Table Structure Recognition Based On Robust Block Segmentation

Thomas G. Kieninger

German Research Center for Artificial Intelligence (DFKI GmbH)
Postfach 2080, 67608 Kaiserslautern, GERMANY

ABSTRACT

This paper presents an efficient approach to identify tabular structures within either electronic or paper documents. The resulting *T-Recs* system takes word bounding box information as input, and outputs the corresponding logical text block units (e.g. the cells within a table environment).

Starting with an arbitrary word as block seed the algorithm recursively expands this block to all words that interleave with their vertical (north and south) neighbors. Since even smallest gaps of table columns prevent their words from mutual interleaving, this initial segmentation is able to identify and isolate such columns.

In order to deal with some inherent segmentation errors caused by isolated lines (e.g. headers), overhanging words, or cells spawning more than one column, a series of postprocessing steps is added. These steps benefit from a very simple distinction between type 1 and type 2 blocks: type 1 blocks are those of at most one word per line, all others are of type 2. This distinction allows the selective application of heuristics to each group of blocks.

The conjoint decomposition of column blocks into subsets of table cells leads to the final block segmentation of a homogeneous abstraction level. These segments serve the final layout analysis which identifies table environments and cells that are stretching over several rows and/or columns.

Keywords: document image analysis, block segmentation, table recognition, structure analysis

1. INTRODUCTION

Today the goals of OCR (Optical Character Recognition) systems go far beyond the simple transformation of document images into plain sequences of words but rather concentrate upon correct detection of structural features. While some structure analysis systems focus on the detection of logical objects in a restricted domain like business letters (as described in Dengel[1]) others try to identify more general structure elements like paragraphs, headers or lists. Hu[2] and Condit[3] both describe representing systems of that class.

The intention of such structure recognition systems is not only the transformation of printed material to an electronic representation but rather to feed associated analysis processes such as automatic message extraction, indexing and classification as described by Baumann et al.[4] Thus it is important to extract as many structural information as possible beyond the words themselves.

Whereas analysis techniques like the above mentioned operate on regular text to spot keywords or extract core messages, tabular data needs to be treated differently. To avoid potential errors, it is strongly recommended to make logical objects such as tables explicit. Numerous researchers have recently made lots of effort towards a layout-based logical labeling.

When talking about structure analysis in the context of tabular documents, one realizes two different topics: the first one is the correct determination of textual units (such as cells within tables), which could be described as *segmentation* or *structure recognition*. The second topic is the analysis of the somehow geometrically arranged blocks and the controlled aggregation of these logical entities to higher level structures. This could be called *logical labeling*, *layout-* or *structure analysis*. The ideas presented in this paper consider both topics with a focus on the segmentation.

Contact information:
Email: kieninger@dfki.de; Telephone: +49-631-205-3485; Fax: +49-631-205-3210; WWW: http://www.dfki.uni-kl.de/da/kieni/

The investigation of existing table structure recognition approaches discloses a significant similarity: the identification of table elements is always driven by the detection of separators: Rus and Summers[5] present a system for the segmentation and labeling of general structures, also considering table structures where they are able to detect narrow columns using so-called *White Space Density Graphs* (WDG). Other approaches relying on sufficiently large white spaces are described by Rahgozar et al.[6] who operate on word segments rather than on the bitmap and Tupaj et al.[7] who take plain text output of OCR systems as input.

Others are explicitly looking for ruling lines that determine the table structure. Some representatives here are Green and Krishnamoorthy,[8] who apply a grammar-based analysis on the set of occuring lines to evaluate the table layout, Itonori,[9] who only considers the labeling aspects and expects well segmented blocks as input, or Hirayama[10] with his interesting *DP matching method.* Chandran and Kasturi[11] consider both (ruled lines and so-called *white streams*) to determine the layout structure.

The central idea in our approach is to not explicitly look for any kind of separators (lines or spacings) but rather to identify words that belong to the same logical unit. We are not looking for features that might give evidences to distinguish two textual areas but rather look for features that might tell us that some words belong to the same textual unit and thus build our structures from bottom up.

It is obvious that we do not depend on potentially occuring table lines nor do we expect sufficiently large spacings between blocks. We can also neglect the restriction of the block shape to rectangles as it is necessary in pixel projection based approaches.

In this paper we describe the overall functionality of our system *T-Recs* (TABLE RECOGNITION SYSTEM), starting with the initial block clustering – the central idea. Next we point out a series of necessary postprocessing steps to correct three classes of system inherent errors. Section 4 discusses the decomposition of columns into individual table cells in order to reduce them from their meta-level existence. While so far the system focuses on the block segmentation, the next section outlines the analysis of the cell arrangement within tabular environments. We finally call attention to a Web-based online demo before ending with some concluding remarks. The appendix shows two more examples and points out implementation-specific details.

2. THE INITIAL CLUSTERING ALGORITHM

The basic idea of the whole system is the initial block segmentation that might be described as a *clustering of words*. Whereas other bottom-up segmentation approaches determine the lines from horizontally adjacent words and the blocks from vertically adjacent lines (like e.g. Gorman[12] who has used nearest neighbors to determine wordclusters), our system directly evaluates textblock structures from word segments.

Therefore we take an arbitrary word as the seed for a new block. As seen in the example of figure 1, we draw a virtual stripe over the bounding box of the word (*consists*) that is to be expanded. This stripe has the width of the bounding box itself and vertically reaches to the directly adjacent lines. All words that overlap with that stripe will be bound to the same block as the initial seed. Thus the block consists of all words that are linked with the graph seen on the right of figure 1.

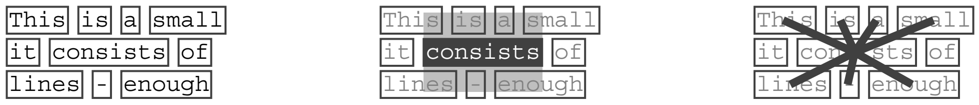


Figure 1. Vertical neighbors of the word "consists".

This expansion procedure will recursively be applied to all the added words until no more newly overlapping words are found, as described in the following algorithm:

1. Find an arbitrary word W_x that has not yet been marked as *expanded*
2. Create a new block B_i

3. mark W_x as *expanded* and add it to B_i
4. Evaluate all horizontally overlapping words W_j in the previous and next line
5. For all these words W_j recursively perform steps 3, 4 and 5
6. If no more overlapping and unmarked words are found, increment i and go to step 1
7. Stop, if no more unmarked words are found in the document

The complete graph for the block of the above example after all expansions are finished is seen in figure 2.

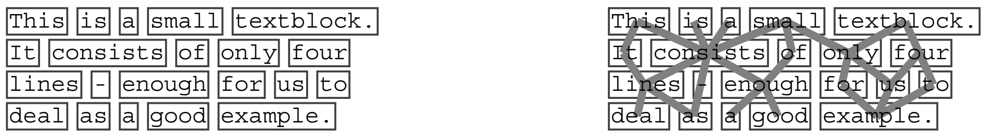
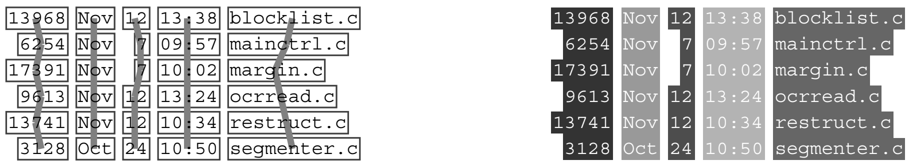


Figure 2. The initial block segmentation graph.

2.1. Strengths Of The Approach

In the examples shown so far, the strengths of this approach are not very obvious since segmentation of regular text blocks might as well be achieved by many exisiting systems. It moreover proves its advantages when applied to tabular structures as shown in figure 3: here we see part of a directory listing with very narrow gaps. Since there are no interleaving words between adjacent columns, the individual column entities are clearly isolated. (For better visualization each block is painted in a different gray shade at the right side.) Besides the above mentioned strengths, the algorithm is characterized by the following features:



13968 Nov 12 13:38 blocklist.c
6254 Nov 7 09:57 mainctrl.c
17391 Nov 7 10:02 margin.c
9613 Nov 12 13:24 ocrread.c
13741 Nov 12 10:34 restruct.c
3128 Oct 24 10:50 segmenter.c

Figure 3. Segmentation of a tabular environment.

- It operates on either electronic (ASCII) or paper (OCR output) documents. The necessary word bounding box geometry for ASCII files can easily be derived from a small preprocessor.
- It neglects grid lines. This allows a faster image preprocessing. (The above mentioned ASCII preprocessor detects and suppresses these grid-lines in order to provide comparable input.)
- It disregards textual contents. Thus it can be applied to low quality documents such as faximiles to perform its block segmentation.
- It detects table columns with very narrow gaps (approximately 1 space).
- It is able to handle *non-Manhattan* layout*, since we do not perform any projections to find column candidates. Blocks do not even need to be left- or right justified.

*Kise et al.[13] use the term *"non-Manhattan"* to refer to a page layout that consists of non-overlapping components that are separated by white space and are not limited to rectangular shapes.

- It detects table-like structures within regular text (e.g. as in logfiles or directory listings) without the presence of characteristic table headers.
- It detects table cells that span more than one document line (as in figure 4).
- It is universally applicable to any document (no limited, fixed set of table classes; no specific rules and no learning phase necessary)

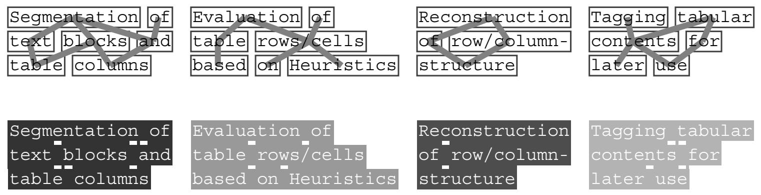


Figure 4. Table cells spanning more than one document line.

2.2. Drawbacks Of The Initial Algorithm

The nature of this segmentation algorithm also has some inherent drawbacks. Thus it would not be able to cluster single lines as in headers because they do not have vertical neighbors and thus they would be interpreted as tables with one line height where each word would represent a column. The same problem occurs with words at the end of a non justified block if they do not overlap with the rest of the block.

A second class of errors occurs with regular paragraphs (typically of only a few lines height) that have a space at the same x position on all lines. The paragraph would hence not be identified as only one homogeneous block.

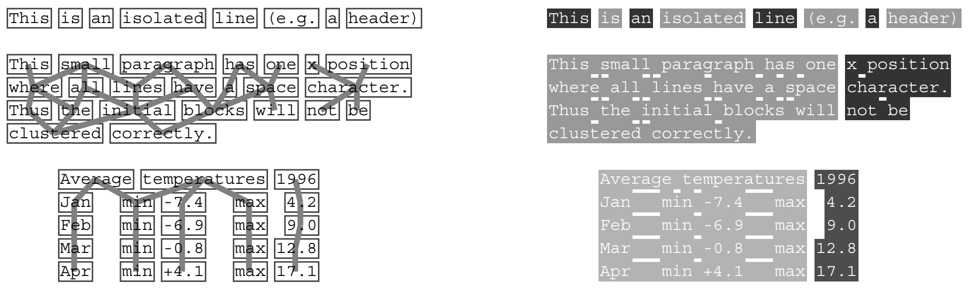

This is an isolated line (e.g. a header)
This small paragraph has one x position
where all lines have a space character.
Thus the initial blocks will not be
clustered correctly.
Average temperatures 1996
Jan min -7.4 max 4.2
Feb min -6.9 max 9.0
Mar min -0.8 max 12.8
Apr min +4.1 max 17.1

Figure 5. A collection of inherent problems.

Another type of error occurs in table columns that have a common header. Such a header would by fault glue the underlying columns together. Figure 5 illustrates all these situations together with the initial block assignment indicated by different colors of the segments.

3. POSTPROCESSING OF INITIAL BLOCK SEGMENTS

To avoid these errors a set of postprocessing steps has to be applied to the initial structures. Most mentionable here is the distinction of blocks between *type 1* and *type 2*: blocks with exactly one word (or *token*) in each line are classified as *type 1* (see figure 3 for an example); all others are of *type 2* (see figures 2 or 4). Obviously, type 1 blocks are those of typical simple table columns.

This classification of blocks allows to selectively apply different mechanisms and heuristics to each of the two classes. In the following we will describe these postprocessing steps and their effects applied to the examples of figure 5.

3.1. Reclustering Of Wrongly Isolated Blocks

In this processing step we make the most use of the above mentioned block classification. Potential candidate blocks for this process must of course either have a left or right neighbor part that is to be merged. At first, we evaluate the spaces between the words of the adjacent blocks.

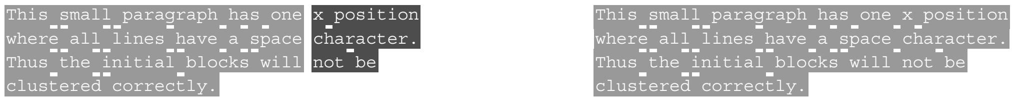

This small paragraph has one x position
where all lines have a space character.
Thus the initial blocks will not be
clustered correctly.
This small paragraph has one x position
where all lines have a space character.
Thus the initial blocks will not be
clustered correctly.

Figure 6. Merging adjacent type 2 blocks.

For type 2 blocks we assume the sum of these spaces (relative to the words average character width) to be slightly bigger than the number of contacting lines. With respect to some tolerances we then decide whether or not the two blocks are merged together.

The distinction of type 1 and 2 guarantees type 1 columns to stay isolated regardless of their gap width whereas at the same time the incorrectly isolated type 2 blocks are bound together. Figure 6 shows two blocks that would be affected by this procedure and the appropriate result.

3.2. Isolation Of Merged Columns

The second problem we are facing are table columns melted together by e.g. a common header. While looking for a simple feature to identify those columns, we realized the *one-to-one relation* one can observe in between the words of typical table columns to be a valuable criterion. This relation is given if word W_a has exactly one lower neighbor W_b and W_b has only that one upper neighbor W_a.

The segmentation graph of the adhered columns block shown in the example of figure 5 also points out a relatively little number of nodes with more than 2 edges in contrast to the regular type 2 blocks – a visual indicator for the discussed relation between words.

Our next step is straightforward: we separate all words that stand in such a one-to-one relation into new blocks which we call *splitted_son* blocks. Thereby we do not consider any other constraints such as resulting block height or word contents. The result for the above mentioned example is seen in figure 7 but the overall procedure is not yet finished.

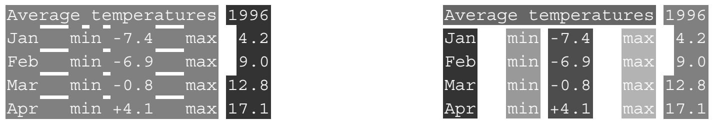


Figure 7. Isolation of merged columns.

Since this procedure is applied to *all* type 2 blocks, it is obvious that there might be numerous words standing in such a one-to-one relation but nevertheless they do not establish a table column. However, we did not apply any more conditions to the above isolation process, because the real value of this separation can best be rated afterwards.

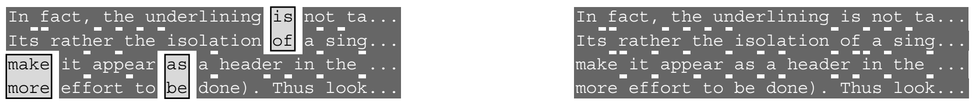

In fact, the underlining is not ta...
Its rather the isolation of a sing...
make it appear as a header in the ...
more effort to be done). Thus look...
In fact, the underlining is not ta...
Its rather the isolation of a sing...
make it appear as a header in the ...
more effort to be done). Thus look...

Figure 8. Remerging of splitted son blocks

A simple rule for a column block is that it always occurs in the context of other columns. Starting from the current intermediate result, we are inspecting the neighborhood of the newly separated blocks. The number of surrounding type 1 blocks, their height, the isolating space to the left and right and optionally the similarity of the words within a column candidate are taken to evaluate an overall evidence. If this value does not reach a given threshold, the blocks are merged with their father block again. Figure 8 shows an example of a type 2 block with three *splitted sons* and the result after they have been remerged to their *father block*.

3.3. Clustering Of Separated Words

Words that do not have an upper nor a lower neighbor might belong to an isolated line (e.g. a header), stick out of the end of a non justified block or represent the content of a table cell. For all except the last case the initial clustering algorithm keeps these words isolated by fault.

Thus, we first have to decide whether such an isolated word matches with a table cell or not and therefore we need a global view over all potential columns. To gain this view, we step through the blocks and wherever two or more distinct blocks occur in a horizontal neighborhood relation, we assume to have a table environment and evaluate what we call a corresponding *margin structure* which itself consists of *margin points*.

The *margin structure* holds information about the vertical range of the potential table environment and carries a list of *margin points*. These points indicate the left and right boundaries of all successive blocks that either have horizontal neighbors or do not exceed a given width. Margin points will only be newly allocated, if there is no existing point within a given range. The points carry information whether they have been triggered by a left or right block border (therefore we call them left and right margin points) and most important they accumulate the number of lines of the blocks referencing that point in their *reference counter*. A large vertical space or a single block covering most part of the document width will close a currently evaluated margin structure.

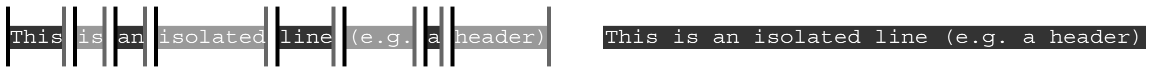

This is an isolated line (e.g. a header)
This is an isolated line (e.g. a header)

Figure 9. Evaluation of block-margins to identify isolated words.

For the next step, the margin-points of all blocks are inspected. If the reference counter of either left or right side of a block does not reach a given limit (which depends on the margins vertical range), this block will potentially be merged with an existing neighbor to the corresponding side that occurs within a specified range (approx. 2 spaces).

The effect of that procedure is to identify isolated words that do not fit into a surrounding table column. Figure 9 shows the evaluated margin points and the resulting block after execution of this procedure to the previously isolated blocks.

4. DECOMPOSITION OF META LEVEL BLOCKS

Whereas typical type 2 blocks all represent structural (text) units like e.g. *paragraphs* or sometimes even *table cells*, the type 1 blocks represent table columns as *aggregation of individual cells*. To achieve an unique abstaction level on our heterogeneous collection of blocks, we need to split all type 1 blocks into their table cell segments. The result of this process applied to the example of figure 7 is shown in figure 10.

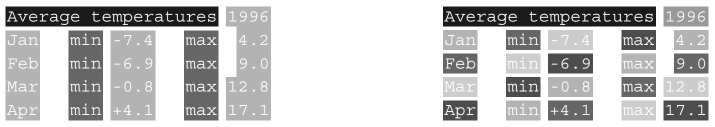


Figure 10. Fragmentation of type 1 columns.

4.1. Type 2 Blocks Within Type 1 Context

In the context of type 1 blocks we might as well find type 2 blocks that need to be treated in the same way as described above, since their line segments belong to individual table rows which themselves are embodied by the surrounding type 1 block fragments.

This constellation is typical for e.g. invoice letters containing several positions in a table setting. There we might e.g. find the slots *position*, *number items*, *item name*, *item description*, *unit price*, and *price*. Figure 11 shows a similar example. Our approach would keep the *item description* column unchanged since it appears to be a type 2 block.

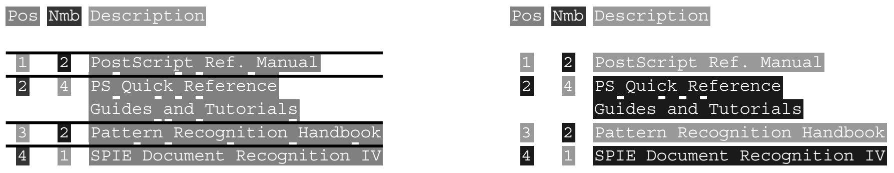


Figure 11. Fragmentation of type 2 columns.

We then assume the fragments of the type 1 blocks to determine the logical rows. The separation of table rows is visualized by horizontal lines (as seen on the left). These lines also determine the breakpoints for the embedded type 2 block.

Before applying this procedure to a block, we need to determine its *degree of line filling*. The blocks of interest are inspected for words that might as well have fitted at the end of a preceeding line. If such words occur, the block would probably have been formatted differently, since modern wordprocessors try to fill up lines as much as possible. A situation like this is an indicator for a logical separation and triggers the discussed separation of the block. The relative number of surrounding type 1 blocks in a table together with the average words per line count give further evidences to apply this splitting procedure on a block. The final segmentation result is seen on the right of figure 11.

5. DETERMINATION OF ROW / COLUMN STRUCTURE

After having gained the segmentation of all elementary blocks (down to table cell level), we need to further exploit these blocks, locate potential table environments and map blocks to their correlating rows and columns.

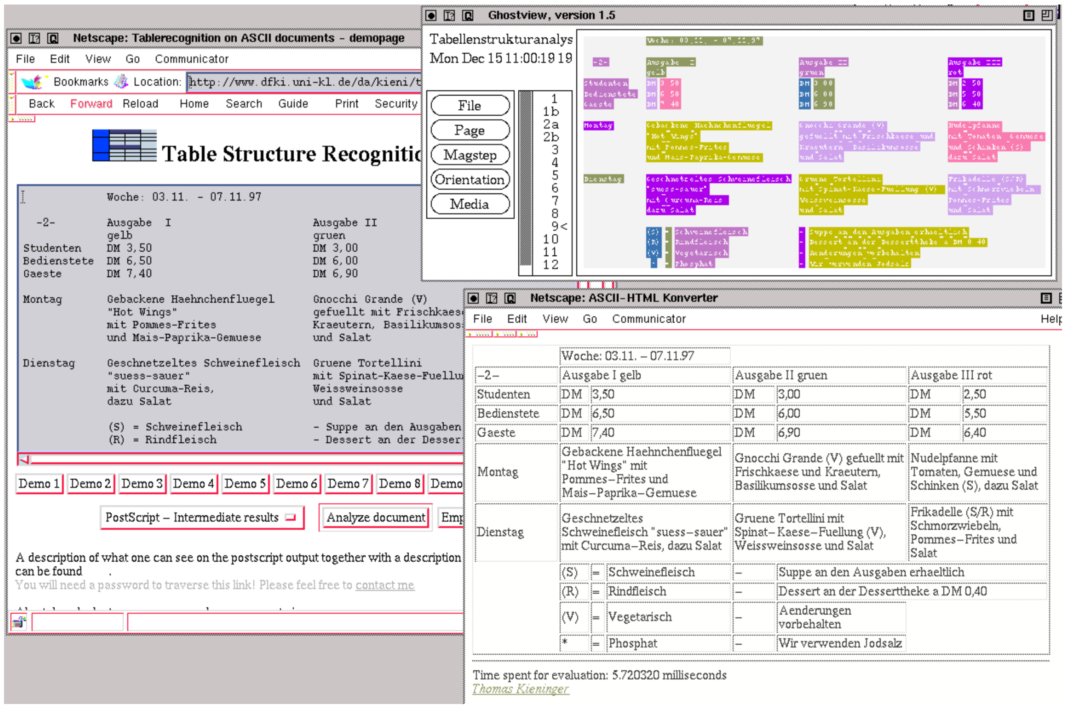
To do this, we reuse the margin and margin point evaluation subsystem used for the detection of isolated words as described in section 3.3. The occurence of blocks in a horizontal neighborship leads to the creation of an appropriate margin structure with a list of margin points.

While traversing the list of margin points from left to right we recognize all transitions from a right to a left margin point as column separator and thus determine the number of columns and their dimensions. Blocks stretching over one or more separators will thus be identified as multicolumn cells.

To evaluate the row structure we make use of a logical row counter. The lines of the document will be traversed from top to bottom and the occurence of a block that starts in that line causes the row counter to be increased by one. The current row counter value is assigned to every document line. Blocks that span more than one table row can now easily be identified by a different row count between top and bottom line. The difference itself indicates the number of overstreched rows.

6. ONLINE DEMO

As mentioned before, our approach can be applied to OCR as well as to ASCII documents. The necessary bounding box information can be easily derived from either source. For the sake of a Web-based, interactive demo environment we have built a *CGI* application (Common Gateway Interface[14]) upon our ASCII interface, that can be accessed via an HTML form. The URL that points you to that page is: **http://www.dfki.uni-kl.de/da/kieni/t_recs/**.


Netscape: Tablerecognition on ASCII documents – demopage
File Edit View Go Communicator
Bookmarks Location: http://www.dfki.uni-kl.de/da/kieni/t
Back Forward Reload Home Search Guide Print Security
Table Structure Recognitic
Demo 1 Demo 2 Demo 3 Demo 4 Demo 5 Demo 6 Demo 7 Demo 8 Demo
PostScript – Intermediate results
Analyze document
Emp
Ghostview, version 1.5
Tabellenstrukturanalys
Mon Dec 15 11:00:19 19
File
Page
Magstep
Orientation
Media
Netscape: ASCII-HTML Konverter
File Edit View Go Communicator
Help

```
               Woche: 03.11. - 07.11.97

  -2-          Ausgabe  I                      Ausgabe II
               gelb                            gruen
Studenten      DM 3,50                         DM 3,00
Bedienstete    DM 6,50                         DM 6,00
Gaeste         DM 7,40                         DM 6,90

Montag         Gebackene Haehnchenfluegel      Gnocchi Grande (V)
               "Hot Wings"                     gefuellt mit Frischkaese
               mit Pommes-Frites               Kraeutern, Basilikumsos
               und Mais-Paprika-Gemuese        und Salat

Dienstag       Geschnetzeltes Schweinefleisch  Gruene Tortellini
               "suess-sauer"                   mit Spinat-Kaese-Fuellu
               mit Curcuma-Reis,               Weissweinsosse
               dazu Salat                      und Salat

               (S) = Schweinefleisch           - Suppe an den Ausgaben
               (R) = Rindfleisch               - Dessert an der Desser
```

A description of what one can see on the postscript output together with a description can be found .
You will need a password to traverse this link! Please feel free to contact me

	Woche: 03.11. – 07.11.97						
–2–	Ausgabe I gelb			Ausgabe II gruen		Ausgabe III rot	
Studenten	DM	3,50		DM	3,00	DM	2,50
Bedienstete	DM	6,50		DM	6,00	DM	5,50
Gaeste	DM	7,40		DM	6,90	DM	6,40
Montag	Gebackene Haehnchenfluegel "Hot Wings" mit Pommes–Frites und Mais–Paprika–Gemuese			Gnocchi Grande (V) gefuellt mit Frischkaese und Kraeutern, Basilikumsosse und Salat		Nudelpfanne mit Tomaten, Gemuese und Schinken (S), dazu Salat	
Dienstag	Geschnetzeltes Schweinefleisch "suess–sauer" mit Curcuma–Reis, dazu Salat			Gruene Tortellini mit Spinat–Kaese–Fuellung (V), Weissweinsosse und Salat		Frikadelle (S/R) mit Schmorzwiebeln, Pommes–Frites und Salat	
	(S)	=	Schweinefleisch	–	Suppe an den Ausgaben erhaeltlich		
	(R)	=	Rindfleisch	–	Dessert an der Desserttheke a DM 0,40		
	(V)	=	Vegetarisch	–	Aenderungen vorbehalten		
	*	=	Phosphat	–	Wir verwenden Jodsalz		

Time spent for evaluation: 5.720320 milliseconds
Thomas Kieninger

Figure 12. WWW interface to online demo.

The text area within the form allows you to paste an arbitrary ASCII text. If *Java Script* is supported and enabled by your browser, you will be able to display some characteristic examples in the text area by simply clicking on any of the *Demo* buttons below. The text appearing in the input field can optionally be modified. The textfield can also be cleared with the *Empty Field* button to allow the input of some completely new text.

Before submitting you can select whether to get the final results presented as a HTML document in a separate window or whether you would rather like to get all intermediate results as *PostScript* file (the default). In the latter case it is recommended to have a *PostScript* interpreter (like *ghostscript* or *ghostview*) configured to display documents which are characterized by the MIME type `application/postscript`.

The *analyze document* button will submit the currently displayed text to our http server, where *T-Recs* performs the complete structure analysis. The builtin ASCII preprocessor thereby filters ruled lines as they sometimes occur within ASCII texts (see online-demo 5 for an example) and passes the coordinates of these lines as meta information to the main process.

We are currently expanding our online demo with a file upload option and *T-Recs* will be enabled to parse the *XDOC* format of the XEROX OCR engine *ScanWorX*.[15] It will thus be possible for the public to test *T-Recs* with own OCR documents.

7. CONCLUSION AND OUTLOOK

With *T-Recs* we have built an extremely robust and very fast system that already proved to be of interest for industrial partners. It is most characterized by its initial clustering and the simple but effective classification of blocks during the postprocessing phase. Nevertheless we see a lot of more features to be included to *T-Recs*:

Thus we are aiming towards three main goals: first we want to further improve our results by involving even more document inherent features in the restructuring processes. In some quite unnaturally constructed constellations we recognized a block segmentation that differed from the man made alternative.

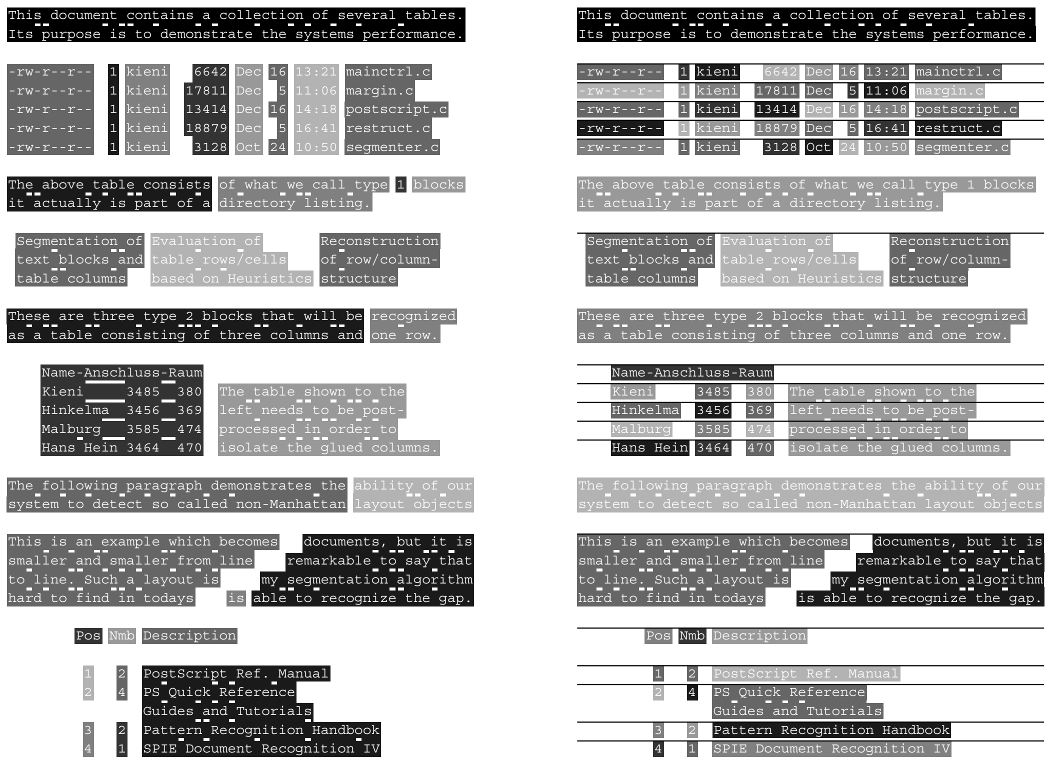
The second improvement should affect the structure analysis of the segmented blocks. When dealing with any kind of tables, our approach does a good work, but it evaluates incorrect structures on the head of business letter images: since some logical items like e.g. recipient address or date are arranged in their horizontal neighborhood, our current system interprets these fields as cells of a table. Here we have to consider more specialized design rules to decide whether or not we are facing a tabular environment. A rule-based bottom-up approach for the analysis of tables as described by Rahgozar and Cooperman[16] or a more general approach for the recognition of general structures as stated by Tao Hu[2] might be considerable extensions.

Finally we want to develop techniques for an objective benchmarking of the results. This is a new research field in the context of table recognition and hence there are many open questions. The major tasks are the preparation of the ground truth on a sufficiently large document base and the development of a user interface tool to gather the appropriate data. Another topic in this context is the definition of characteristic values that point out an overall quality of the results. The developed classification methods need to be fair, generally acceptable and applicable.

Together with the above mentioned tool for the acquisition of ground truth data we can think of user interface tools to postprocess the systems proposed structures. Such a system could be seen as an authoring tool that helps to bridge from paper to any wordprocessor.

APPENDIX A. EXAMPLES

The first example (seen in figure 13) shows a collection of word constellations as they are problematic for the initial algorithm. Whereas the left side shows the initial segmenation as derived from the clustering algorithm, the right side shows the final result after applying all postprocessing steps.



```
This document contains a collection of several tables.
Its purpose is to demonstrate the systems performance.

-rw-r--r--  1 kieni   6642 Dec 16 13:21 mainctrl.c
-rw-r--r--  1 kieni  17811 Dec  5 11:06 margin.c
-rw-r--r--  1 kieni  13414 Dec 16 14:18 postscript.c
-rw-r--r--  1 kieni  18879 Dec  5 16:41 restruct.c
-rw-r--r--  1 kieni   3128 Oct 24 10:50 segmenter.c

The above table consists of what we call type 1 blocks
it actually is part of a directory listing.

 Segmentation of Evaluation of       Reconstruction
 text blocks and table rows/cells    of row/column-
 table columns   based on Heuristics structure

These are three type 2 blocks that will be recognized
as a table consisting of three columns and one row.

    Name-Anschluss-Raum
    Kieni     3485 380  The table shown to the
    Hinkelma  3456 369  left needs to be post-
    Malburg   3585 474  processed in order to
    Hans Hein 3464 470  isolate the glued columns.

The following paragraph demonstrates the ability of our
system to detect so called non-Manhattan layout objects

This is an example which becomes   documents, but it is
smaller and smaller from line    remarkable to say that
to line. Such a layout is      my segmentation algorithm
hard to find in todays      is able to recognize the gap.

         Pos Nmb Description

          1   2  PostScript Ref. Manual
          2   4  PS Quick Reference
                 Guides and Tutorials
          3   2  Pattern Recognition Handbook
          4   1  SPIE Document Recognition IV
```

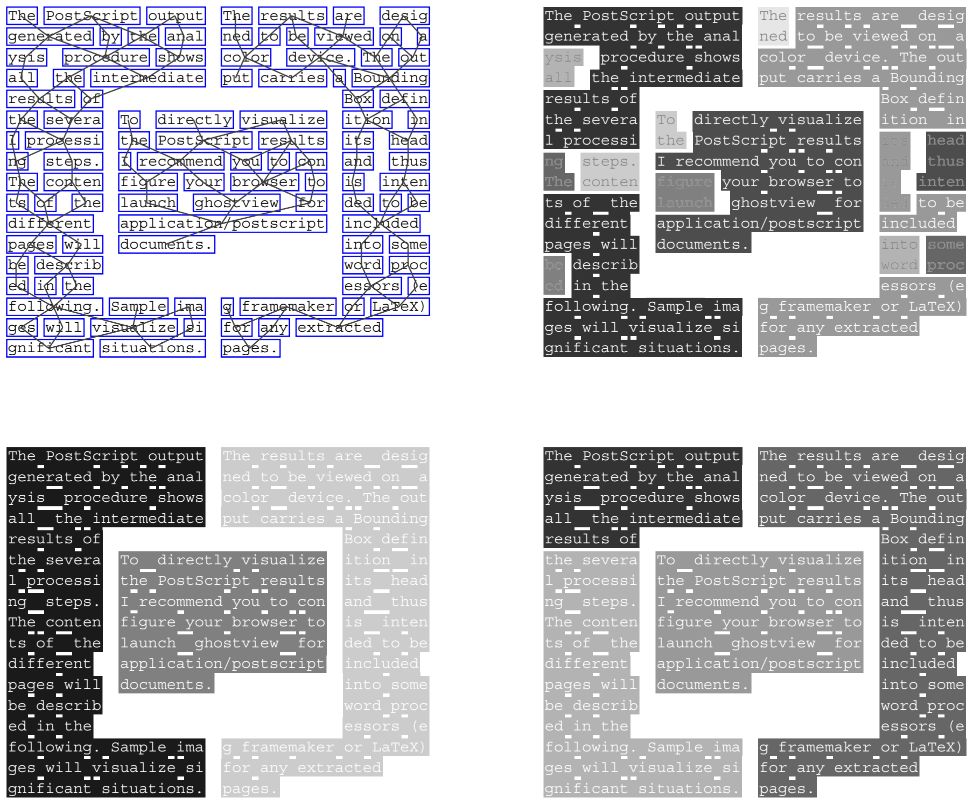
```
This document contains a collection of several tables.
Its purpose is to demonstrate the systems performance.

-rw-r--r--  1 kieni   6642 Dec 16 13:21 mainctrl.c
-rw-r--r--  1 kieni  17811 Dec  5 11:06 margin.c
-rw-r--r--  1 kieni  13414 Dec 16 14:18 postscript.c
-rw-r--r--  1 kieni  18879 Dec  5 16:41 restruct.c
-rw-r--r--  1 kieni   3128 Oct 24 10:50 segmenter.c

The above table consists of what we call type 1 blocks
it actually is part of a directory listing.

 Segmentation of Evaluation of       Reconstruction
 text blocks and table rows/cells    of row/column-
 table columns   based on Heuristics structure

These are three type 2 blocks that will be recognized
as a table consisting of three columns and one row.

    Name-Anschluss-Raum
    Kieni     3485 380  The table shown to the
    Hinkelma  3456 369  left needs to be post-
    Malburg   3585 474  processed in order to
    Hans Hein 3464 470  isolate the glued columns.

The following paragraph demonstrates the ability of our
system to detect so called non-Manhattan layout objects

This is an example which becomes   documents, but it is
smaller and smaller from line    remarkable to say that
to line. Such a layout is      my segmentation algorithm
hard to find in todays      is able to recognize the gap.

         Pos Nmb Description

          1   2  PostScript Ref. Manual
          2   4  PS Quick Reference
                 Guides and Tutorials
          3   2  Pattern Recognition Handbook
          4   1  SPIE Document Recognition IV
```

Figure 13. Block segmentation after initial clustering (left) and the final result (right)

The example summarizes the following features of the restructuring steps: merging type 2 blocks that are isolated by a gap; isolation of columns that are conjoined by a common header; identification of isolated words outside of matching table columns; fragmentation of meta level blocks (type 1 columns and appropriate type 2 blocks stretching over borders of logical rows). The image on the right side also shows horizontal rules that indicate the borders of logical table rows.

A.1. Applicability To Non-Manhattan Layout Documents

The example shown in figure 14 consists of three blocks, two of them having a non-rectangular shape and thus beeing an example for the *non-Manhattan* layout as mentioned in section 2.1. The two "outside" blocks build a kind of shell in which the third block is embedded. This sample will hardly be called a table and hence it is neglectable that our overall system stops with an incorrect segmentation. In fact, the error only arises during the fragmentation of type 2 blocks as described in section 4.1 – a procedure that is highly specialized for tabular environments. A more intelligent analysis of the structures at this stage might extend the system from recognizing table settings only and would probably allow a more comprehensive structure analysis. But it is most mentionable that *T-Recs* is able to isolate these textual entities, a result that would not have been achieved by any of the projection based approaches (like e.g. the one described by Itonori[9]).



```
The PostScript output  The results are desig
generated by the anal  ned to be viewed on a
ysis procedure shows   color device. The out
all the intermediate   put carries a Bounding
results of                        Box defin
the severa  To directly visualize ition in
l processi  the PostScript results its head
ng steps.   I recommend you to con and thus
The conten  figure your browser to is inten
ts of the   launch ghostview for  ded to be
different   application/postscript included
pages will  documents.            into some
be describ                        word proc
ed in the                         essors (e
following. Sample ima  g framemaker or LaTeX)
ges will visualize si  for any extracted
gnificant situations.  pages.
```

Figure 14. Intermediate results and final segmentation of a *non-Manhattan* layout example

The different parts of the figure show the following steps: the top left visualizes the input together with the segmentation graph as built by the initial clustering process. The top right shows the only mentionable intermediate result in this example – the temporarily isolated words that occur in a vertical *one-to-one* relation. The bottom left shows the best result (in this case identical to the initial clustering) which unfortunately is not the final result as seen on the bottom right: the splitting of the left block into two parts is caused by the postprocessing explained in section 4.1.

APPENDIX B. IMPLEMENTATION CHARACTERISTICS

T-Recs is completely implemented in *C*. The sources total to about 7500 lines of code (approx. 186 kByte) including the ASCII preprocessor. The runtime for the example of figure 13 took about 8 milliseconds on a Sparc Ultra 2; the *non-Manhattan* layout example took about 4 milliseconds. The runtime will moreover always be appended to the HTML output of the online demo. It does not include the image- nor the ASCII preprocessing. The time complexity of *T-Recs* is linear to the number of words – $O(n)$.

REFERENCES

1. A. Dengel, "About the logical partitioning of document images," in *Proc. of Int'l Symposium on Document Analysis and Information Retrieval, Las Vegas, Nevada*, Apr. 1994.
2. T. Hu, *New Methods for Robust and Efficient Recognition of the Logical Structures in Documents.* PhD thesis, Institute of Informatics of the University of Fribourg, Switzerland, 1994.
3. A. S. Condit, "Autotag: A tool for creating structured document collections from printed materials," Master's thesis, Dept. of Computer Science, University of Nevada, Las Vegas, 1995.
4. S. Baumann, M. Malburg, H.-G. Hein, R. Hoch, T. Kieninger, and N. Kuhn, "Document analysis at DFKI, part 2: Information extraction," DFKI Research Report RR-95-03, German Research Center for Artificial Intelligence (DFKI), Kaiserslautern, March 1995.
5. D. Rus and K. Summers, "Using white space for automated document structuring," Technical Report TR 94 - 1452, Department of Computer Science, Cornell University, 1994.
6. M. A. Rahgozar, Z. Fan, and E. V. Rainero, "Tabular document recognition," in *Proc. of the SPIE Conference on Document Recognition*, 1994.
7. S. Tupaj, Z. Shi, and D. C. H. Chang, "Extracting tabular information from text files." Available at http://www.ee.tufts.edu/ hchang/paper1.ps, 1996.
8. E. Green and M. Krishnamoorthy, "Recognition of tables using table grammars," in *Proc. of the 4-th Symposium on Document Analysis and Information Retrieval - SDAIR95, Las Vegas, Nevada*, 1995.
9. K. Itonori, "Table structure recognition based on textblock arrangement and ruled line position," in *Proc. of International Conference on Document Analysis and Recognition - ICDAR 93*, 1993.
10. Y. Hirayama, "A method for table structure analysis using dp matching," in *Proc. of International Conference on Document Analysis and Recognition - ICDAR 95, Montreal, Canada*, 1995.
11. S. Chandran and R. Kasturi, "Structural recognition of tabulated data," in *Proc. of International Conference on Document Analysis and Recognition - ICDAR 93*, 1993.
12. L. O'Gorman, "The document spectrum for bottom-up page layout analysis," in *Advances in Structural and Syntactic Pattern Recognition*, H. Bunke, ed., pp. 270 – 279, World Scientific, 1992.
13. K. Kise, A. Sato, and K. Matsumoto, "Document image segmentation as selection of voronoi edges," in *Proc. of IEEE Computer Society Conference on Computer Vision and Pattern Recognition CVPR 97*, June 1997.
14. The W3 Consortium, http://www.w3.org/CGI/, *CGI: Common Gateway Interface.* Starting point to lots of information concerning CGI programming.
15. Xerox Corp., Peabody, MA, USA, *XDOC Data Format: Technical Specification, Version 3.0*, Mar. 1995.
16. M. A. Rahgozar and R. Cooperman, "A graph-based table recognition system," in *Proc. of the third SPIE Conference on Document Recognition, San Jose, California*, Jan. 1996.