LAMP-TR-58
CAR-TR-955
CS-TR-4190

September 2000

Software Architecture of PSET: A Page Segmentation Evaluation Toolkit

Song Mao and Tapas Kanungo

Software Architecture of PSET: A Page Segmentation Evaluation Toolkit

Song Mao and Tapas Kanungo

Language and Media Processing Laboratory
Center for Automation Research
University of Maryland, College Park, MD

Abstract

Empirical performance evaluation of page segmentation algorithms has become increasingly important due to the numerous algorithms that are being proposed each year. In order to choose between these algorithms for a specific domain it is important to empirically evaluate their performance. To accomplish this task the document image analysis community needs i) standardized document image datasets with groundtruth, ii) evaluation metrics that are agreed upon by researchers, and iii) freely available software for evaluating new algorithms and replicating other researchers' results.

In an earlier paper (SPIE Document Recognition and Retrieval 2000) we published evaluation results for various popular page segmentation algorithms using the University of Washington dataset. In this paper we describe the software architecture of the PSET evaluation package, which was used to evaluate the segmentation algorithms. The description of the architecture will allow researchers to understand the software better, replicate our results, evaluate new algorithms, experiment with new metrics and datasets, etc. The software is written using the C language on the SUN/UNIX platform and is being made available to researchers at no cost.

This research was funded in part by the Department of Defense under Contract MDA 9049-6C-1250, Lockheed Martin under Contract 9802167270, the Defense Advanced Research Projects Agency under Contract N660010028910, and the National Science Foundation under Contract IIS9987944.

1 Introduction

It is important to quantitatively monitor progress in any scientific field. The information retrieval community and the speech recognition community, for example, have yearly competitions in which researchers evaluate their latest algorithms on clearly defined tasks, datasets, and metrics. To make such evaluations possible, researchers have access to standardized datasets, metrics, and freely available software for scoring the results produced by algorithms [18, 1].

In the Document Image Analysis area, regular evaluations of OCR accuracy have been conducted by UNLV [3]. Page segmentation algorithms, which are crucial components of OCR systems, were at one time evaluated by UNLV based on the final OCR results, but not on the geometric results of the segmentation. Recently [14], we empirically compared various commercial and research page segmentation algorithms, using the University of Washington dataset. We used a well-defined (geometric) line-based metric and a sound statistical methodology to score the segmentation results. Furthermore, unlike the UNLV evaluations, we trained the segmentation algorithms prior to evaluating them.

In this paper we describe in detail the software architecture of the package called PSET, which we used in [14] to evaluate page segmentation algorithms. This package was developed by us at the University of Maryland and will be made available to researchers at no cost. Publication of the package will allow researchers to implement our five-step evaluation methodology and evaluate their own algorithms.

Software architecture can be described using methods such as Petri Nets and Data Flow Diagrams [8]. We describe the architecture of PSET, the I/O file formats, etc. using Object-Process Diagrams (OPDs) [5], which are similar in spirit to Petri Nets.

The package, called the Page Segmentation Evaluation Toolkit (PSET), is modular, written using the C language, and runs on the SUN/UNIX platform. The software has been structured so that it can be used at the UNIX command line level or compiled into other software packages by calling API functions. The description in this paper will aid users in using, updating, and modifying the PSET package. It will also help users to add new algorithm modules to the package and to interface it with other software tools and packages. The PSET package includes three research page segmentation algorithms; [1] a textline-based benchmarking algorithm; and a Simplex-based optimization algorithm for estimating algorithm parameters from training datasets.

This paper is organized as follows. In Section 2, we discuss the page segmentation problem. In Section 3, we present our five-step page segmentation performance evaluation methodology. In Section 4, we describe the architecture and file formats of our PSET package in detail and show how to implement each step of our five-step performance evaluation methodology. In Section 5, we give the hardware and software requirements for using the PSET package. In Section 6, we discuss our future work. Finally in Section 7, we give a summary of the article. A detailed description of our textline-based metric is given in an Appendix for completeness.

[1] We implemented the X-Y cut algorithm [15] and the Docstrum algorithm [16]. Kise [11] provided us the C implementation of his Voronoi-based algorithm.

2 The Page Segmentation Problem

There are two types of page segmentation, physical and logical. Physical page segmentation is a a process of dividing a document page into homogeneous zones. Each of these zones can contain one type of object. These objects can be of type text, table, figure, halftone image, etc. Logical page segmentation is a process of assigning logical relations to physical zones. For example, reading order labels order the physical zones in the order in which they should be read. Similarly, assigning section and sub-section labels to physical zones creates a hierarchical document structure. In this paper, we focus on physical page segmentation and refer to it as simply page segmentation hereafter.

Page segmentation is a crucial preprocessing step for an OCR system. In many cases, OCR engine recognition accuracy depends heavily on page segmentation accuracy. For instance, if a page segmentation algorithm merges two text zones horizontally, the OCR engine will recognize text across text zones and hence generate unreadable text. Page segmentation algorithms can be categorized into three types: top-down, bottom-up, and hybrid approaches. Top-down approaches iteratively divide a document page into smaller zones according to some criterion. The X-Y cut algorithm developed by Nagy *et al.* [15] is a typical top-down algorithm. Bottom-up approaches start from document image pixels, and iteratively group them into bigger regions. The Docstrum algorithm of O'Gorman [16] and the Voronoi-based algorithm of Kise *et al.* [11] are representative bottom-up approaches. Hybrid approaches are usually a mixture of top-down and bottom-up approaches. The algorithm of Pavilidis and Zhou [17] is an example of the hybrid approach that employs a split-and-merge strategy.

3 Performance Evaluation Methodology

In order to objectively evaluate page segmentation algorithms, a performance evaluation methodology should take into consideration the performance metric, the dataset, the training and testing methods, and the methodology of analyzing experimental results. In this section, we introduce a five-step methodology that we proposed earlier [14, 12, 13]. The PSET package is an implementation of this methodology.

Let $\mathcal{D}$ be a given dataset containing (document image, groundtruth) pairs (I, G), and let $\mathcal{T}$ and $\mathcal{S}$ be a training dataset and a test dataset respectively. The five-step methodology is described as follows:

1. Randomly divide the dataset $\mathcal{D}$ into two mutually exclusive datasets: a training dataset $\mathcal{T}$ and a test dataset $\mathcal{S}$. Thus, $\mathcal{D} = \mathcal{T} \cup \mathcal{S}$ and $\mathcal{T} \cap \mathcal{S} = \phi$, where ϕ is the empty set.

2. Define a computable performance metric $\rho(I, G, R)$. Here I is a document image, G is the groundtruth of I, and R is the OCR segmentation result on I. In our case, $\rho(I, G, R)$ is defined as textline accuracy, as described in the Appendix.

3. Given a segmentation algorithm A with a parameter vector $\mathbf{p}^A$, automatically search for the optimal parameter value $\hat{\mathbf{p}}^A$ for which an objective function $f(\mathbf{p}^A; \mathcal{T}, \rho, A)$

assumes the optimal value on the training dataset $\mathcal{T}$. In our case, this objective function is defined as the average textline error rate on a given training dataset:

$$f(\mathbf{p}^A; \mathcal{T}, A, \rho) = \frac{1}{\#\mathcal{T}} \left[\sum_{(I,G)\in\mathcal{T}} 1 - \rho(G, Seg_A(I, \mathbf{p}^A)) \right].$$

4. Evaluate the segmentation algorithm A with the optimal parameter $\hat{\mathbf{p}}^A$ on the test dataset $\mathcal{S}$ by

$$\Phi\left(\{\rho(G, Seg_A(I, \hat{\mathbf{p}}^A)) | (I,G) \in \mathcal{S}\}\right)$$

   where Φ is a function of the performance metric ρ on each (document image, groundtruth) pair (I, G) in the test dataset $\mathcal{S}$, and $Seg_A(\cdot,\cdot)$ is the segmentation function corresponding to A. The function Φ is defined by the user. In our case,

$$\Phi\left(\{\rho(G, Seg_A(I, \hat{\mathbf{p}}^A)) | (I,G) \in \mathcal{S}\}\right) = 1 - f(\hat{\mathbf{p}}^A; \mathcal{S}, \rho, A),$$

   which is the average of the textline accuracy $\rho(G, Seg_A(I, \hat{\mathbf{p}}^A))$ achieved on each (document image, groundtruth) pair (I, G) in the test dataset $\mathcal{S}$.

5. Perform a statistical analysis to evaluate the statistical significance of the evaluation results, and analyze the errors to identify/hypothesize why the algorithms perform at their respective levels.

4 Architecture, File Formats, and Evaluation Methodology

In this section, we first describe the software architecture of the PSET package and the formats of the files used to communicate with the package. Next we show how this software package can be used to implement the five steps of the page segmentation evaluation methodology described in Section 3. Generic file format descriptions as well as specific examples are provided, for clearer understanding. This description of the architecture and file formats will allow users to i) understand the working of the PSET package, ii) replicate our results, iii) modify the parameter files for datasets, metrics, etc. and conduct their own evaluation experiments, iv) understand, maintain and improve the software, and v) evaluate new algorithms and compare the results with existing algorithms. The PSET package has been used to evaluate five page segmentation algorithms [14, 13].

4.1 Architecture and File Formats

The PSET package can be used to i) automatically train a given page segmentation algorithm, i.e., automatically select optimal algorithm parameters on a given training dataset, and ii) evaluate the page segmentation algorithm with the optimal parameters found in i) on a given test dataset. Figure 1 shows the overall architecture of the PSET package and illustrates these two functionalities.

The overall architecture shows all the input files that are needed to conduct the training and testing experiments for a given page segmentation algorithm, and all the

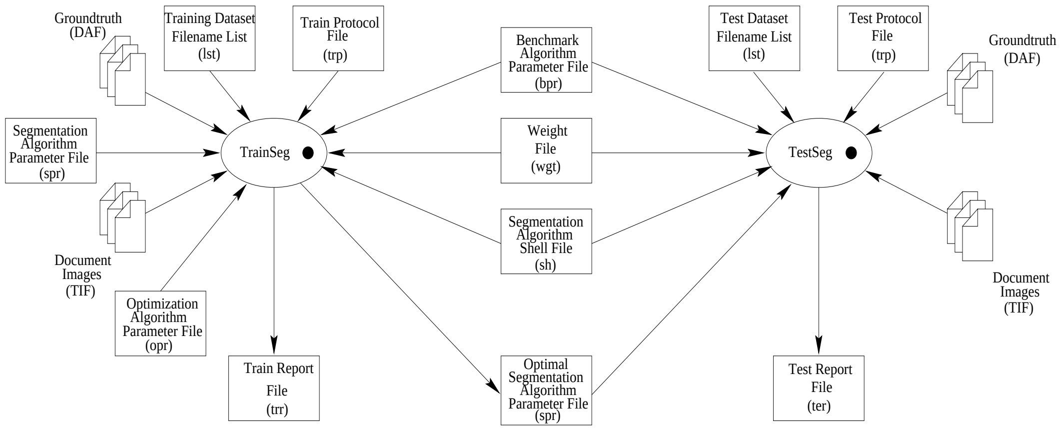


Figure 1: Overall PSET architecture. The left half of the architecture represents the training phase; the right half represents the testing phase. Note that in the testing phase, the optimal page segmentation parameter found in the training phase is used. The training and testing phases use the same performance metric related input files (benchmark algorithm parameter file (bpr) and weight file (wgt)) and the same segmentation algorithm shell file (sh).

Table 1: Summary of the file formats in the PSET package.

File Type	Extension	Description
Dataset List File	lst	It saves the root name of each image in a dataset.
Train Protocol File	trp	It saves the protocol parameters of the training experiment.
Test Protocol File	tep	It saves the protocol parameters of the testing experiment.
Segmentation Algorithm Parameter File	spr	It saves the parameters of a page segmentation algorithm that are to be trained.
Benchmarking Algorithm Parameter File	bpr	It saves all parameters of a benchmarking algorithm.
Optimization Algorithm Parameter File	opr	It saves all parameters of an optimization algorithm.
Groundtruth File	DAF	It saves document images and their groundtruth information.
Segmentation Result File	dafs	It saves document images and their segmentation results.
Train Report File	trr	It saves the training result of a segmentation algorithm.
Test Report File	ter	It saves the test result of a segmentation algorithm.
Weight File	wgt	It saves a set of weights for a set of error measures.
Segmentation Algorithm Shell File	sh	It saves a shell command for running segmentation algorithm executable. It is a Bourn shell program.

output files generated by the training and testing procedures. Table 1 lists all the files used, their purposes, and their file name extensions.

Input files include various initial algorithm parameter files (an optimization algorithm parameter file (opr), a page segmentation algorithm parameter file (spr), and a benchmark algorithm parameter file (bpr)), dataset files (lst), a shell file (sh), and experimental protocol files (training protocol file (trp) and test protocol file (tep)). Users need to provide these files to the PSET package to conduct training or testing experiments. The output files of the training phase include a training report file (trr) and an optimal segmentation algorithm parameter file (spr). The training report file (trr) records intermediate as well as final training results of the training experiment. The optimal segmentation algorithm parameter file (spr) records the optimal segmentation algorithm parameter values found in the training phase. The output of the testing phase is a testing report file (ter), which records a set of error measures, timing and performance scores for each image in the test dataset, and a final average performance score over all images in the test dataset. Figure 2 shows various input file formats. Figure 3 shows the training report file format and Figure 4 shows the test report file format.

```
# [comments]

DATASET              = <dataset file name>
GROUNDTRUTH_DIR      = <groundtruth directory name>
IMG_DIR              = <image directory name>
GT_SUFFIX            = <groundtruth file suffix>
SG_SUFFIX            = <segmentation result file suffix>
IMG_SUFFIX           = <image file suffix>
TRAIN_RESULT_DIR     = <training result file location>
OPT_ALG              = <optimization algorithm name>
BEN_ALG              = <benchmark algorithm name>
SEG_ALG              = <page segmentation algorithm name>
```

(a)

```
# [comments]

DATASET              = <testing dataset file name>
GROUNDTRUTH_DIR      = <groundtruth directory name>
IMG_DIR              = <image directory name>
GT_SUFFIX            = <groundtruth file suffix>
SG_SUFFIX            = <segmentation result file suffix>
IMG_SUFFIX           = <image file suffix>
TEST_RESULT_DIR      = <testing result file location>
BEN_ALG              = <benchmark algorithm name>
SEG_ALG              = <page segmentation algorithm name>
```

(b)

```
# [comments]
<parameter 1 name>   =   <value>
<parameter 2 name>   =   <value>
        .            =      .
        .            =      .
        .            =      .
<parameter N name>   =   <value>
```

(c)

File Attribute Name	Description
DATASET	The filename of a list file that saves the root name of each image in a dataset.
GROUNDTRUTH_DIR	The location of the groundtruth files.
IMG_DIR	The location of the image files.
GT_SUFFIX	The suffix of a groundtruth filename, e.g. the suffix of groundtruth file "A001.DAF" is ".DAF".
SG_SUFFIX	The suffix of a segmentation result filename, e.g. the suffix of segmentation result file "A001.dafs" is ".dafs".
IMG_SUFFIX	The suffix of an image filename, e.g. the suffix of image file "A001BIN.TIF" is "BIN.TIF".
TRAIN_RESULT_DIR	The location of the training result files generated by a training experiment.
TEST_RESULT_DIR	The location of the testing result files generated by a test experiment.
OPT_ALG	The name of the optimization algorithm that is to be used.
BEN_ALG	The name of the benchmarking algorithm that is to be used.
SEG_ALG	The name of the page segmentation algorithm that is to be used.

(d)

Figure 2: Input file formats. The training protocol file format is shown in (a), the test protocol file format is shown in (b), and the algorithm parameter file format is shown in (c). The description of the attributes in (a) and (b) is given in (d).

The parameter values in the parameter files are first read into the corresponding data structures inside the TrainSeg and the TestSeg modules as shown in Figure 5. The Train module shown in Figure 5(a) is shown at a finer level of detail in Figure 6, where the interaction of the optimization algorithm and the objective function computation module is illustrated. A detailed view of the *Objective Function Genscore* showing the interaction between the segmentation algorithm module and the performance metric computation module is shown in Figure 7(a). Finally, a blown-up view of the Test module shown in Figure 5(b) is shown in Figure 7 (b).

4.2 Implementing the Evaluation Methodology

In this section we show how a user can implement each step of the five-step evaluation methodology described in Section 3. Each variable in the methodology is mapped to a specific parameter file and each step is mapped to a specific group of modules in the package.

1. The training dataset $\mathcal{T}$ is specified in the image root name list file (lst). The file name and location of the list file and the location of the image and groundtruth files

```
# [experimental environments]
#
# Feval  p[1]    p[2]    . . .  p[n]    score   timing  plow[1] plow[2] . . .  plow[n] Flow
1        <data>  <data>  . . .  <data>  <data>  <data>  <data>  <data>  . . .  <data>  <data>
2        <data>  <data>  . . .  <data>  <data>  <data>  <data>  <data>  . . .  <data>  <data>
.          .       .     . . .    .       .       .       .       .     . . .    .       .
.          .       .     . . .    .       .       .       .       .     . . .    .       .
.          .       .     . . .    .       .       .       .       .     . . .    .       .
M        <data>  <data>  . . .  <data>  <data>  <data>  <data>  <data>  . . .  <data>  <data>

Optimal_Parameter_Vector = <param 1> <param 2> . . . <param N>
Optimal_Performance_Value = <data>

# End of the training.
```

(a)

Item Name	Description
Feval	Number of objective function evaluations.
p[1], p[2], . . . , p[n]	Current objective function parameter vector value; here the objective function parameter vector is the page segmentation parameter vector being trained. n is the dimensionality of the parameter vector.
score	Current performance measure, in this case, textline error rate.
timing	The time it takes to obtain the current score.
plow[1], plow[2], . . . , plow[n]	The objective function parameter vector value that gives the best score so far.
Flow	The best score so far — in this case, the minimum textline error rate so far.

(b)

Figure 3: The training report file format. The format is shown in (a) and the description of each column entry in (a) is shown in (b).

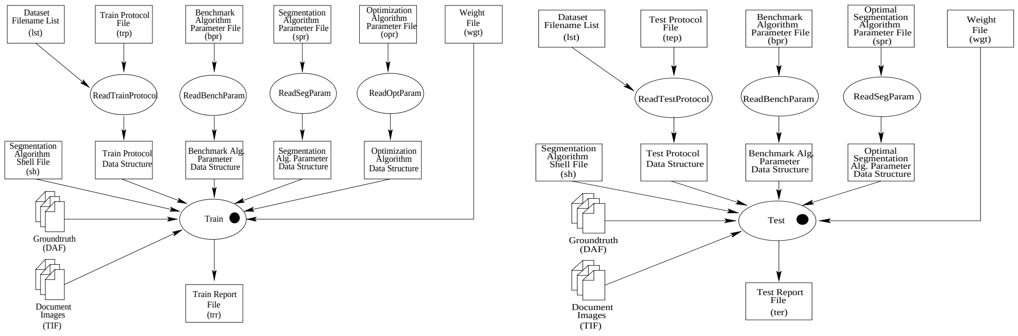
```
# <experimental environments>
#
#Img                 nSpl    nMrg    nFA     nSplL   nMrgL   nMisL   nErrL   nGtL    score   timing
<img_root_name 1>    <data>  <data>  <data>  <data>  <data>  <data>  <data>  <data>  <data>  <data>
<img_root_name 2>    <data>  <data>  <data>  <data>  <data>  <data>  <data>  <data>  <data>  <data>
.                      .       .       .       .       .       .       .       .       .       .
.                      .       .       .       .       .       .       .       .       .       .
.                      .       .       .       .       .       .       .       .       .       .
<img_root_name M>    <data>  <data>  <data>  <data>  <data>  <data>  <data>  <data>  <data>  <data>

The average textline accuracy = <data>

# End of testing.
```

(a)

Column Entry	Description
Img	The root name of the current image file.
nSpl	The number of split errors.
nMrg	The number of horizontal merge errors.
nFA	The number of false alarm errors.
nSplL	The number of split textlines.
nMrgL	The number of horizontally merged textlines.
nMisL	The number of mis-detected textlines.
nErrL	The number of error textlines (textlines that are either split, horizontally merged or mis-detected).
nGtl	The number of groundtruth textlines.
score	The performance measure (textline error rate) on current image.
timing	The time taken to obtain the score.

(b)

Figure 4: The test report file format. The format is shown in (a) and the description of each column entry in (a) is shown in (b).



(a) (b)

Figure 5: Parameter reading stage of the training phase (a) and the testing phase (b). At this level, various parameter files are read into their corresponding data structures which are fed into the Train and Test modules.

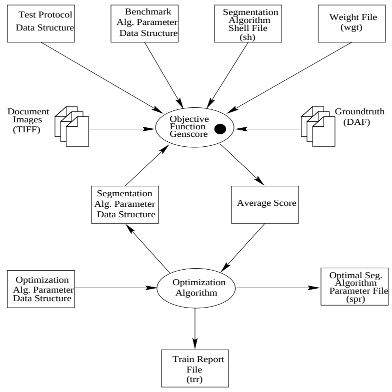
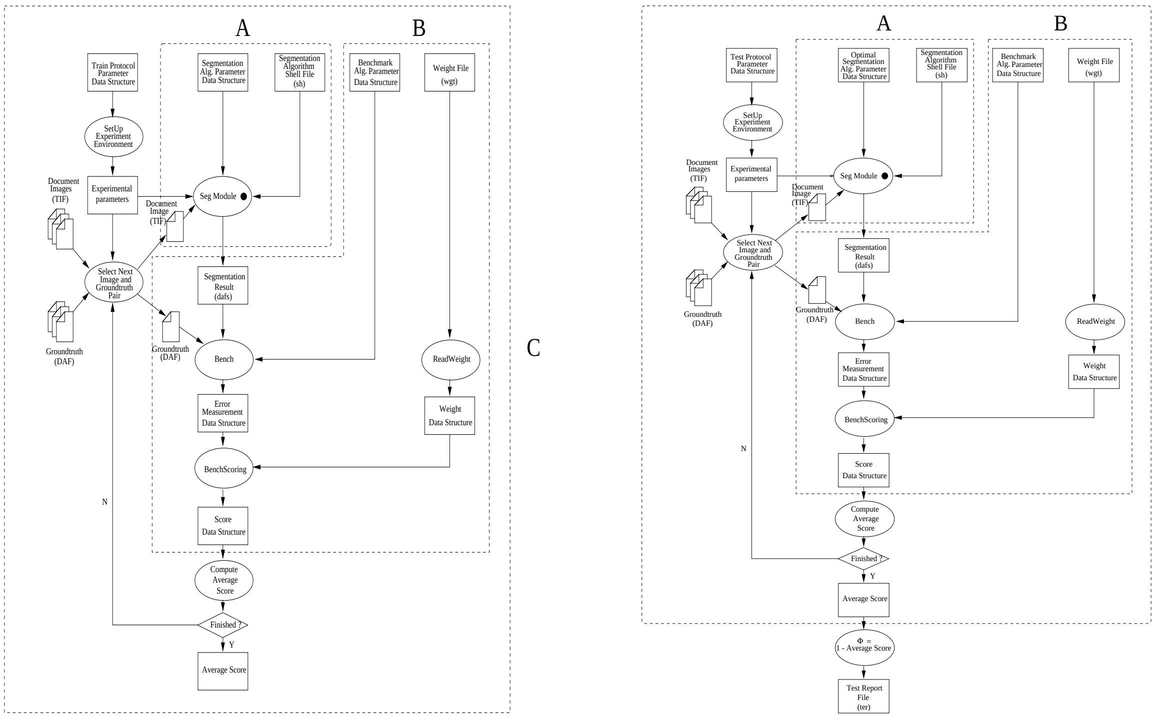


Figure 6: The Train module. In this module, the objective function is optimized over a given training dataset. Two files are generated by this module, a train report file (trr) and an optimal segmentation algorithm parameter file (spr).

are specified in the training protocol file (trp). This information is later read into the Train Protocol Parameter Data Structure as shown in Figure 5(a). Similarly, a test dataset $\mathcal{S}$ is specified in another image root name list file (lst). The file name and location of the list file and the location of image and groundtruth files are specified in the test protocol file (tep). This information is later read into the test protocol parameter data structure as shown in Figure 5(b). Other experimental protocol parameters such as file suffix and algorithms used are also specified in the training protocol file (trp) and test protocol file (tep). Figures 2(a) and (b) show generic formats for these two files and Figure 8 shows samples of these two files.

2. The performance metric $\rho(I, G, R)$ is computed in module B, shown in Figures 7(a) and (b). (I, G) is an (image, groundtruth) pair, which is represented by two single pages in the architecture, and R is the segmentation result file represented by Segmentation Result (dafs). The error counter algorithm for generating a set of error measures is implemented in the *Bench* module. In the *BenchScoring* module,



(a) (b)

Figure 7: Software architectures of the objective function module and the test module. Module A represents the page segmentation algorithm module, module B represents the page segmentation error counter and scoring module, and module C represents the objective function module. The test module in (b) has sub-modules similar to those in (a). It also has a module for computing a final testing performance score (average textline accuracy).

a weighted error measure $1 - \rho(I, G, R)$ is computed. The formal definitions of error measures and performance metrics are given in the Appendix. To compute a performance metric, two input files, a benchmark Algorithm Parameter File (bpr) and a weight file (wgt), are required. Examples of these two files are shown in Figure 13. Users can substitute their own performance metrics and error counters in place of these two modules. However, this also requires that the users write a new *ReadBenchParam* module and define a new benchmark algorithm parameter data structure as shown in Figure 5.

3. The objective function $f(\mathbf{p}^A; \mathcal{T}, A, \rho)$ is represented by the module C in Figure 7(a), where page segmentation algorithm A is represented by module A, the training dataset $\mathcal{T}$ is specified in the train protocol parameter data structure, the computation of performance metric ρ is conducted in module B, and objective function parameter vector $\mathbf{p}^A$ is represented by the segmentation algorithm parameter data structure in the architecture. The optimization procedure is shown in Figure 6 in a simplified representation. In addition, a benchmark algorithm parameter file (bpr), weight file (wgt), shell file (sh), list file (lst), training protocol file (trp),

```
# Training experiment protocal
# By: Song Mao
# Feb. 21, 2000
# LAMP, UMCP

DATASET             = train.lst
GROUNDTRUTH_DIR     = /fs/mirak2/LAMP/UWIII/ENGLISH/LINEWORD/DAFS/
IMG_DIR             = /fs/mirak2/LAMP/UWIII/ENGLISH/LINEWORD/IMAGEBIN/
GT_SUFFIX           = .DAF
SG_SUFFIX           = .dafs
IMG_SUFFIX          = BIN.TIF
TRAIN_RESULT_DIR    = ./
OPT_ALG             = simplex
BEN_ALG             = textline_based
SEG_ALG             = docstrum
```

(a)

```
# Test experiment protocal
# By: Song Mao
# Feb. 21, 2000
# LAMP, UMCP

DATASET             = test.lst
GROUNDTRUTH_DIR     = /fs/mirak2/LAMP/UWIII/ENGLISH/LINEWORD/DAFS/
IMG_DIR             = /fs/mirak2/LAMP/UWIII/ENGLISH/LINEWORD/IMAGEBIN/
GT_SUFFIX           = .DAF
SG_SUFFIX           = .dafs
IMG_SUFFIX          = BIN.TIF
TEST_RESULT_DIR     = ./
BEN_ALG             = textline_based
SEG_ALG             = xycut
```

(b)

Figure 8: Sample protocol files. From both the train protocol file (a) and the test protocol file (b), we can see that the list files of the training dataset and test dataset are *train.lst* and *test.lst* respectively, the optimization algorithm used is the *Simplex* algorithm, the benchmarking algorithm used is the *Textline-based* algorithm, the page segmentation algorithm trained is the *Docstrum* algorithm, and the page segmentation algorithm tested is the *X-Y cut* algorithm. We can also find the locations of the groundtruth files, image files and training and test result files. Moreover, the suffixes for various files are given for file name manipulation in the PSET API.

optimization algorithm parameter file (opr) and segmentation algorithm parameter file (spr) are required to conduct objective function optimization. Samples of opr and spr are shown in Figure 9. The generic file format of these sample files is shown in Figure 2.

```
# The Simplex Optimization
# Algorithm Parameters
NDIM      =   4
CRIFLG    =   nelder-mead
NMAX      =   500
FTOL      =   0.000001
ALPHA     =   1.0
BETA      =   0.5
GAMMA     =   2.0
SIGMA     =   0.5
P         =   100,80,100,50
SCALE     =   20,20,20,20
```

(a)

```
# The X-Y Cut Page Segmentation
# Algorithm Parameters
ALG_MODE    = func_call
TNX         = 100
TNY         = 80
TCX         = 100
TCY         = 50
```

(b)

Figure 9: Samples of an optimization algorithm parameter file (opr) and a segmentation algorithm parameter file (spr). A sample file for the Simplex optimization algorithm is shown in (a) and a sample file for the X-Y cut segmentation algorithm is shown in (b). Their detailed parameter descriptions can be found in [12].

The optimal objective function parameter vector $\hat{\mathbf{p}}^A$ is stored in the file optimal segmentation algorithm parameter file (spr) shown in Figure 6. Users can substitute their own objective function in place of the architecture shown in Figure 7(a) and their own optimization algorithm module in the place of the *Optimization Algorithm* module shown in Figure 6. Again, they need to write new parameter reading functions and define corresponding data structures. This step generates two files, a training report file (trr) and an optimal segmentation algorithm parameter file (spr). Figure 10(a) shows a sample training report file.

```
#
# File: TrainDocstrum_1,4,2.1,6.trr
# Purpose: training result of the Docstrum algorithm using Simplex algorithm.
# User: maosong
# Date: 09/18/2000/ 19:12:25
# Operating system: SunOS, 5.6, Generic_105181-19
# Machine name: hanzi.cfar.umd.edu
# Working directory: /hanzi/maosong/software/SegEvalToolKit/pset-1.0/experiments/TrainDocstrum
# Machine type: sun4u
# Command line: TrainSeg -p train_protocol.trp -b bench.bpr -o simplex.opr -s docstrum.spr
-w weight.wgt -t TrainDocstrum_1,4,2.1,6.trr -r docstrum_optimal_1,4,2.1,6
#

# Feval p[1]  p[2]  p[3]  p[4]  score  timing plow[1] plow[2] plow[3] plow[4] Flow
1      1.000  4.000 2.100 6.000 39.874 206.6  1.000   4.000   2.100   6.000   39.874
2      2.000  4.000 2.100 6.000 39.698 155.0  2.000   4.000   2.100   6.000   39.698
3      1.000  5.000 2.100 6.000 43.337 206.3  2.000   4.000   2.100   6.000   39.698
4      1.000  4.000 3.100 6.000 44.073 207.5  2.000   4.000   2.100   6.000   39.698
5      1.000  4.000 2.100 7.000 39.874 204.2  2.000   4.000   2.100   6.000   39.698
6      1.250  4.250 2.100 6.250 39.761 172.2  2.000   4.000   2.100   6.000   39.698
7      1.500  4.500 1.100 6.500 34.718 160.4  2.000   4.000   2.100   6.000   39.698
8      1.750  4.750 0.100 6.750 30.138 158.4  2.000   4.000   2.100   6.000   39.698
9      1.438  4.188 1.600 6.438 35.710 162.4  1.750   4.750   0.100   6.750   30.138
10     1.875  3.375 1.100 6.875 25.513 155.1  1.750   4.750   0.100   6.750   30.138
11     2.312  2.562 0.600 7.312 10.513 153.2  1.750   4.750   0.100   6.750   30.138
12     1.766  3.828 1.225 6.766 31.076 156.2  2.312   2.562   0.600   7.312   10.513
13     2.531  3.656 0.350 7.531 27.372 153.2  2.312   2.562   0.600   7.312   10.513
.      .      .     .     .     .      .      .       .       .       .       .
.      .      .     .     .     .      .      .       .       .       .       .
.      .      .     .     .     .      .      .       .       .       .       .
160    2.533  1.975 0.647 7.547 5.336  153.4  2.535   1.978   0.645   7.550   5.336
161    2.533  1.977 0.646 7.548 5.336  153.2  2.533   1.975   0.647   7.547   5.336

Optimal_Parameter_Vector = 2.533 1.975 0.647 7.547
Optimal_Performance_Value = 5.336

# End of the training.
```

(a)

```
#
# File: TestXycut_78,32,35,54.ter
# Purpose: testing result of the X-Y cut algorithm.
# User: maosong
# Date: 09/20/2000/ 10:58:33
# Operating system: SunOS, 5.6, Generic_105181-19
# Machine name: hangul.cfar.umd.edu
# Working directory: /a/hanzi/hanzi/maosong/software/pset-1.0/experiments/TestXycut
# Machine type: sun4u
# Command line: TestSeg -p test_protocol.tep -b bench.bpr -s xycut_optimal.spr
-w weight.wgt -t TestXycut_78,32,35,54.ter
#

# ImgnSpl nMrg nFA nSplL nMrgL nMisL nErrL nGtL score timing
A001 1    0    19  1     0     0     1     35   0.029 3.060
A002 2    0    6   2     0     1     3     5    0.600 2.030
A004 1    0    5   1     0     0     1     44   0.023 2.620
A005 1    46   8   1     52    0     53    62   0.855 2.290
A006 3    0    5   3     0     0     3     116  0.026 2.890
A007 4    0    11  4     0     0     4     127  0.031 3.050
A008 1    0    2   1     0     0     1     104  0.010 2.610
A009 1    0    2   1     0     0     1     47   0.021 2.140
A00A 1    0    2   1     0     0     1     45   0.022 2.170
A00B 2    0    4   2     0     0     2     183  0.011 3.130
A00C 11   0    4   11    0     0     11    155  0.071 2.770
A00D 0    0    4   0     0     1     1     35   0.029 2.000
.    .    .    .   .     .     .     .     .    .     .
.    .    .    .   .     .     .     .     .    .     .
.    .    .    .   .     .     .     .     .    .     .
V00N 2    0    1   2     0     0     2     95   0.021 2.520

The average textline accuracy = 0.829185

# End of testing.
```

(b)

Figure 10: Samples of a training report file format (a) and a test report file format (b). The comment lines provide experimental environment information about the training and test experiments. They are automatically generated by calling various GNU C functions. They are crucial for replicating experimental results. In the data area, both intermediate information and final results are recorded. This information can be used to analyze the convergence properties of the training process and to study the statistical significance of the test experiment results. A detailed description of each column entry can be found in Figure 3(b) and Figure 4(b).

4. After the optimal objective function parameter vector $\hat{\mathbf{p}}^A$ has been found, the page segmentation algorithm is evaluated on a given test dataset $\mathcal{S}$. Figure 7(b) shows the architecture of the test procedure. The test dataset $\mathcal{S}$ is specified in the test protocol parameter data structure. Performance metric ρ is computed in module B. Note that module C here has the same architecture as module C in Figure 7(a). The computation of the final performance value Φ is represented in module Φ. Users can define their own Φ function by changing the *Bench, BenchScoring, Compute Average Score,* and Φ modules in Figure 7(b). This step generates a test report file (ter) which records a performance score for each image in the test dataset as well as a final average performance score over all images in the test dataset. Figure 10(b) shows a sample test report file.

5. The statistical analysis of the test experimental results can be conducted using a standard statistics software package such as S-PLUS [4] or SPSS [6].

4.3 Algorithm Calling Mode in the Segmentation Algorithm Module

An important feature of the PSET package is that there are two page segmentation algorithm calling modes: function call and shell call. If the source code of a segmentation algorithm is available as a function, the user can link the function into the training and testing modules. In many cases, however, source code of a segmentation algorithm is not available, but executable code is. In such cases the shell calling mode can be used to run the segmentation algorithm from within the training or testing module. Furthermore, if a segmentation algorithm source code is not well debugged, e.g., if it leaks memory after each function call, the leaked memory can accumulate after many function calls and can finally cause algorithm crash at some point. The shell call mode is a good solution to this problem since in this case the executable code is used, and after each call all leaked memory is freed. The disadvantage of the shell call mode is that it can be slower than the function call mode. Figure 12 shows the architecture of the software implementation of these two calling modes. A shell file is required in the page segmentation algorithm shell call mode. A sample shell file is shown in Figure 11.

```
#! /bin/sh

Docstrum -t $1 -p $2 -u $3 -d $4 $5 $6 $7
```

Figure 11: A sample shell file.

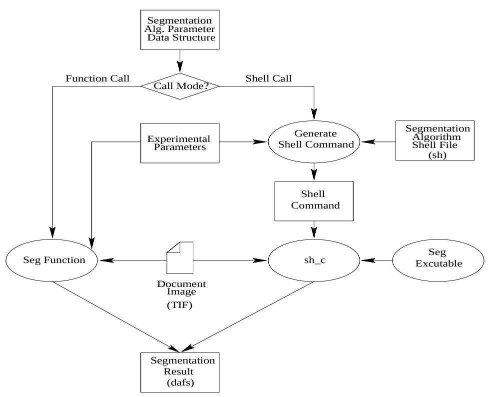


Figure 12: Page segmentation algorithm calling modes: function call and shell call. The left half represents the function calling mode and the right half represents the shell calling mode. The shell calling mode can be used only when the algorithm executable is available; otherwise the function calling mode can be used. Note that the executable is called by the function *sh_c*.

5 Hardware and Software Requirements

The PSET package has been developed in ANSI C on SUN Ultra 1, 2, and 5 workstations running the Solaris 2.6 operating system. The compiler used was GNU gcc 2.7.2. Two public-domain libraries, DAFS and TIFF, were used in PSET and have been included in the distribution. The DAFS data structure library [7] was used for manipulating intermediate datatypes and the TIFF library [2] was used for image I/O.

6 Future Work

We are currently generalizing the PSET package to include i) other metrics, ii) other training/optimization algorithms, and iii) non-text region evaluation. Once the package is in the public domain, we expect that the international community will add other segmentation algorithms to the package. We are also porting the package to the Linux platform. A visualization tool called TRUEVIZ [10] that can display the segmentation and evaluation results of our PSET package is under development. For example, different types of errors can be visualized in various colors. TRUEVIZ can also be used for creating groundtruth for segmentation. Furthermore, we are developing an XML-based representation for zone groundtruth and intend to migrate to this representation from the current DAFS representation.

7 Summary

We have described the architecture and the file formats of a page segmentation evaluation toolkit (PSET). The overall architecture and the file formats were described to illustrate two major functionalities of the PSET package: i) automatically train a given page segmentation algorithm on a given training dataset and ii) evaluate the page segmentation algorithm with the optimal parameters found in i) on a given test dataset. The details of the architecture and samples of file formats were then described as an implementation of our five-step performance evaluation methodology. This paper is intended to assist users in understanding, using, updating and modifying the PSET package. It will also aid programmers who intend to add new algorithm modules to the package and interface it with other software tools.

A Textline-Based Error Measures and Error Metrics

In the following sections, we define page segmentation, a set of textline-based error measurements, and a performance metric that we used in our previous evaluation of page segmentation algorithms [14, 13], These definitions are based on set theory and mathematical morphology [9]. We then define a general metric that users can customize for their individual tasks.

A.1 Page Segmentation Definition

Let I be a document image, and let G be the groundtruth of I. Let $Z(G) = \{Z_q^G, q = 1, 2, \ldots, \#Z(G)\}$ be a set of groundtruth zones of document image I where $\#$ denotes the cardinality of a set. Let $L(Z_q^G) = \{l_{qj}^G, j = 1, 2, \ldots, \#L(Z_q^G)\}$ be the set of groundtruth textlines in groundtruth zone Z_q^G. Let the set of all groundtruth textlines in document image I be $\mathcal{L} = \cup_{q=1}^{\#Z(G)} L(Z_q^G)$. Let A be a given segmentation algorithm, and $Seg_A(\cdot, \cdot)$ be the segmentation function corresponding to algorithm A. Let R be the segmentation result of algorithm A such that $R = Seg_A(I, \mathbf{p}^A)$ where $Z(R) = \{Z_k^R | k = 1, 2, \ldots, \#Z(R)\}$.

Let $D(\cdot) \subseteq \mathcal{Z}^2$ be the domain of its argument. The groundtruth zones and textlines have the following properties: 1) $D(Z_q^G) \cap D(Z_{q'}^G) = \phi$ for $Z_q^G, Z_{q'}^G \in Z(G)$ and $q \neq q'$, and 2) $D(l_i^G) \cap D(l_{i'}^G) = \phi$ for $l_i^G, l_{i'}^G \in \mathcal{L}$ and $i \neq i'$.

A.2 Error Measurements and Metric Definitions

In this section, we define four error measurements and a metric. Let $T_X, T_Y \in Z^+ \cup \{0\}$ be two length thresholds (in pixels) that determine if the overlap is significant or not. Each of these thresholds is defined in terms of an absolute threshold and a relative threshold. The absolute threshold is in pixels and the relative threshold is a percentage. T_X and T_Y are defined as follows:

$$T_X = \min\{HPIX, (100 - HTOL) \cdot h/100\} \tag{1}$$

$$T_Y = \min\{VPIX, (100 - VTOL) \cdot v/100\} \tag{2}$$

where $HPIX$ and $VPIX$ are the the two thresholds in pixels, $HTOL$ and $VTOL$ are the two thresholds in percentages, and h, v are the minimum width and height (in pixels) of two regions that are tested for significant overlap. Users must specify the $HTOL, VTOL, HPIX$ and $VPIX$ parameter values in the benchmark algorithm parameter file (bpr). Figure 13(b) shows a sample benchmark algorithm parameter file.

```
# The Textline-Based Benchmark
# Algorithm Parameters

HTOL    = 90
VTOL    = 80
HPIX    = 11
VPIX    = 8
```

(a)

```
# weight file

wSpl        =   0
wMrg        =   0
wMis        =   0
wFA         =   0
wSplLine    =   1
wMrgLine    =   1
wMisLine    =   1
wFAZone     =   0
```

(b)

Figure 13: Samples of a benchmark algorithm parameter file (bpr) (a) and a weight file (wgt) (b).

Let $E(T_X, T_Y) = \{e \in Z^2 | -T_X \leq X(e) \leq T_X, -T_Y \leq Y(e) \leq T_Y\}$ be a region of a rectangle centered at $(0, 0)$ with a width of $2T_X + 1$ pixels, and a height of $2T_Y + 1$ pixels where $X(\cdot)$ and $Y(\cdot)$ denote the X and Y coordinates of the argument, respectively. We now define two morphological operations: dilation and erosion [9]. Let $A, B \subseteq Z^2$. Morphological *dilation* of A by B is denoted by $A \oplus B$ and is defined as $A \oplus B = \{c \in Z^2 | c = a + b \text{ for some } a \in A, b \in B\}$. Morphological *erosion* of A by B is denoted by $A \ominus B$ and is defined as $A \ominus B = \{c \in Z^2 | c + b \in A \text{ for every } b \in B\}$.

We now define three types of textline based error measurements:

1) Groundtruth textlines that are missed:

$C_L = \Big\{ l^G \in \mathcal{L} | D(l^G) \ominus E(T_X, T_Y)$

$\subseteq (\cup_{Z^R \in Z(R)} D(Z^R))^c \Big\}$,

2) Groundtruth textlines whose bounding boxes are split:

$$S_L = \Big\{ l^G \in \mathcal{L} | (D(l^G) \ominus E(T_X, T_Y)) \cap D(Z^R) \neq \phi, \\ (D(l^G) \ominus E(T_X, T_Y)) \cap (D(Z^R))^c \neq \phi, \\ \text{for some } \; Z^R \in Z(R) \Big\},$$

3) Groundtruth textlines that are horizontally merged:

$$M_L = \Big\{ l^G_{qj} \in \mathcal{L} | \exists l^G_{q'j'} \in \mathcal{L}, Z^R \in Z(R), q \neq q', \\ Z^G_q, Z^G_{q'} \in Z(G) \text{ such that} \\ (D(l^G_{qj}) \ominus E(T_X, T_Y)) \cap D(Z^R) \neq \phi, \\ (D(l^G_{q'j'}) \ominus E(T_X, T_Y)) \cap D(Z^R) \neq \phi, \\ ((D(l^G_{qj}) \ominus E(0, T_Y)) \oplus E(\infty, 0)) \cap D(Z^G_{q'}) \neq \phi, \\ ((D(l^G_{q'j'}) \ominus E(0, T_Y)) \oplus E(\infty, 0)) \cap D(Z^G_q) \neq \phi \Big\}.$$

4) Noise zones that are falsely detected (false alarm):

$$F_L = \Big\{ Z^R \in Z(R) | D(Z^R) \subseteq (\cup_{l^G \in \mathcal{L}} (D(l^G) \ominus E(T_x, T_Y)))^c \Big\}$$

Let the number of groundtruth error textlines be $\#\{C_L \cup S_L \cup M_L\}$ (mis-detected, split, or horizontally merged), and let the total number of groundtruth textlines be $\#\mathcal{L}$. We define the performance metric $\rho(I, G, R)$ as textline accuracy:

$$\rho(I, G, R) = \frac{\#\mathcal{L} - \#\{C_L \cup S_L \cup M_L\}}{\#\mathcal{L}}.$$

In the PSET package, we also define some other error measurements. Table 2 shows the error measurements, the metric defined in the PSET package, and the corresponding symbols used in the above discussion.

Table 2: Summary of error measurements and the corresponding symbols defined in this section.

Error Measure Defined in the PSET package	Equivalent Term in this Section	Description
$nSpl$	none	The number of split errors.
$nMrg$	none	The number of horizontal merge errors.
nFA	$\#F_L$	The number of false alarm errors.
$nSplL$	$\#S_L$	The number of split textlines.
$nMrgL$	$\#M_L$	The number of horizontally merged textlines.
$nMisL$	$\#C_L$	The number of mis-detected textlines.
$nErrL$	$\#\{C_L \cup S_L \cup M_L\}$	The number of error textlines (textlines that are either split, horizontally merged or mis-detected).
$nGtl$	$\#\mathcal{L}$	The number of groundtruth textlines.

In general, the performance metric can be any function of the error measures shown in Table 2. In the PSET package, a performance metric can be defined as a weighted sum of these error measures in function *BenchScoring*. Let $wSpl$ be the weight of the error measurement $nSpl$. The weights of other error measurements are defined similarly. A general performance metric is defined as follows:

$$
\begin{aligned}
N &= wSpl * nSpl + wMrg * nMrg + wFA * nFA + wSplL * nSplL \\
&\quad + wMrgL * nMrgL + wMisL * nMisL, \\
D &= wSpl + wMrg + wFA + wSplL + wMrgL + wMisL, \\
\rho^*(I, G, R) &= \frac{N}{D}. \qquad (3)
\end{aligned}
$$

Figure 14 gives a set of possible errors as well as an experimental example.

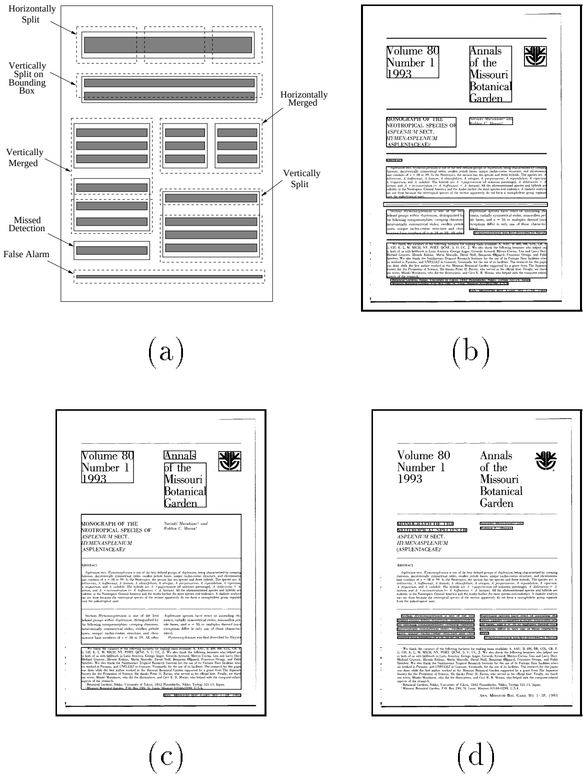


Figure 14: (a) This figure shows a set of possible textline errors. Solid-line rectangles denote groundtruth zones, dashed-line rectangles denote OCR segmentation zones, dark bars within groundtruth zones denote groundtruth textlines, and dark bars outside solid lines are noise blocks. (b) A document page image from the University of Washington III dataset with the groundtruth zones overlaid. (c) OCR segmentation result on the image in (b). (d) Segmentation error textlines. Notice that there are two horizontally merged zones just below the caption and two horizontally merged zones in the middle of the text body. In OCR output, horizontally split zones cause reading order errors whereas vertically split zones do not cause such errors.

Acknowledgement

We would like to thank Dr. Kise of Osaka Prefecture University for providing us with a software implementation of his segmentation algorithm and modifying it for our evaluation purposes; Glenn van Doren of the Department of Defense for supporting this effort; and Dr. Azriel Rosenfeld of the University of Maryland for his comments.

This research was funded in part by the Department of Defense under Contract MDA 9049-6C-1250, Lockheed Martin under Contract 9802167270, the Defense Advanced Research Projects Agency under Contract N660010028910, and the National Science Foundation under Grant IIS9987944.

References

[1] *DARPA Broadcast News Workshop*, Herndon, VA, Feburary 1999. `http://www.itl.nist.gov/iaui/894.01/publications/darpa99/index.htm`.

[2] Aldus Corporation. *TIFF*. `ftp://sgi.com/graphics/tiff/`.

[3] A. D. Bagdanov. The fourth annual test of OCR accuracy. In A. D. Bagdanov, editor, *Annual Report*. Information Science Research Institute, University of Nevada, Las Vegas, NV, 1995.

[4] R. A. Becker, J. M. Chambers, and A. R. Wilks. *The New S Language*. Wadsworth & Brooks/Cole, Pacific Grove, CA, 1988.

[5] D. Dori, I. Phillips, and R. M. Haralick. Incorporating documentation and inspection into computer integrated manufacturing: An object-process approach. In S. Adiga, editor, *Applications of Object-Oriented Technology in Manufacturing*. Chapman & Hall, London, UK, 1994.

[6] J. J. Foster. *Data Analysis Using SPSS for Windows — A Beginner's Guide*. SAGE Publications, London, UK, 1998.

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

[8] C. Ghezzi, M. Jazayeri, and D. Mandrioli. *Software Engineering*. Prentice Hall, Englewood Cliffs, NJ, 1991.

[9] R. M. Haralick and L. G. Shapiro. *Computer and Robot Vision*. Addison-Wesley, Reading, MA, 1992.

[10] T. Kanungo, C. H. Lee, J. Czorapinski, and I. Bella. TRUEVIZ: A groundtruth/metadata editing and visualizing toolkit for OCR. In *Proceedings of SPIE Conference on Document Recognition and Retrieval*, San Jose, CA, January 2001.

[11] K. Kise, A. Sato, and M. Iwata. Segmentation of page images using the area Voronoi diagram. *Computer Vision and Image Understanding*, 70:370–382, 1998.

[12] S. Mao and T. Kanungo. A methodology for empirical performance evaluation of page segmentation algorithms. Technical Report CAR-TR-933, University of Maryland, College Park, MD, December 1999. http://www.cfar.umd.edu/kanungo/pubs/trsegeval.ps.

[13] S. Mao and T. Kanungo. Automatic training of page segmentation algorithms: An optimization approach. In *Proceedings of International Conference on Pattern Recognition*, pages 531–534, Barcelona, Spain, September 2000.

[14] S. Mao and T. Kanungo. Empirical performance evaluation of page segmentation algorithms. In *Proceedings of SPIE Conference on Document Recognition and Retrieval*, pages 303–314, San Jose, CA, January 2000.

[15] G. Nagy, S. Seth, and M. Viswanathan. A prototype document image analysis system for technical journals. *Computer*, 25:10–22, 1992.

[16] L. O'Gorman. The document spectrum for page layout analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:1162–1173, 1993.

[17] T. Pavlidis and J. Zhou. Page segmentation and classification. *Graphical Models and Image Processing*, 54:484–496, 1992.

[18] E. M. Voorhees and D. K. Harman, editors. *The Seventh Text REtrieval Conference (TREC 7)*. National Institute of Standards and Technology, 1998. http://trec.nist.gov/pubs.html.