The T-Recs Table Recognition and Analysis System

Thomas Kieninger and Andreas Dengel

DFKI-GmbH, Postfach 2080, 67608 Kaiserslautern, FRG,
Thomas.Kieninger@dfki.de, Andreas.Dengel@dfki.de,
WWW page: http://www.dfki.uni-kl.de/~kieni/t_recs/

Abstract. This paper presents a new approach to table structure recognition as well as to layout analysis. The discussed recognition process differs significantly from existing approaches as it realizes a bottom-up clustering of given word segments, whereas conventional table structure recognizers all rely on the detection of some separators such as delineation or significant white space to analyze a page from the top-down.
The following analysis of the recognized layout elements is based on the construction of a tile structure and detects row- and/or column spanning cells as well as sparse tables with a high degree of confidence.
The overall system is completely domain independent, optionally neglects textual contents and can thus be applied to arbitrary mixed-mode documents (with or without tables) of any language and even operates on low quality OCR documents (e.g. facsimiles).

1 Introduction

Document structure analysis is emerging as a key technology for enabling the intelligent treatment of information and provides the key for an efficient handling of documents. The information inherent to the layout of tables is even more important, since their relational character requires different analysis techniques in the context of document understanding.

This paper presents the approach of *T-Recs*, a system that deals with the identification of tables within arbitrary documents, the isolation of individual table cells and the analysis of the layout to determine a correct row/column mapping.

We start with the initial block clustering – the central idea. Then we point out required postprocessing steps to correct some system inherent errors. While so far we focused on the segmentation, Sect. 4 sketches the analysis of the table cell layout.

1.1 The *T-Recs* Document Model

The *T-Recs* document model has 3 different hierarchically organized structure types and one non-hierarchically embedded auxiliary structure for text lines:

Words are our elementary objects. Their bounding box geometry and optionally their textual contents constitute the input of the system. Formally, a word is described as a triple $W = (T, G, A)$, where T keeps the textual contents, G denotes the bounding box geometry, specified by the quadruple $G = (x_0, y_0, x_1, y_1)$ and A holds the recognized font attributes.

Lines (also referred to as *text lines*) are an initially built aggregation of words that serves as *auxiliary structure* for all following procedures!

They are described as quadruples $L = (W_0, succ, G, A)$, specifying a sorted list of words (with W_0 naming the first word and *succ* the appropriate successor function), the bounding box G and the attributes $A = (linenumber, rownumber, spc\_len)$ that hold the unique *linenumber*, the *logical rownumber* (see Sect. 3.5) and average space width.

Blocks are dynamic aggregations of words. Note that *blocks are not aggregations of lines*! The initial *word-to-block mapping* is made by the central clustering algorithm but is changed by the various postprocessing steps. Blocks thus keep the main segmentation information.

Like lines, blocks are described as quadruple $B = (W_0, succ, G, A)$. The additional block attributes $A = (type, justification, height, nmbwords)$ describe the classification into *type 1* and *2* (see Sect. 3.2), the justification, height (as number of lines) and number of words in the block.

Document The document is defined as quadruple $D = (B_0, succ, G, A)$, where B_0 and *succ* specify a *sorted list of blocks*. The successor function $succ_{doc}(s)$ is defined for *words*, *lines* and *blocks*. The attributes $A = (l\_spc)$ contain the average linespacing.

Significantly large distances between adjacent lines cause the construction of *dummy lines* between them. This prevents the $succ_{doc}(line)$ function from bridging large line spacings as between paragraphs.

In our notation, objects of higher level instances are denoted in a functional way: $block(w_x)$ stands for the associated block of word w_x. The attributes of an object are denoted similarly but with brackets instead of parentheses around the argument. Thus, $fontsize[w_x]$ would describe the *fontsize* attribute of the word w_x. The bounding box geometry $G = (x_0, y_0, x_1, y_1)$ itself is specified by the upper-left and lower-right corner coordinates respectively.

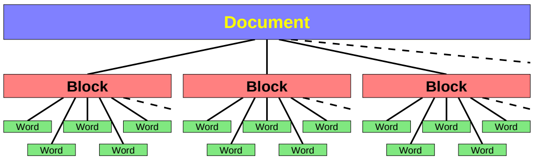

Fig. 1. Document model of the *T-Recs* System (hierarchical part)

Omiting the subordinated textline structure we achieve the hierarchical document model seen in Fig. 1, consisting of *words*, *blocks* and the *document* itself (in bottom-up order). Each object is part of a higher level instance.

2 Segmentation

2.1 Top-Down vs. Bottom-Up

Structure recognition itself is a wide research area and numerous people are addressing this topic. Existing approaches can be divided into systems which are specialized to detect logical objects in a restricted domain such as business letters as described in Dengel [3] or systems which try to identify more general structure elements like paragraphs, headers or lists. Hu [7] and Condit [2] both describe systems of that class. But only a small part of such systems consider the specific problems of tabular structures. They rather focus on objects like *paragraphs*, *headers* and *lists* in the general case or *sender*, *recipient*, *date* and *body* in case of business letters.

The investigation of existing table structure recognition approaches discloses a significant similarity: segmentation can always be characterized as a *top-down* approach that is driven by the detection of separators. Rus and Summers [16] present a system for the segmentation and labeling of general structures, also considering tables where they are able to detect narrow columns, using so-called *White Space Density Graphs* (WDG). Some other approaches relying on sufficiently large white spaces are described by Rahgozar et al. [13] who operates on word segments rather than on the bitmap, Tupaj et al. [18] who takes plain text output of OCR systems as input and Spitz [17] who determines so-called horizontal and vertical *rivers of white space* to define the boundaries of the islands of print material. Others are explicitly looking for ruling lines that determine the table structure. Representatives of this class are Green and Krishnamoorthy [5], who apply a grammar-based analysis on the set of occuring lines to evaluate the table layout, Itonori [8], who only considers the labeling aspects and expects well segmented blocks as input, or Hirayama [6] with his interesting *DP matching method*. Chandran and Kasturi [1] consider both (ruled lines and so-called *white streams*) to determine the layout structure.

T-Recs differs from conventional table segmentation systems as it represents a *bottom-up* approach. The central idea of *T-Recs* 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. The motivation for this unconventional strategy was twofold:

While typical top-down systems rely on the detection of some evidence to separate layout regions from each other, appropriate table segmentation systems concentrate on either characteristic delineation or conspicuous white spaces. This leads to a somewhat limited applicability of those systems.

The second reason for our design was driven by the insight that a human's way of recognizing a tabular structure within a document is based on the word-segments themselves. When looking at a table, a human never tries to identify textlines which he then divides into fragments of columns nor does he search for a given delineation to decompose layout regions. He rather realizes the blocks and columns directly as aggregations of words.

2.2 The *T-Recs* Bottom-Up Clustering Approach

Elementary to bottom-up approaches is the fact that some ordered elements (words) are aggregated to higher level instances (blocks). Since other bottom-up clustering systems either rely on nearest neighbors [4], run-length smoothing [19] or on the construction of Voronoi diagrams as block separators [12] [10] and thus localize adjacent elements to *all directions*, they will not be able to detect narrow column gaps. In contrast, *T-Recs* limits its search to neighboring words in the previous and next line (relative to the currently inspected item) whose bounding-boxes *horizontally overlap* with the inspected word. This symmetrical, binary relation $ovl(w_1, w_2)$ is given if the projections of the bounding boxes of two words w_1 and w_2 to the x-axis have a common range and if the words are located in subsequent lines:

$$\begin{aligned} ovl(w_1, w_2) \;\Leftrightarrow\; & (x_1[w_1] \geq x_0[w_2]) \wedge (x_0[w_1] \leq x_1[w_2]) \;\; \wedge \\ & (\, line(w_1) = succ_{doc}(line(w_2) \quad \vee \\ & \;\; line(w_2) = succ_{doc}(line(w_1)\,) \end{aligned}$$

Figure 2 shows a sample block and the area that contains potential overlapping words (gray stripe over the initial word "consists"). The words (or bounding boxes) that are "touched" by this virtual stripe will be clustered to the same block as the initial word.

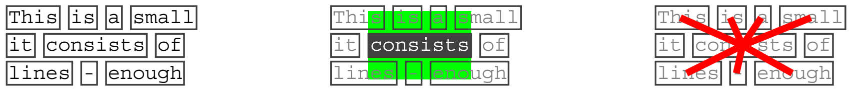

Fig. 2. Vertical neighbors of the word "consists"

Based on this relation, the clustering works as follows:

1. Find an *unexpanded* word $w_x := w_0$ (the *seed*) and create block b_i;
2. Mark *current word* w_x as *expanded* and add it to b_i;
3. Evaluate all *unexpanded* words w_j in $ovl(w_x, w_j)$;
4. For all w_j, make it the *current word* w_x and perform steps 2, 3 and 4;
5. If no more matching words are found, increment i and go to step 1;
6. Stop, if all words are marked as *expanded*.

Blocks are constructed "around" the *block seed* w_0 and can thus be described as transitive hull ovl^* of the overlapping relation. We also write $w_x \sim_{ovl^*} w_y$ if the words belong to the same cluster and define:

$$\begin{array}{rll} \mathcal{M} & := \{w_i\} & \text{the set of all words;} \\ [w_0]_{ovl^*} & := \{w_x | w_x \in \mathcal{M} \ \wedge \ w_0 \sim_{ovl^*} w_x\} & \text{the cluster of } w_0\text{;} \\ \mathcal{M}|ovl^* & := \{[w_x]_{ovl^*} | w_x \in \mathcal{M}\} & \text{set of all clusters.} \end{array}$$

Since the *ovl* relation is symmetrical, and hence $\sim_{ovl^*}$ is an equivalence relation, it is obvious, that the choice of the block seed has no effect on the clustering result: $w_x \sim_{ovl^*} w_y \ \rightarrow \ [w_x]_{ovl^*} = [w_y]_{ovl^*}$

We use the so called *segmentation graph* (Fig. 2, right) to visualize the clustering. The nodes of that graph are the centerpoints of the bounding boxes and an edge between two nodes indicates the overlapping relation between the appropriate words.

Advantages of the *T-Recs* Approach The strengths of *T-Recs* are not directly visible in the context of the above example but at least it is obvious that *T-Recs* is capable of recognizing and clustering regular blocks. It is moreover clear, that isolated blocks will be recognized as isolated elements since their elements do not interleave mutually. An example of a tabular environment with a dense column arrangement is seen in Fig. 3.

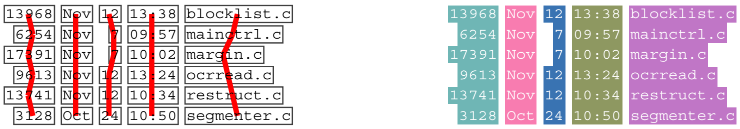
13968 Nov 12 13:38 blocklist.c
6254 Nov 7 09:57 mainctrl.c
17391 Nov 7 10:02 margin.c
9613 Nov 12 13:24 ocrread.c
13741 Nov 12 10:34 restruct.c
3128 Oct 24 10:50 segmenter.c
13968 Nov 12 13:38 blocklist.c
6254 Nov 7 09:57 mainctrl.c
17391 Nov 7 10:02 margin.c
9613 Nov 12 13:24 ocrread.c
13741 Nov 12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

Fig. 3. Segmentation of a tabular environment

A first advantage of this approach compared to top-down systems is the independence of delineations or conspicuous white spaces. We can thus accept any documents regardless of the occurence of the above features.

A second advantage is the independence of accurate vertical or horizontal cuts between block. *T-Recs* is able to segment blocks whose rectangular block bounding boxes intersect. In other words: the block layout is not limited to rectangular shapes. Here we also speak of *Non-Manhattan Layout* [11].

System Inherent Errors Although the results look good on a large collection of documents, we recognize that there are some segmentation errors that are inherent to this approach. Aiming towards the identification of significant features of all mis-segmentations to provide appropriate corrections algorithms, we were able to develop a series of specialized procedures.

Throughout this document we will use Fig. 4 (left) as reference to give examples of the different classes of errors but also for discussing appropriate postprocessing procedures. The error classes can be described as follows:

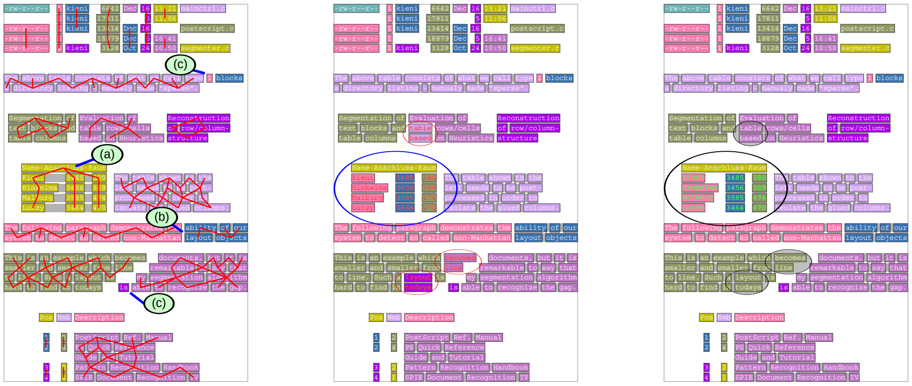

Fig. 4. Initial clustering, intermediate and remaining "split sons"

- Columns that are merged together by a common header which consistently overlaps with the first word of each column (Fig. 4, left: a).
- Blocks that are split into parts by an occasional gap at the same X-position throughout the block (so called *rivers*) (Fig. 4, left: b).
- Words that have neither an upper nor a lower neighbor will remain isolated. Each word of a header will for instance produce an individual, isolated block (Fig. 4, left: c).

The following paragraph discusses ways to escape these inherent errors. Each error-class will have its own postprocessing procedure.

3 Postprocessing Steps

3.1 Isolation of Merged Columns

The first problem that we want to consider are column blocks that are joined together by a common header which spans all the columns. In this case we need to identify the subcolumns. Looking more closely at the segmentation graph, we recognize a significant feature: the words of subcolumns all occur in a *one-to-one* relation. This relation $one_{one}(w_x, w_y)$ is given between two words, if word w_x has exactly one lower overlapping neighbor w_y and w_y has exactly this one upper neighbor w_x. Sequences of such words (as typically found in subcolumns) are especially visually conspicuous. We need to introduce some new functions in order to express this relation formally:

$$\begin{aligned} w_y = ovl^{first}(w_x) &\iff ovl(w_x, w_y) \;\wedge \\ &\qquad \neg\exists w_z : \; (w_y = succ_{line}(w_z) \;\wedge\; ovl(w_x, w_z)) \\ w_y = ovl^{first}_{up}(w_x) &\iff w_y = ovl^{first}(w_x) \;\wedge\; line(w_x) = succ_{doc}(line(w_y)) \end{aligned}$$

$$w_y = ovl^{first}_{down}(w_x) \Longleftrightarrow w_y = ovl^{first}(w_x) \ \wedge \ line(w_y) = succ_{doc}(line(w_x))$$

Using the above functions, we can define the *one-to-one* relation as follows:

$$\begin{aligned} one_{one}(w_x, w_y) \Longleftrightarrow & \ (\ w_y = ovl^{first}_{down}(w_x) \ \wedge \ \neg ovl(w_x, succ_{line}(w_y)) \ \wedge \\ & \ w_x = ovl^{first}_{up}(w_y) \ \wedge \ \neg ovl(w_y, succ_{line}(w_x)) \ ) \ \vee \\ & \ one_{one}(w_y, w_x) \qquad \text{(this line ensures symmetry)} \end{aligned}$$

We now separate all *sequences of words* standing in such a one-to-one relation and moreover call them *Split Sons* ($\mathcal{B}^{split}$). No other parametrical limits are applied at that point. The set of words for each of these newly created blocks is characterized as the transitive hull of the one_{one} relation :

$$\mathcal{B}^{split}_{w_x} \ := \ [w_x]_{{one_{one}}^*} \ := \ \{w_y | w_y \in \mathcal{M} \ \wedge \ w_x \sim_{{one_{one}}^*} w_y\}$$

The result of this step applied to our reference document is seen in Fig. 4 (center). The interesting regions are marked with ovals.

As expected, this negligent isolation step results in some $\mathcal{B}^{split}$ blocks that are incorrect (small ovals). But the value of each isolated block can best be rated afterwards because the most significant factor for being a proper subcolumn is given by the surrounding columns. As we know, table columns typically occur only in the neighborhood of other columns. The basis for the split sons to remain isolated is calculated on the following features: number of directly adjacent columns; average amount of space to the left and right neighbor; block height; textual or structural similarity of the words (optional).

If the basis of a $\mathcal{B}^{split}$ block does not reach a given threshold, it will be remerged with its father-block. The result of this remerging operation is seen in Fig. 4 (right). The shaded ovals point to the “repaired” blocks.

3.2 Elimination of “Rivers”

In some cases a regular block might show some white space at the same x-position throughout the complete block. These so called *rivers of whitespace* are said to be *bad layout* and are tried to be avoided by modern typesetting programs and wordprocessors. They are more likely to occur in small blocks of only a few lines, using fixed width fonts (e.g. ASCII texts).

Applying the simple heuristics of merging adjacent blocks if the average gap size between the words at the border is not significantly bigger than one space, would end in the loss of isolated table columns with narrow gaps as well. To avoid this error, the merging operation must be applied selectively.

A closer look at the blocks which would be affected points out a very distinctive feature: those table column blocks that we want to prevent from the remerging are all characterized by having at most one word (or token) per line.

We thus classify all blocks into two types: the typical column blocks with one word per line are classified as *type 1*. All others are of *type 2*.

The actual postprocessing step is quite straightforward: we simply merge all (and only!) adjacent type 2 blocks if the white space between the words of the adjacent blocks is not significantly bigger than one space [1]. The intermediate clustering after this merging operation is seen in Fig. 5 (left).

3.3 Clustering of Isolated Words

The third class of errors is caused by words which have neither upper nor lower neighbor and thus would not be clustered to any block. A header would for instance be interpreted as a table of one line height with each word representing one column. In general, words that have neither upper nor lower neighbor might either 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.

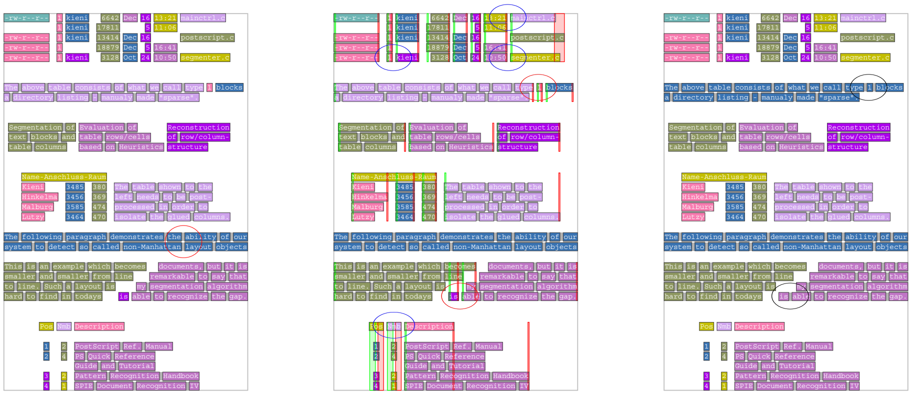

Fig. 5. Closed "Rivers", Reference points and clustered isolated words

We thus have to decide whether an isolated word fits into a table column or not. To gain a global view over all potential columns, we scan through all blocks and wherever a relatively narrow block or at least two blocks in a horizontal neighborhood occur, we assume to be inside a potential table environment and evaluate what we call the corresponding *margin structure* with its *margin points* MP and the *reference points* RP.

Therefore, we visit all blocks sequentially and if a block presumably belongs to a margin structure we evaluate the appropriate block cluster $\mathcal{B}^{b\_clst}$ and the list of *margin points*. These MPs indicate the left and right borders of blocks. An MP can represent more than one block if these blocks are aligned to the

[1] The threshold value used for this comparison is given as an external parameter.

appropriate side. The major attributes of an MP are the accumulated height of its inducing blocks and its type (left or right) which corresponds to the triggering block side and the x-position.

The *reference points* RP are then constructed based on a sorted list (by x position) of margin points. The RPs gather a sequence of MPs with the same type (not interrupted by different type MPs!) and within a threshold x-range. The RPs accumulate the height of the MPs in the *ref_counter* attribute.

The reference points of our sample document are shown in Fig. 5 (center): gray and black bars for the left and right RPs respectively. The dark ovals point towards isolated words that should be bound to their horizontal neighbor. The bright ovals point towards words that should remain isolated.

Thus, every block in a cluster $\mathcal{B}^{b\_clst}$ has one left and one right RP. If any of the appropriate RPs *ref_counter* of a block is smaller than a given limit, it is presumed **not to match** with a surrounding column and will occasionally be merged to surrounding neighbor blocks on the appropriate side.

The effect of this procedure is to select isolated words that do not fit into a surrounding table column for the merging operation. The resulting blocks are seen in Fig. 5 (right). The gray ovals point out the "repaired" parts.

3.4 Delineation Based Block Separation

The intuitive semantics of delineations is an explicit demarcation of the distinct text areas. In the rare case of very dense blocksetting, it might happen, that the initial clustering builds blocks over such delineations. Therefore we apply the following processing steps.

We call a block *strictly cut*, if all endpoints of an intersecting line (or touching lines) are outside of the block bounding box. If one end of a separator is outside and one is inside of the block bounding box (thereby not touching any other separator), we say it is *weakly cut*. In either case the block is subject to further analysis.

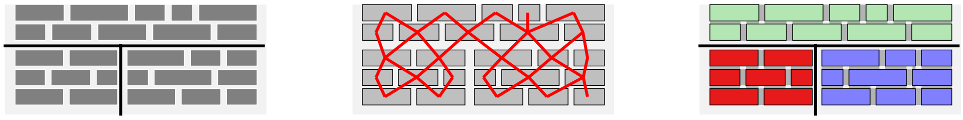

Fig. 6. Separator based splitting of a strictly cut block

Separation of Strictly Cut Blocks In this case, we simply have to decide for each word on which side relative to the separator it is positioned. The words are moved to new blocks accordingly. Figure 6 gives an example of three blocks that are isolated by two separators (left). Due to the dense arrangements of all words of these blocks, the initial clustering of *T-Recs* causes all words to be mapped to one block, as indicated by the segmentation graph (center). The above mentioned operation would achieve the final segmentation (right).

Separation of Weakly Cut Blocks Figures 7 and 8 both show examples of blocks which are weakly cut by a horizontal separator. As the block geometry itself gives no further discriminating information on how to proceed, we need to have a closer look at the words themselves when dealing with *weakly cut* blocks.

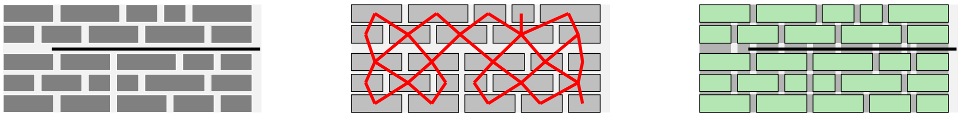

Fig. 7. Example of an unaffected weakly cut block

Even as a human reader we have no intuitive idea how to deal with the situation given in Figure 7. It is not clear, whether to break the block or not and if yes, what to do with the words that are not projected onto the separator. We decided to leave such blocks untouched.

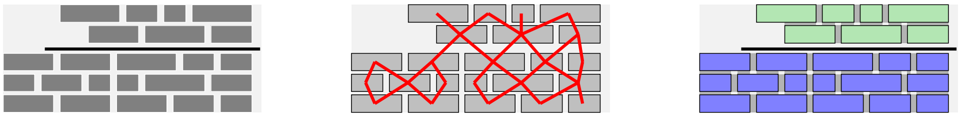

Fig. 8. Example of an affected weakly cut block

The example of Figure 8 is more intuitive: While one endpoint is inside the block bounding box, the decomposition based on the given separator looks nonetheless reasonable. This is due to the fact that the projections of the words on both sides of the separator to the extended line do not meet beside the separator itself. The upcoming operation is straightforward: The words are being moved to separate blocks according to their relative side.

3.5 Unification of Block Abstraction Level

Looking at the type 1 blocks (see Sect. 3.2) we realize that they all represent columns (or parts of a column), whereas type 2 blocks represent atomic textual units. To achieve a homogeneous view, we simply decompose all type 1 blocks into their individual line segments. The resulting structure is seen on the left of Fig. 9.

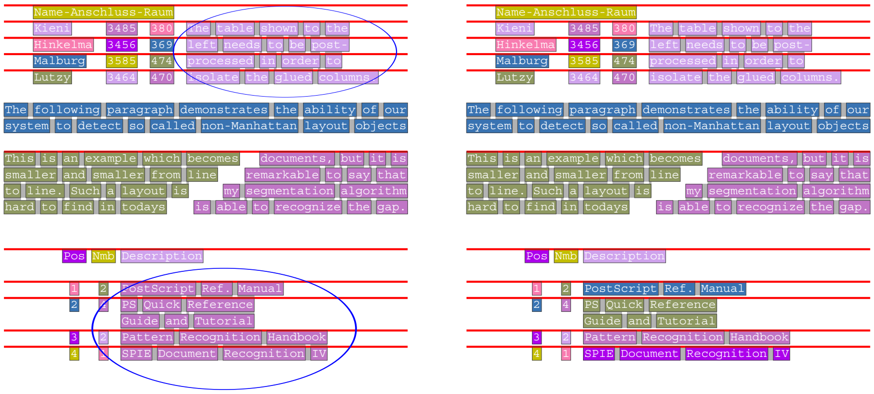
Name-Anschluss-Raum
Kieni 3485 380 The table shown to the
Hinkelma 3456 369 left needs to be post-
Malburg 3585 474 processed in order to
Lutzy 3464 470 isolate the glued columns.
The following paragraph demonstrates the ability of our
system to detect so called non-Manhattan layout objects
This is an 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
Guide and Tutorial
3 2 Pattern Recognition Handbook
4 1 SPIE Document Recognition IV

Fig. 9. Decomposed type 1 blocks and type 2 blocks split at row borders

The figure moreover shows some conspicuous black horizontal lines, called (logical) *row borders*. These are triggered by the top edges of those blocks that are standing inside a tabular environment. We might further observe two distinct type 2 blocks (indicated by the gray ovals) which are intersected by some row borders. If such a block is an aggregation of table cells, it will also be decomposed at the edges of the row borders.

To decide whether or not a type 2 block has to be treated like this, we consider that word processors try to fill the existing line space as much as possible. Thus, a word at the beginning of each line would not have fit on the end of the previous line. In this case we say that a block is *properly filled* and conclude that it contains no explicit returns and hence builds a logical entity. The resulting structure which represents the final segmentation result is seen on the right of Fig. 9.

We like to refer to [9] for a detailed description of the postprocessing steps discussed in Sects. 3 and 3.5 and to the online demo of the *T-Recs* system under `http://www.dfki.uni-kl.de/~kieni/t_recs/`

4 Table Layout Analysis

To ensure a common terminology, we need to specify some objects used in the context of tables. To point out the differences between the data *cells* and the table *tiles* we take an example with an empty field as well as a column- and a row spanning data cell as shown in Fig. 10. The block segments are represented by black outlined rectangles. The gray filled rectangles indicate the structures to be explained.

From left to right we see the original *data cells* represented by their bounding boxes. Next we see the *columns* followed by the *rows*. The very right shows the *table tiles*, a regular structure, defined by the combination of rows and columns. The overall structure is called the *table*.

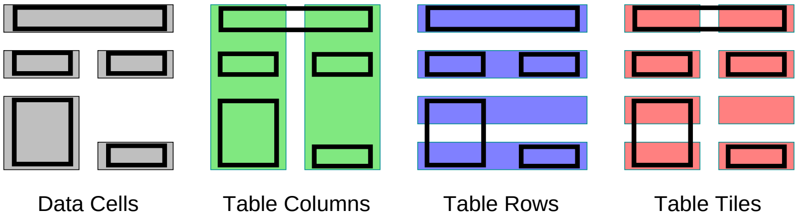

Fig. 10. Logical elements of a table.

After the initial clustering and error correction phases as well as the unification of the block abstraction level, we have a set of blocks that either represent regular paragraphs or table cells. We now need to identify table cell blocks and analyze their structural arrangement.

Wherever two or more blocks occur as horizontal neighbors (indicated by blocks with a dedicated margin structure), we assume a table and construct a structure that we call *table tiles*.

4.1 Construction of Table Tiles

Instead of rule based approaches that perform a bottom-up aggregation of higher level objects, *T-Recs* evaluates a grid of tiles which is fine enough so that each tile covers at most one table cell. Figure 11 (left) shows an example of some blocks (indicated by bright rectangles) and some horizontal and vertical block separators. The horizontal separators are identical to the row borders as described in Sect. 3.5. The vertical separators overlay the areas between right and left margin points as described in Sect. 3.3.

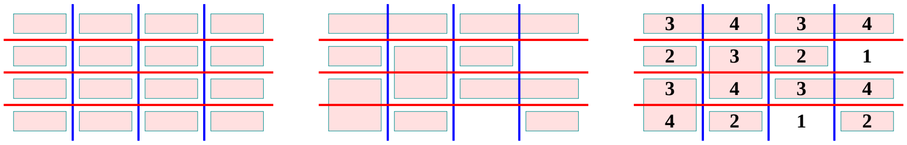

Fig. 11. Tiles of a regular and a degenerated table and appropriate tile classification

The tiles are defined by the grid that is made up of the horizontal and vertical separators and in the case of Fig. 11 (left) they are identical with the blocks of the cells themselves.

It is obvious, that each tile is covered by *at most one table cell* while on the other hand we *do not have any more tiles than needed* to achieve this condition. We can further state, that the tiles are identical to the table cell blocks if the table is dense and none of the cells spans more than one row and column.

In contrast, Fig. 11 (center) shows a degenerate table with various row or column spanning cells and even "missing" cells (i.e. the table is *sparse*). Nonetheless we get the same tile structure as in the example of Fig. 9, indicating 4 rows and 4 columns. Obviously, the tiles are no longer identical to the cells. Again, each tile is covered by at most one cell - but we also realize that some cells touch more than one tile and moreover we see that some tiles remain untouched.

4.2 Internal Structured Representation

The internal representation of the document structure consists of a sequence of regular paragraph blocks and blocks occuring as cells within a table. The latter blocks contain pointers to appropriate tile structures. The presence of tile structures causes the system to handle the appropriate blocks differently than other blocks, i.e. with a function that is designed for that particular purpose. First, we need to identify the different states of a tile:

1. A tile can be left blank (if a table is sparse).
2. A tile can be covered by a block that does not reach to adjacent tiles.
3. A tile can be the first (topmost and leftmost) in a sequence of tiles to be covered by one column and/or row spanning cell.
4. A tile can be the "non-first" in a sequence of tiles to be covered by a column and/or row spanning cell.

Figure 11 (right) shows the center example again, but with numbers in each tile, indicating the appropriate classification according to the numbering of the different states.

4.3 Generating Tagged Output of Tables

Since the current output of *T-Recs* is a HTML document, we briefly summarize the main HTML tags that are used for the definition of tables. All tags specify an environment which ends with the corresponding end-tag. The end-tags are indicated by a slash in front of the actual tag name. An overall table will be nested between a pair of `<TABLE>` and `</TABLE>` tags.

`<TABLE>`	Defines the start of a new table
`<TR>`	Defines begin of a new row of a table
`<TH>`	Defines begin of a header cell
`<TD>`	Defines begin of a data cell
`COLSPAN=`x	Optional Parameter for `<TD...>` and `<TH...>` that defines a cell to span x columns
`ROWSPAN=`y	Optional Parameter for `<TD...>` and `<TH...>` that defines a cell to span y rows

The preparation of the tagged output of tabular environments is done by traversing the tile structure from top to bottom, left to right. The actions performed for each tile are thereby controlled by the discussed state. The following abstract algorithm describes the individual operations:

1. If a new table starts, print "<TABLE>".
2. If current tile is the first one in a new row, print "<TR>".
3. If the status number of the current tile is 1, print "<TD></TD>" to define an empty field.
4. If the status number of the current tile is 2, print "<TD>" .
5. If the status number of the current tile is 3, print "<TD COLSPAN=x ROWSPAN=y>" where x and y denote the number of columns and rows respectively that are covered by the associated block.
6. Skip tiles with a status number of 4.
7. If the status number of the current tile is 2 or 3, print the textual contents of the associated block and the end tag "</TD>".
8. If current tile is the last one the current row (regardless of the state), print "</TR>".
9. If the current table ends, print "</TABLE>"
   else move to next tile and goto step 2.

5 Conclusion and Outlook

While classical character recognition systems do not show recent significant improvements [14] [15], commercial OCR systems focus more and more on the detection of structural information (such as tables) as key technology for their products.

Since benchmarking systems have not yet been developed for tabular structures, we cannot give quantitative statements about *T-Recs* results. But the demonstrated interest of OCR vendors in the *T-Recs* technology might count as proof for being on the right track.

Both of the *T-Recs* subsystems, segmentation and layout analysis, are still subject to further research with the goal of an improved performance. We will for instance implement new heuristics to avoid the misinterpretation of layout objects in business letters as tables.

In order to allow an objective benchmarking of the recognition accuracy we are currently developing a graphical user interface to gather ground truth data. This frontend also allows us to manipulate the layout of given documents while keeping track of the document logic. Thus, it is possible to construct large collections of ground truth data in a ready-to-use format for the actual application (no printing, scanning or OCR process neccessary) which is moreover free of any unwanted noise.

Benchmarking not only allows us to compare different analysis systems but also to document system progress and to optimize predefined system parameters.

References

[1] Surekha Chandran and Rangachar Kasturi: Structural Recognition of Tabulated Data. In *Proc. of International Conference on Document Analysis and Recognition - ICDAR 93*, 1993.

[2] Allen 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.

[3] Andreas Dengel: About the Logical Partitioning of Document Images. In *Proceedings SDAIR-94, Int'l Symposium on Document Analysis and Information Retrieval, Las Vegas, NV*, pages 209–218, April 1994.

[4] Lawrence O'Gorman: The Document Spectrum for Bottom-Up Page Layout Analysis. In H. Bunke, editor, *Advances in Structural and Syntactic Pattern Recognition*, pages 270 – 279. World Scientific, 1992.

[5] 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.

[6] Yuki 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.

[7] Tao 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.

[8] Katsuhiko 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.

[9] Thomas Kieninger: The *T-Recs* Table Converting System. available at http://www.dfki.uni-kl.de/~kieni/doc/trecs3.ps.gz, April 1998.

[10] Koich Kise, Akinori Sato, and Keinosuke 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.

[11] George Nagy and S. Seth: Hierarchical Representation of Optically Scanned Documents. In *Proc. of the 7th Intl. Conference on Pattern Recognition (ICPR)*, 1984.

[12] T. Ohya, M. Iri, and K. Murota: A fast Voronoi Diagram Algorithm with Quaternary Tree Bucketing. In *Information Processing Letters, Vol. 18, No. 4*, 1984.

[13] M. Armon Rahgozar, Zhigang Fan, and Emil V. Rainero: Tabular Document Recognition. In *Proc. of the SPIE Conference on Document Recognition*, 1994.

[14] Stephen Rice, Frank Jenkins, and Thomas Nartker: The Fourth Annual Test of OCR Accuracy. Technical report, Information Science Research Institute (ISRI), Univ. of Nevada, Las Vegas, 1995.

[15] Stephen V. Rice, Frank R. Jenkins, and Thomas A. Nartker: The Fifth Annual Test of OCR Accuracy. Technical report, Information Science Research Institute (ISRI), Univ. of Nevada, Las Vegas, 1996.

[16] Daniela Rus and Kristen Summers: Using White Space for Automated Document Structuring. Technical Report TR 94 - 1452, Department of Computer Science, Cornell University, 1994.

[17] A. Lawrence Spitz: Recognition Processing for Multilingual Documents. In *Proceedings of the International Conference on Electronic Publishing, Document Manipulation & Typography*, Gaithersburg, Maryland, September 1990.

[18] Scott Tupaj, Zhongwen Shi, and Dr. C. Hwa Chang: Extracting Tabular Information from Text Files. Available at http://www.ee.tufts.edu/~hchang/paper1.ps, 1996.

[19] K. Y. Wong, R. G. Casey, and F. M. Wahl: Document Analysis System. *IBM Journal of Research & Developement, 1982*, 26(6):647–656, 1982.