAME Zone9
REGION_POLYGON 722 838 78 266

R_TYPE TEXT
R_SUBTYPE CAPTION
R_NUMBER 10
R_ATTACHMENT 2
R_PARENT 0
R_DRAW_TYPE Box
R_NAME Zone10
REGION_POLYGON 1510 1584 82 348

... ...

R_TYPE IMAGE
R_SUBTYPE HRULE
R_NUMBER 15
R_ATTACHMENT 0
R_PARENT 0
R_DRAW_TYPE Box
R_NAME Zone15
REGION_POLYGON 136 150 0 1920

R_TYPE IMAGE
```

```
R_SUBTYPE VRULE
R_NUMBER 16
R_ATTACHMENT 0
R_PARENT 0
R_DRAW_TYPE Box
R_NAME Zone16
REGION_POLYGON 306 3034 1142 1156

...

R_TYPE TEXT
R_SUBTYPE REG_TEXT
R_NUMBER 18
R_ATTACHMENT 0
R_PARENT 0
R_DRAW_TYPE Box
R_NAME Zone18
REGION_POLYGON 304 426 722 1110

R_TYPE TEXT
R_SUBTYPE REG_TEXT
R_NUMBER 19
R_ATTACHMENT 0
R_PARENT 0
R_DRAW_TYPE Constrained-Polygon
R_NAME Zone19
REGION_POLYGON 428 1086 376 1116
COUNT 14
774 428
982 428
982 464
1116 464
1116 1048
954 1048
954 1086
718 1086
718 1048
376 1048
376 714
722 714
722 466
774 466

... ...

R_ORDER_RULE 18 < 19 < 20 < 21 < 22 < 23 < 24 < 25
```

Note the R_ORDER_RULE at the end of this file. This represents a set of ordering constraints between regions. Indeed, as mentioned in Section 1.1, we firmly believe that one complete region order is inadequate. Instead, a number of ordering constraints (or rules) should be used. For example, ordering constrainges for regular text regions should be separate from ordering constrains among insets or side-bars. The latter regions are indeed not part of the main flow of text. Similarly, having several ordering constraints makes it possible to deal with ambigous cases such as the one shown in Fig. 1.

Accordingly, in the RDIFF format, any number of partial ordering rules can be specified using the R_ORDER_RULE tag. An example where several of these tags would be needed is shown in Fig. 2. Note that ordering constraints are not always the correct solution. Sometimes, *attachements* should be used instead. For example, a caption is "attached" to a picture, it does not come before or after this picture. Attachements can also be specified in the RDIFF language.

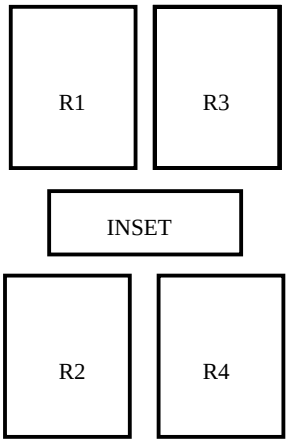

Figure 1: Ambiguous case for region ordering. The correct order between text regions R_1 through R_4 could be R_1, R_2, R_3, R_4 or R_1, R_3, R_2, R_4.

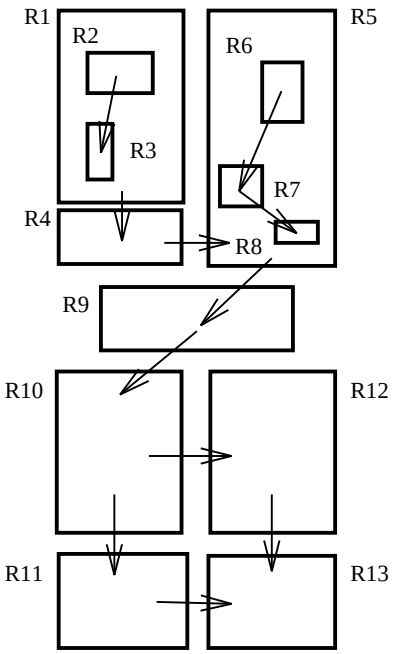

Figure 2: The order between regions in this "page" can be specified using four partial orders: $R_1 < R_4 < R_5 < R_9 < R_{10} < R_{11} < R_{13}$, $R_2 < R_3$, $R_6 < R_7 < R_8$, and $R_{10} < R_{12} < R_{13}$.

To conclude this section on RDIFF, let us point

out that region ordering relationships are often considered as not being part of page segmentation itself. In fact, the `R_ORDER_RULE` tags are currently used only by a very small subpart of our benchmarking system. We made them part of our ground-truth format mainly in an effort to be complete.

2.2 X-Window Based Tool to Assist Ground-Truth Creation: GroundsKeeper

To easily create test suites of segmentation ground-truth files in the RDIFF format, we developed an X-Window based tool called *GroundsKeeper* (see Fig. 3). For convenience, this tool provides three different region drawing methods: rectangles, that are sufficient for most cases, constrained polygons, i.e., polygons whose edges are bound to be vertical or horizontal, and arbitrary polygons, that are useful for complex layouts, or skewed pages. After their creation, regions can be deleted, moved, or modified at will. Previously created files can also be read in and updated. "Sequences" (i.e. region ordering rules) can be specified by simply clicking on regions one after the other. The functionality described in this paragraph is available under the "Zone" menu shown in Fig. 4.

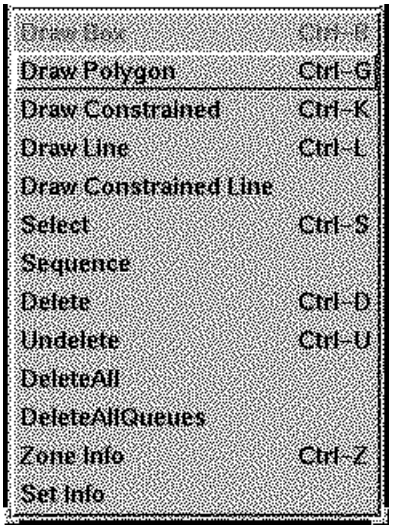

Figure 4: *GroundsKeeper*'s "Zone" menu.

Different display modes are also available. By default, the image is shown, with the regions and the sequences drawn over it, but one can also choose to display only the regions, or only the image (see "Display" menu in Fig. 5. Sequences, normally shown as a succession of arrows, can also be ommitted from the display if desired. More importantly, different zoom levels are available. In fact, a good way to use *Groundskeeper* is to first draw the regions coarsely at low-resolution, then zoom-in and adjust the polygons created.

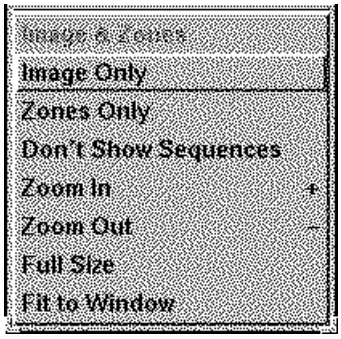

Figure 5: *GroundsKeeper*'s "Display" menu.

The polygons created are given unique identifying numbers, and can be tagged with their type, their sub-type, plus any number of *attributes.* Types, subtypes, and attributes are defined in an `init` file that *GroundsKeeper* reads at startup: they are therefore completely customizable for any particular application. In house, we use the `init` file shown in Fig. 6. Note that we chose to create very detailed ground-truth files: indeed, one can always choose to use less information that these files contain, but if the files were not detailed enough for a given application, they would be much less useful.

Groundskeeper has a number of additional features designed to ease and speed-up the creation of RDIFF files. Depending on the complexity of the input document, it takes between 2 and 30 minutes to create a complete detailed ground-truth file. We have created half a dozen test suites of ground-truthed documents (magazines, newspapers, business letters, etc), for a total of about 500 RDIFF files. Currently, these suites are primarily used for testing and improving the segmentation benchmarking algorithms in *cluzo,* but progressively, we will use them more and more for benchmarking in-house and external segmentation algorithms.

3 Segmentation Benchmarking Algorithms

3.1 Use of Region Maps

One of the key features in our region-based approach to benchmarking segmentation is that regions are *not* compared based on their shape, but based on their ON-pixel contents. Two regions that enclose the same image features (text, image, etc) should indeed be considered as identical regardless of how

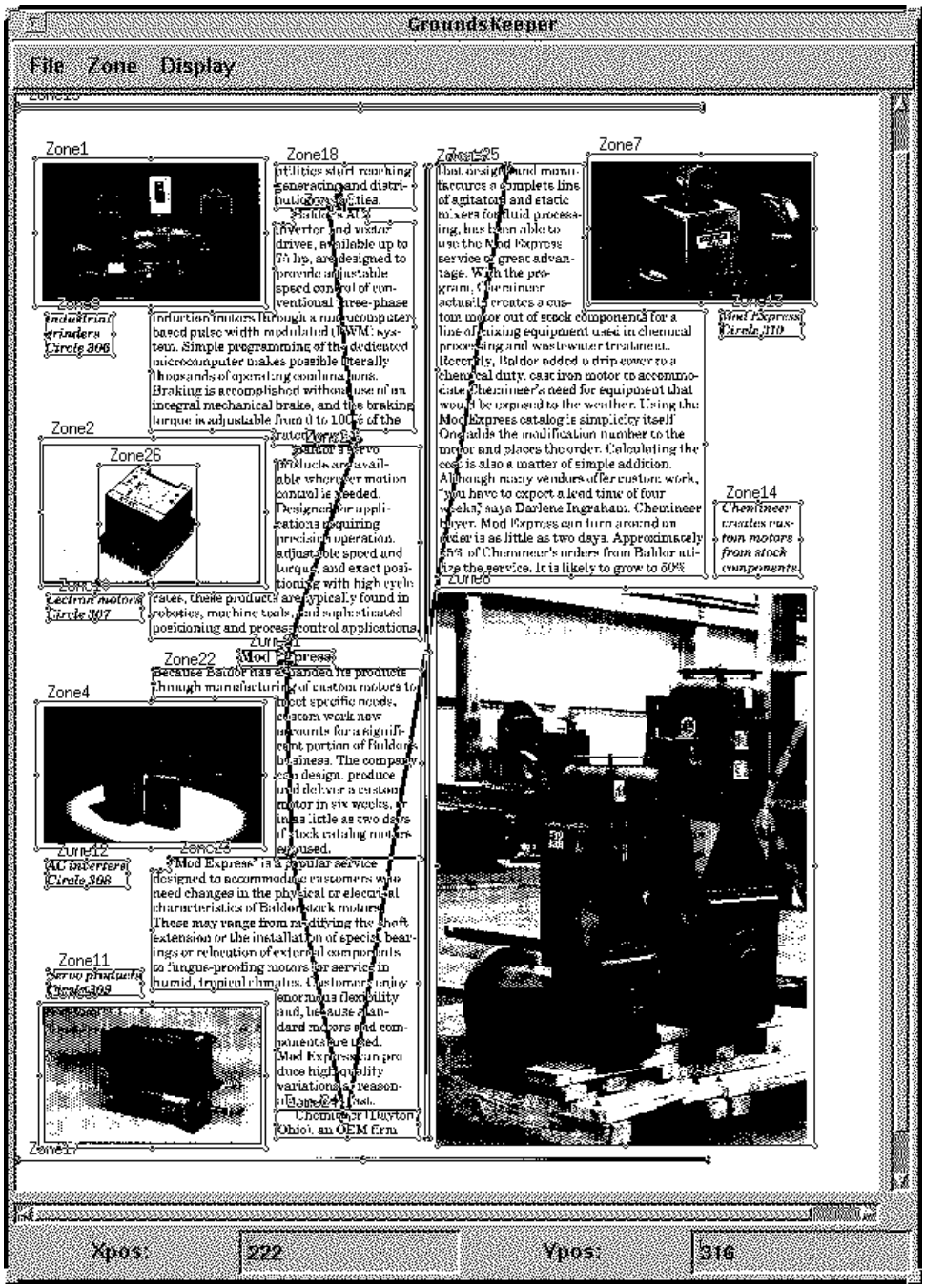

Figure 3: *GroundsKeeper* window showing document image `halftone21.tif` together with the complete set of ground-truth regions created. Ordering constraints between regions are shown as arrows (currently selected).

```
ATTRIBUTE_NAME  ANGLED
ATTRIBUTE_NAME  BAR-CHART
ATTRIBUTE_NAME  BULLETS
ATTRIBUTE_NAME  CELL-TABLE
ATTRIBUTE_NAME  CURVED-TEXT
ATTRIBUTE_NAME  HAND-DRAWN
ATTRIBUTE_NAME  ILLEGIBLE
ATTRIBUTE_NAME INCOMPLETE
ATTRIBUTE_NAME  OUTLINED
ATTRIBUTE_NAME  PIE-CHART
ATTRIBUTE_NAME  REVERSE_VIDEO
ATTRIBUTE_NAME  SHADED_BACKGROUND
ATTRIBUTE_NAME  TAB-TABLE
ATTRIBUTE_NAME  UNDERLINED
ATTRIBUTE_NAME  UPSIDE-DOWN
ATTRIBUTE_NAME  VERTICAL

IMAGE_TYPE_NAME CAPTION
IMAGE_TYPE_NAME CAPTION_or_HEADLINE
IMAGE_TYPE_NAME CAPTION_or_REG_TEXT
IMAGE_TYPE_NAME CELL
IMAGE_TYPE_NAME DROPCAP
IMAGE_TYPE_NAME FOOTER
IMAGE_TYPE_NAME FOOTNOTE
IMAGE_TYPE_NAME FRAMEBOX
IMAGE_TYPE_NAME GRAPHICS
IMAGE_TYPE_NAME GRID
IMAGE_TYPE_NAME HALFTONE
IMAGE_TYPE_NAME HEADER
IMAGE_TYPE_NAME HEADER_or_HEADLINE
IMAGE_TYPE_NAME HEADLINE
IMAGE_TYPE_NAME HEADLINE_or_REG_TEXT
IMAGE_TYPE_NAME HRULE
IMAGE_TYPE_NAME INSET
IMAGE_TYPE_NAME LOGO
IMAGE_TYPE_NAME RAISEDCAP
IMAGE_TYPE_NAME REG_TEXT
IMAGE_TYPE_NAME SIDEBAR
IMAGE_TYPE_NAME SUBTITLE
IMAGE_TYPE_NAME TABLE
IMAGE_TYPE_NAME TABLE_COLUMN
IMAGE_TYPE_NAME TABLE_LINE
IMAGE_TYPE_NAME TIMESTAMP
IMAGE_TYPE_NAME UNKNOWN_TYPE
IMAGE_TYPE_NAME VRULE
```

Figure 6: Example of *GroundsKeeper* init file.

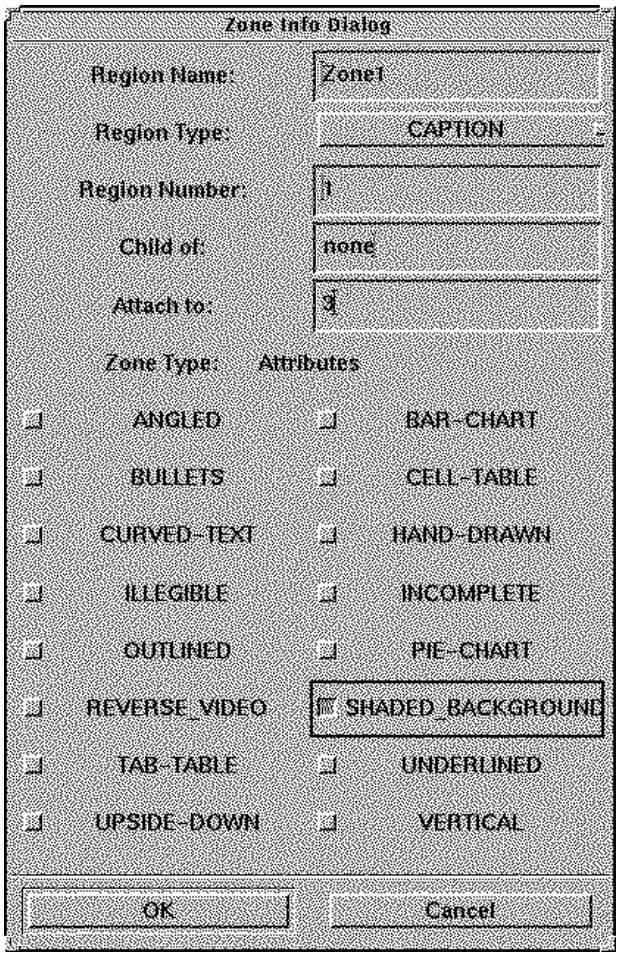

Figure 7: *GroundsKeeper*'s "Zone Info" dialog box. The attributes shown correspond to the list present in the `init` file of Fig. 6.

tightly they enclose these features.

In order to efficiently deal with regions as sets of ON-pixels, the first step in our system is to create *region maps.* A region map is created for the ground-truth regions, and another one for the "segmentation regions". These two maps are called ground-truth map and segmentation map respectively. Each one is a low-resolution representation of the original document image, in which each ON-pixel is tagged according to the region(s) it belongs to.

3.2 Region Matching and Identification of Errors

The region maps are used for identifying the following types of segmentation errors:

- missed region
- extra region
- erroneous region type
- vertical split
- horizontal split
- vertical merge
- horizontal merge

To this end, both maps are scanned concurrently in order to detect which segmentation regions intersect with each ground-truth region, and which ground-truth regions intersect with each segmentation region.

One-to-one matches are first considered: if the region types involved are identical (or equivalent for the purpose of the segmentation being benchmarked), then they correspond to zones that have been correctly segmented. Otherwise, mislabeling errors are identified.

All other cases correspond to segmentation errors. For example, if a ground-truth region does not match with any segmentation region (i.e. does not intersect with any segmentation region), it means that the corresponding page object has been missed by the segmentation algorithm.

Take now the case of a segmentation region S that matches with several ground-truth regions G_1, G_2,..., G_n. This means that regions have been erroneously merged. In this case, depending on region types, and depending on the type of error analysis desired, it may be useful to detect whether the merge was *vertical* or *horizontal.* For example, if the segmentation is to be used in the context of an OCR system, vertical merging of text regions is usually not a big deal, especially if the regions are consecutive within the same ordering rule. On the other hand, horizontal merging of text regions should be strongly penalized. To detect whether a merging is horizontal, we scan the pixels of the segmentation region S and see for each scanline if the corresponding ON-pixels in the ground-truth map belong to several regions. When this is the case, the entire scanline corresponds to an erroneous merge and is flagged as such. With this method, we are able to only flag those pixels that are actually affected by the horizontal merge. This is illustrated by Fig. 8. The same analysis can be done for vertical merging as well if needed.

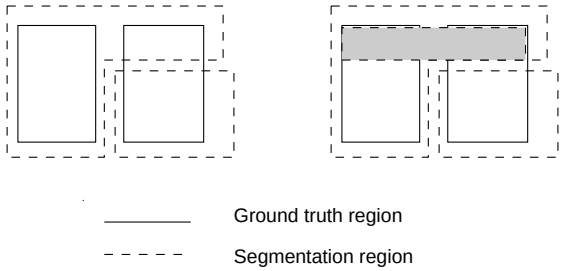

Figure 8: When benchmarking the segmentation of an OCR system, only the pixels in the shaded area will be found to be part of an erroneous horizontal merge of text regions.

Similarly, when a ground-truth region is found to match several segmentation regions, this is a sure sign that this region has been erroneously split by segmentation. The same kind of analysis as described in the previous paragraph can be done in order to decide whether the split was vertical or horizontal. For more details on how all the different cases are handled, refer to [21].

3.3 Weighting and Combination of Errors

When a segmentation error is detected, how should it be weighted? How should the seriousness of this error be assessed? How should errors be combined if they affect the same ground-truth region? For example, if (part of) a ground-truth region is found to have been split twice by segmentation, should this be considered twice as bad? Similarly, if this region is found to have been both merged and split, should one of these error types have precedence over the other one, or should both error be charged?... The answers to all those questions depend on the type of segmentation being benchmarked, and should be specified in the `init` file used by our benchmarking tool (see next section).

To deal with all the possible ways one may want to weight and combine errors, we use a scheme based on *error maps.* Each time an error is detected, the ON-pixels involved are flagged in this error map. When-

ever a pixel is found to be involved in several errors (for example, an erroneous split and an erroneous merge), we can decide to flag this pixel as being involved in all these errors, or only in the most serious one (the seriousness of each error type is specified in the `init` file).

Eventually, once all the errors have been detected, the resulting error map is analyzed and the segmentation quality is then characterized by a few numbers, such as: percentage of mislabeled pixels, percentage of pixels involved in horizontal merges, etc. The set of numbers used to characterize segmentation accuracy is derived from the `init` file used by the system.

As an example, consider the set of ground-truth regions shown in Fig. 9: 8 text regions and 2 image regions are present. The double-spaced text was zoned into 5 different paragraphs, according to our ground-truthing guidelines, but for most applications, zoning it into 2 galleys would have been sufficient. The automatic segmentation algorithm we used here did not see the two different galleys present, and made a mess of this page, as shown in Fig. 10. The corresponding error map created by our benchmarking algorithm is shown in Fig. 11. Note that in this map, only the pixels where horizontal merging occured are flagged (black pixels in the map). Indeed, the `init` file we used specified that vertical merge/split of text should not be penalized.

The output of the benchmarking tool for this example is given below. Note that for "historical" reasons we are still distinguishing between labelling errors and all other error types. The `init` file used to produce this output specified 6 different types of mislabelling, but only one type of vertical merge, horizontal merge, etc. Also recall that the cost of vertical merges and splits was set to 0.

```
Groundtruth file:          bhng11.sgt
Segmentation file:         bhng11.rdiff
Image file:                bhng11.tif

ZONING:

Missed regions              =     0.00 %
Noise regions               =     0.00 %
Horizontal merges           =    49.22 %
Horizontal splits           =     0.00 %
Vertical merges             =     0.00 %
Vertical splits             =     0.00 %

Overall zoning quality      =    50.78 %

LABELING:

Text recognized as image    =     0.00 %
Img. recognized as text     =     0.45 %
Text recognized as noise    =     0.00 %
Img. recognized as noise    =     0.00 %
Noise recognized as text    =     0.00 %
Noise recognized as image   =     0.00 %

Overall labeling quality    =    99.55 %
```

3.4 Customizing the Benchmarking

Just like *GroundsKeeper,* our benchmarking tool *cluzo* can be customized to a particular application. The `init` file required by the program is decomposed into several sections:

- Listing of all the region types and attributes that can be encountered in RDIFF files
- Description of region equivalences: in this section, a user can specify that type A is in fact equivalent to type B, and that type D is equivalent to type B as well, etc. For example, if the user did not care about distinguishing between different types of image regions (such as graphics, halftones, line-art, etc) and text regions (such as captions, headlines, footers, etc), the total number of region types could be reduced to 2.
- Listing of all the different error types considered, together with their weights. This can be done for generic regions as well as for particular region types. For example, the `init` file we currently use contains the following:

  ```
  Weight_of Vertically_split  REGION   0
  Weight_of Vertically_split  GRAPHICS 3
  Weight_of Vertically_split  HALFTONE 3
  ```

  This specifies that a vertical split is of cost 0, except when it involves a GRAPHICS region or a HALFTONE region, in which case the associated cost becomes 3.
- Error weighting method to use. By default, each error is weighted by its intrinsic weight (as defined in the previous section) multiplied by the number of pixels involved. In some cases however, one may want to be able to specify that an error of a given type is of nominal cost, regardless of the number of pixels involved. Similarly, when text regions are involved, it may make sense to use the *height* ($\approx$ proportional to

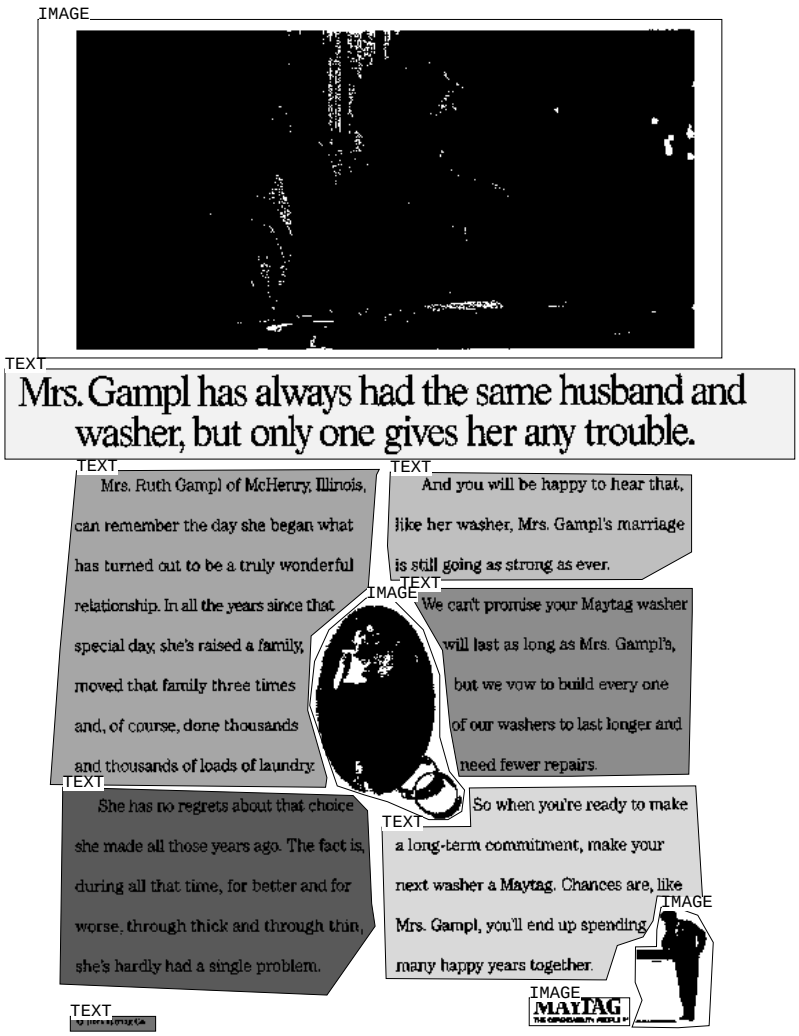

Figure 9: Display of the ground-truth regions for a document image.

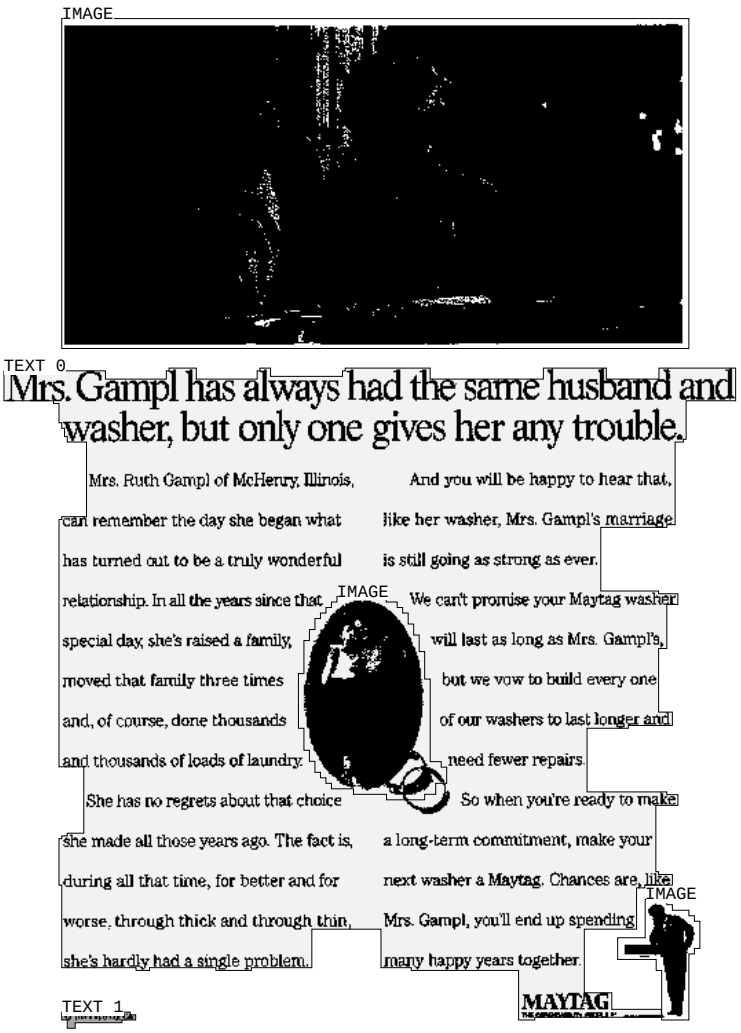

Figure 10: A segmentation result for the same image as in Fig. 9. Note the horizontal merging of text galleys.

Figure 11: Error map corresponding to the segmentation result displayed in Fig. 10. Pixels involved in a merging error are black in this map.

the number of text lines) of the identified error region as weight. We are currently working on techniques to enable all these different weighting method to coexist harmonioulsy.

- Definition of the error combination method: when an area of the page is involved in multiple segmentation mistakes, should all these mistakes been taken into account, or should the most serious mistake (the one with highest associated weight) prevail?

The format of this `init` file is still evolving, as we are working on better and simpler ways to describe a user's benchmarking needs in this file.

4 Conclusions, Future Work

We described the region-based page segmentation benchmarking environment currently under development at Xerox. Every aspect of the system has been developed with flexibility in mind, and we always tried to think of all the different ways this system could be used in practice. Instead of computing a single number/curve characterizing segmentation accuracy, our system is able to keep track of an arbitrarily large number of error types, and therefore can provide very detailed information on segmentation quality. For these reasons, and because the system does not require any OCR, we believe that it is a compelling alternative to text-based segmentation benchmarking approaches.

Most of the functionality has been implemented, and we are currently in an incremental improvement phase. In particular, we are working on better and simpler ways to specify error types and weights in the system. We are also considering a few options for extending our benchmarking system to deal with grayscale images: this would potentially open a whole new range of new applications for this system, beyond document analysis.

The actual error numbers reported are still a bit arbitrary at times, and this makes it still difficult to use our system to compare very different segmentation algorithms. However, as is, the system is already invaluable for segmentation algorithm developers: indeed, absolute error figures are less useful to developers than relative numbers. Using *cluzo* on a large enough test suite makes it easy to automatically flag degradations and improvements in different aspects of segmentation, and to fine-tune the parameters of a given zoning algorithm.

References

[1] A. Antonacopoulos and R. Ritchings. Flexible page segmentation using the background. In *12th Int. Conf. on Pattern Recognition*, pages 339–344, Jerusalem, Oct. 1994.

[2] H. S. Baird. Background structure in document images. In *Advances in Structural and Syntactic Pattern Recognition*, volume 5, pages 253–269. World Scientific, 1992.

[3] O. Desforges and D. Barba. Segmentation of complex documents multilevel images: a robust and fast text bodies-headers detection and extraction scheme. In *Int. Conf. on Document Analysis and Recognition.*, pages 770–773, Montreal, 1995.

[4] J. Fisher, S. Hinds, and D. D'Amato. A rule-based system for document image segmentation. In *Int. Conf. on Pattern Recognition*, pages 567–572, Atlantic City, NJ, June 1990.

[5] J. Ha, R. Haralick, and I. Phillips. Document page decomposition by the bounding-box projection technique. In *Int. Conf. on Document Analysis and Recognition.*, pages 1119–1122, Montreal, 1995.

[6] J. Kanai, T. A. Nartker, S. V. Rice, and G. Nagy. Performance metrics for document understanding systems. In *Int. Conf. on Document Analysis and Recognition.*, pages 424–427, Tsukuba, Japan, Oct. 1993. IEEE Computer Society Press.

[7] J. Kanai, S. V. Rice, T. Nartker, and G. Nagy. Automatic evaluation of OCR zoning. *IEEE Trans. Pattern Anal. Machine Intell.*, 17(1):86–90, Jan. 1995.

[8] Y.-W. Lin. Digital image processing in the Xerox docutech document processing system. In L. Vincent and T. Pavlidis, editors, *SPIE/SPSE Vol. 2181, Document Recognition*, San Jose CA, Feb. 1994.

[9] M. Nadler. A survey of document segmentation and coding techniques. *Comp. Vis., Graphics and Image Processing*, 28:240–262, 1984.

[10] G. Nagy and S. Seth. A prototype document analysis system for technical journals. *IEEE Computer*, 25(7):10–22, July 1992.

[11] L. O'Gorman. The document spectrum for bottom-up layout analysis. *IEEE Trans. Pattern Anal. Machine Intell.*, 15(10):1162–1173, Oct. 1993.

[12] T. Pavlidis and J. Zhou. Page segmentation by white streams. In *Int. Conf. on Document Analysis and Recognition.*, pages 945–953, Saint-Malo, France, 1991.

[13] T. Pavlidis and J. Zhou. Page segmentation and classification. *Comp. Vis., Graph. Im. Proc.: Graphical Models and Image Processing*, 54(6):484–496, Nov. 1992.

[14] S. Randriamasy and L. Vincent. Benchmarking page segmentation algorithms. In *IEEE Int. Computer Vision and Pattern Recog. Conference*, Seattle, WA, June 1994.

[15] S. Randriamasy, L. Vincent, and B. Wittner. An automatic benchmarking scheme for page segmentation. In L. Vincent and T. Pavlidis, editors, *SPIE/SPSE Vol. 2181, Document Recognition*, San Jose CA, Feb. 1994.

[16] S. V. Rice, F. Jenkins, and T. A. Nartker. The fourth annual test of OCR accuracy. Technical report, ISRI, 1995.

[17] S. V. Rice, J. Kanai, and T. A. Nartker. A preliminary evaluation of automatic zoning. Technical report, ISRI, 1993.

[18] S. V. Rice, J. Kanai, and T. A. Nartker. The third annual test of OCR accuracy. Technical report, ISRI, 1994.

[19] E. Ukkonen. Algorithms for approximate string matching. *Information and Control*, 64:100–118, 1985.

[20] L. Vincent. Page segmentation in Textbridge. Unpublished Document, Xerox Desktop Document Systems, 1993.

[21] B. Yanikoglu and L. Vincent. Ground-truthing and benchmarking document page segmentation. In *Int. Conf. on Document Analysis and Recognition.*, pages 610–604, Montreal, Aug. 1995.