Table Identification and Reconstruction in Spreadsheets

Elvis Koci[1,2(✉)], Maik Thiele[1], Oscar Romero[2], and Wolfgang Lehner[1]

[1] Database Technology Group, Department of Computer Science, Technische Universität Dresden, Dresden, Germany
{elvis.koci,maik.thiele,wolfgang.lehner}@tu-dresden.de
[2] Departament d'Enginyeria de Serveis i Sistemes d'Informació (ESSI), Universitat Politècnica de Catalunya-BarcelonaTech, Barcelona, Spain
{ekoci,oromero}@essi.upc.edu

Abstract. Spreadsheets are one of the most successful content generation tools, used in almost every enterprise to perform data transformation, visualization, and analysis. The high degree of freedom provided by these tools results in very complex sheets, intermingling the actual data with formatting, formulas, layout artifacts, and textual metadata. To unlock the wealth of data contained in spreadsheets, a human analyst will often have to understand and transform the data manually. To overcome this cumbersome process, we propose a framework that is able to automatically infer the structure and extract the data from these documents in a canonical form. In this paper, we describe our heuristics-based method for discovering tables in spreadsheets, given that each cell is classified as either header, attribute, metadata, data, or derived. Experimental results on a real-world dataset of 439 worksheets (858 tables) show that our approach is feasible and effectively identifies tables within partially structured spreadsheets.

Keywords: Speadsheet · Document · Tabular · Grid · Table · Layout · Recognition · Identification

1 Introduction

Spreadmarts, i.e. reporting or analysis systems running on desktop software, are used in more than 90% of all organizations [7]. 41% of these are built with Excel [7] which can be found on most office computers and, hence, do not incur any additional costs. Besides the low costs there are plenty of other reasons for using Excel as a data analysis tool, such as the high degree of autonomy, the fast information provisioning process compared to data warehouses, and the user desire to protect interests. While spreadmart solutions have their raison d'être, they come with the risk that information stored in them is getting lost since they are not part of the enterprise-wide administration. The problem of visibility is partly tackled by new information management principles such as *data lakes* [11,12] but the core problem still remains: how to extract and harvest

E. Dubois and K. Pohl (Eds.): CAiSE 2017, LNCS 10253, pp. 527–541, 2017.
DOI: 10.1007/978-3-319-59536-8_33

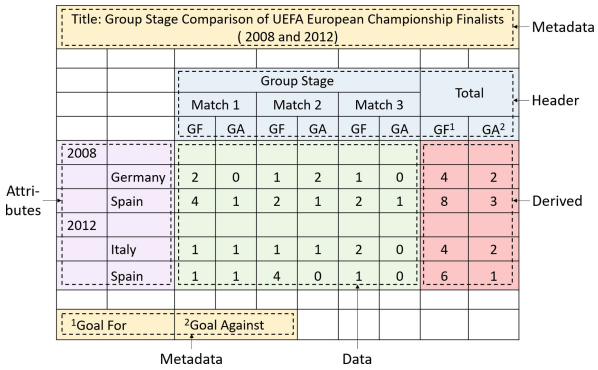


Fig. 1. Cell classification label [10]

the rich information found in spreadsheet formats enabling their reuse and thus fostering the understanding of data maintained in these files.

Our aim is to overcome this challenge by focusing our efforts on approaches for table identification and layout inference in spreadsheets. In a previous paper [10], we have proposed a machine learning approach for layout inference. We focused on the level of individual cells, considering a large number of features not covered by related work. From these, 43 were chosen for the final evaluation. The results show very high accuracy with all the defined classes (labels for the cells), and an overall 97% F1 measure.

Figure 1 provides examples for each of the cell labels. *Header* and *Data* are the basic building blocks of a table. In addition to this, we are using the notion of *Attributes*, i.e. specific data fields on the left of the table structured in a hierarchical way. *Derived* cells hold aggregations of data cells. Finally, *Metadata* cells provide additional information about the table as a whole (e.g., the title) or its parts (e.g., the unit of numeric values in a column). Additional information on these labels and the overall project can also be found on our website[1].

In this paper, we build upon these notions developing novel techniques to identify and reconstruct tables. Our approach takes as input the results from the cell classification task. Cells are then grouped to form regions (clusters) based on their label and location. These regions become the input for our heuristics framework, called TIRS (Table Identification and Reconstruction in Spreadsheets), which outputs tables and their layout. In the following sections we describe in detail each individual step of this process.

The subsequent parts of the paper are organized as follows: In Sect. 2, we define the concepts used throughout the proposed approach. The steps and heuristics for the table identification process are described in Sect. 3. In Sect. 4, we present the results of our evaluation. Finally, we review related work on table identification in Sect. 5, and conclude this paper with a short summary in Sect. 6.

[1] https://wwwdb.inf.tu-dresden.de/research-projects/deexcelarator.

2 Preliminaries

In the following sections, we define the concepts that are used in our heuristical framework, TIRS, and we discuss the pre-processing phase of our approach.

2.1 Cell Clusters

We decided to group cells based on the label they were assigned during the classification process and their location. We believe that these larger structures will help us streamline the table identification process. It is much simpler and intuitive to work on collections of cells rather than on individual cells. Furthermore, we have to handle a much smaller number of elements, thus decreasing the complexity of the overall process.

In the following paragraphs we provide formal definitions of the concepts used throughout the creation of the cell clusters. We start with the definition of the structure used to represent a sheet, which is referred to as worksheet in the Microsoft Excel environment.

Definition 1 (Worksheet Matrix). *A worksheet is represented by an m-by-n matrix of cells, denoted as* $\mathcal{W}$*. We refer to a cell in the matrix as* $\mathcal{W}_{i,j}$*.*

In this paper, we look at worksheets whose cells were previously classified by our method [10]. We assign a label to each non-empty cell.

Definition 2 (Classified Cell). *Is a cell in a worksheet, s.t.* $\texttt{Empty}(\mathcal{W}_{i,j}) \neq 1$ *and* $\texttt{Label}(\mathcal{W}_{i,j}) = \ell$*. Here, function* `Empty` *returns 1 for empty cells (i.e., without value), 0 otherwise. Function* `Label` *returns the label assigned to a classified cell, where* $\ell \in Labels, Labels = \{Data, Header, Attribute, Metadata, Derived\}$*.*

As we stated in the beginning of this section, our goal is to cluster cells. We initiate this process at the row level by grouping together consecutive cells of the same label. We refer to these mini row clusters as *Label Intervals*.

Definition 3 (Label Interval ($\mathcal{LI}$)). *A label interval is a submatrix of* $\mathcal{W}$*, denoted as* $\mathcal{W}[i; j, j']$*. For every cell* $\mathcal{W}_{i,j''}$ *in* $\mathcal{LI}$*, where* $j \leq j'' \leq j'$*, the* $\texttt{Label}(\mathcal{W}_{i,j''}) = \ell$*. To ensure maximal intervals, we enforce that* $\texttt{Label}(\mathcal{W}_{i,j-1}) \neq \ell$ *and* $\texttt{Label}(\mathcal{W}_{i,j'+1}) \neq \ell$*.*

We proceed further by grouping cells to even larger clusters, which we call *Label Regions*($\mathcal{LR}s$). Intuitively, $\mathcal{LR}s$ can be seen as an attempt to bring together $\mathcal{LI}s$ of the same label[2] from consecutive rows of $\mathcal{W}$. This is not straightforward, since the start column, end column, and order (size) can vary among these $\mathcal{LI}s$. Therefore, we target those $\mathcal{LI}s$ that are at least partially stacked vertically.

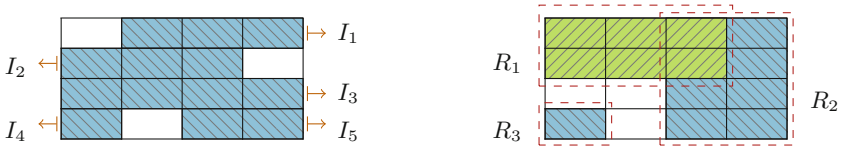
Definition 4 (Stacked $\mathcal{LI}s$). *Let* I *be the collection of all* $\mathcal{LI}s$ *in* $\mathcal{W}$*, then* I_k *and* $I_{k'}$ *are stacked iff there exists at least a pair* $(\mathcal{W}_{i,j}, \mathcal{W}_{i+1,j})$ *of cells s.t* $\mathcal{W}_{i,j}$ *in* I_k*,* $\mathcal{W}_{i+1,j}$ *in* $I_{k'}$*.*

[2] More specifically, all cells from these intervals have the same label.

We aim at constructing maximal $\mathcal{LR}s$, by merging all intervals of the same label that are vertically stacked.

Definition 5 (Label Region($\mathcal{LR}$)). *A label region is a $p \times q$ matrix of cells, where $1 \leq p \leq m$ and $1 \leq q \leq n$. In the trivial case a $\mathcal{LR}$ is made of a single $\mathcal{LI}$ (i.e., $\mathcal{LR}$ and $\mathcal{LI}$ are the same matrix). Otherwise, let r and $r+1$ be the indices of any two consecutive rows in $\mathcal{LR}$, where $1 \leq r < r+1 \leq p$. Then, there exists a pair of same-label stacked intervals (I_k,$I_{k'}$) that respectively correspond to stacked sub-matrices in row r and $r+1$ of $\mathcal{LR}$. An interval $I_{t''}$* ***is not*** *part of $\mathcal{LR}$ iff it has a different label or it has the same label but is not stacked with any of intervals in $\mathcal{LR}$.*

Note that a $\mathcal{LR}$ is not a submatrix of $\mathcal{W}$. They share $\mathcal{LI}s$, but the remaining parts of the $\mathcal{LR}$ might be different. We use empty cells to fill the gaps from the clustered $\mathcal{LI}s$, when necessary. In this way we ensure equally sized columns and equally sized rows for $\mathcal{LR}$ matrices.



a) Intervals b) Regions

Fig. 2. Cell clustering

Example. In Fig. 2, we provide two examples that illustrate how we cluster classified cells. Blue (backward-sloping lines) cells are of the same label λ_1, and green (forward-sloping lines) cells of the same label λ_2. The ones that are blank represent empty cells or cells that were assigned a different label than λ_1 and λ_2. In Fig. 2(a) cells are clustered into five label intervals. The label intervals I_4 and I_5, although in the same row, are separated because there is a cell of a different label between them. All intervals in Fig. 2(a) can be clustered into one label region. Contrary, in Fig. 2(b) there are three label regions, two blue (R_2 and R_3) and one green (R_1). We note that R_1 and R_2 "overlap". In the following sections we discuss how we treat these cases. For now we can say that this "overlap" hints some kind of relation between these regions. Also, the case of R_3 is particular, since it is a single cell region. Such cases can happen when it is not possible to cluster the cells both row-wise and column-wise.

2.2 Rectangular Abstractions

Although cell matrices are suitable structures for maintaining the $\mathcal{LR}s$, it is rather challenging to define heuristics on top of them. Therefore, we decided to go for a more abstract representation, namely the rectangle. An $\mathcal{LR}$ can be seen

as a rectangular structure that bounds cells of the same label. In the literature this is called the minimum bounding rectangle (MBR) for a set of objects, and is commonly used for tasks of spatial nature [4,13].

In our case, MBRs exist in the space defined by the original worksheet. The top-left corner of the worksheet becomes the origin $(0, 0)$. As shown in Fig. 3, the x-axis extends column-wise, while the y-axis extends row-wise. The edges of the MBRs are either parallel or perpendicular with these axes.

In this coordinate system, cells are rectangles, having unit width and unit height. As such, a cell $\mathcal{W}_{i,j}$ is represented by the coordinates[3]: $x_{min} = j - 1$, $x_{max} = j$, $y_{min} = i - 1$, and $y_{max} = i$.

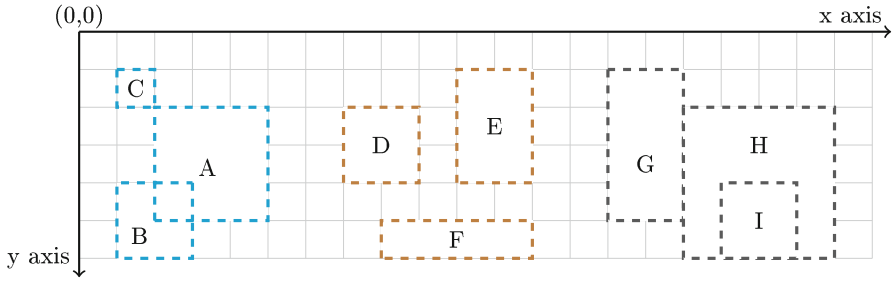


Fig. 3. Spatial relations between rectangles in a worksheet

We can determine the MBR coordinates for a $\mathcal{LR}$ from the indices of the cells it bounds. Specifically, we focus on the top-left and bottom-right cells in the $\mathcal{LR}$.

2.3 Spatial Arrangements

Here, we provide some of the notions used to describe spatial relations between the rectangles ($\mathcal{LR}s$). Our aim is to explain them intuitively using the examples in Fig. 3. Similar notions have been defined more formally in [13].

We start with the concepts that describe the relative location of rectangles. We use the notions *on the left of* and *on the right of* to describe relations like D to E and E to D, respectively. Likewise, F is *under* E, and the other way around E is *above* F. We can make these relations even more specialized. For example, F it is *not strictly under* E, since F is wider. However, D is *strictly on the left of* E, since its projection to the y-axis is within (covered by) E's y-axis projection. We are also interested in intersecting rectangles, and we have distinguished several of such cases: Two rectangles might *overlap*, such as A and B. They could *meet at a vertex*, like C and A. Rectangles G and H *meet at* or partially *share an edge*. Finally, rectangle I is *inside* rectangle H.

[3] Note, MBRs rely on a reference coordinate system, while $\mathcal{LR}s$ rely on the spreadsheet notation (i.e., column and row numbers).

3 Table Identification

TIRS consists of a series of heuristics that are based on the concepts presented in the previous sections. In addition to covering various table layouts, we had to minimize the effects of incorrect classifications and empty cells (i.e., missing values). Furthermore, we opted for heuristics that work on worksheets having multiple tables, stacked horizontally and/or vertically.

3.1 Tables in TIRS

Data, Header, and Attribute regions play the most important role in our analysis, since for us they are the base ingredients to form tables. Intuitively, a Data region ($\mathcal{LRD}$) acts like the core that brings everything together. A Header ($\mathcal{LRH}$) and Attribute region ($\mathcal{LRA}$) can help us distinguish the boundaries of tables. Therefore, we refer to them as "fences", a term borrowed from [1]. Fences can be horizontal (only Headers) or vertical (Headers[4] and Attributes).

A valid table should have at least a fence ($\mathcal{LRF}$) paired with a $\mathcal{LRD}$. In terms of dimension, tables must be at least a 2×2 matrix of cells. This means that $\mathcal{LRD}$ and $\mathcal{LRF}$ regions are at least 1×2 or 2×1 matrices.

$$table := \{Data, HHeaders, VHeaders, Attributes, Derived, Metadata, Other\}$$

Tables extracted by TIRS can be stored as collections of $\mathcal{LR}s$. More specifically, as shown above, a table has seven distinct sets of $\mathcal{LR}s$. For most of the cases we organize the regions forming the table by their label. We specialize Headers to vertical and horizontal. While the set "Other" contains regions for which we can not tell the layout function despite of their label. We provide more details on the latter in the following sections.

Finally, we utilize the MBR concept for tables, in addition to label regions. A table MBR is the minimum bounding rectangle for the $\mathcal{LR}s$ that compose it.

3.2 Pairing Fences with Data Regions

As mentioned in the previous section, TIRS needs to pair $\mathcal{LRD}s$ with $\mathcal{LRF}s$ to form tables. Valid pairs comply with the following three conditions.

C1. The $\mathcal{LRF}$ is on the top or on the left of the $\mathcal{LRD}$ although not necessarily adjacent to it.
C2. For a $\mathcal{LRF}$, the selected $\mathcal{LRD}$ is the closest[5]. Specifically, for a horizontal fence we measure the distance from the top edge of the Data region. Respectively, we work with the left edge for vertical fences.

[4] Vertical Headers occur infrequently in our annotated dataset for "pivoted" tables.
[5] We quantify this using the smallest Euclidean distance between two MBRs.

C3. The pair of MBRs representing correspondingly the $\mathcal{LRD}$ and the $\mathcal{LRF}$ are projected in one of the axes, depending on the fence orientation. The length of the segment shared by both projections represents the overlap. We transform the overlap into a ratio by dividing it with the largest projection.

$$\frac{Overlap(xProjection(\mathcal{LRD}), xProjection(\mathcal{LRF}))}{Max(xProjection(\mathcal{LRD}), xProjection(\mathcal{LRF}))} > \theta \quad (1)$$

Equation 1 shows how to calculate this for the x-axis (relevant to horizontal fences). The threshold θ was determined empirically and set to 0.5.

3.3 Heuristics Framework

The TIRS framework is composed of eight heuristic steps (activities). The initial Data-Fence pairs are created in the first five steps. While, the subsequent activities aim at completing the table construction by incorporating the remaining unpaired regions. In the following paragraphs we present each step and illustrate their relevance with examples from Fig. 4.
We should note that the examples in Fig. 4 hide the complexity of tables in our real-world dataset. For instance, fences might contain hierarchical structures, spanning in multiple rows and columns. Furthermore, misclassifications and empty cells can occur in arbitrary locations, and implicate various label regions (not only fences).

S1. In the first step, we attempt to create one-to-one pairs of Fence-Data, based on the three conditions listed in Sect. 3.2. Figure 4(a) and (d) provide examples of such tables.

S2. Mainly due to misclassifications multiple fence regions can be found that satisfy C1 and C2, but fail to comply with C3. An example is shown in Fig. 4(b). In such cases, we treat the individual regions as one composite fence, omitting the in-between "barriers". Equations 2 and 3 respectively show how to calculate the overlap ratio and projection-length to the x-axis for a composite fence ($\mathcal{CF}$), containing N sub-regions. We handle these calculations similarly for y-axis projections. Having the results from the equations, we proceed to check if C3 is satisfied.

$$cmp\_overlp = \sum_{i=1}^{N} Overlap(xProjection(\mathcal{LRD}), xProjection(\mathcal{CF}_i)) \quad (2)$$

$$cmp\_length = \sum_{i=1}^{N} xProjection(\mathcal{CF}_i) \quad (3)$$

S3. There can be a fence (simple or composite) that satisfies C3, but it is located inside the Data region far from the top edge or left edge. This might happen due to incorrect classification in worksheets that contain conjoined tables (i.e., not separated by empty columns or rows). We provide an example in

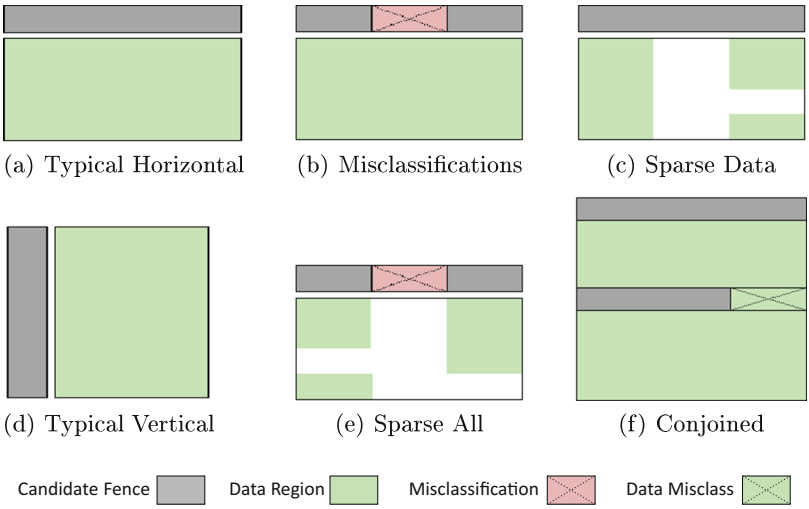


Fig. 4. Table types: cases b, c, e, and f also occur for tables with vertical fences

Fig. 4(f). When such a fence is identified, we separate the Data region into two parts. When the fence is horizontal, we pair it with the lower part, otherwise with the right part.

S4. There are cases where "small" Data regions are under or on the right of a "bigger" fence (e.g. the table in Fig. 4(c)). For these cases, the fence is treated as first-class citizen. Data regions that comply to condition C1 and are closer to this fence, than other ones, are grouped together. Again, we use similar formulas to Eqs. 2 and 3 to calculate the overlap and the projection-length of composite Data regions.

S5. At this step, we take a much more aggressive approach, in order to form tables with the remaining unpaired regions. We start by grouping fences. When working horizontally, we merge fences whose y-axis projections overlap. Likewise, we look for overlaps on the x-axis for vertical fences. Afterwards, we proceed in the same way as in step S4. Figure 4(e) illustrates a table that can be the output of this step.

S6. Here, we attempt to incorporate unpaired regions located in-between existing tables (i.e., constructed during S1–S5). In addition to the Data and fences, we also consider Metadata and Derived regions. For a pair of tables stacked horizontally, we assign the unpaired regions to the top table. When working with vertically stacked tables, we favor the left one. Obviously this and the following step, make sense when there are more than one extracted tables.

S7. We proceed by merging tables whose MBRs overlap. This will correct inconsistencies that might have happened during the previous steps. For example, a Data region is partially under a fence from another table.

S8. Finally, we assign the remaining unpaired regions, of all labels, to the nearest existing table.

Algorithm 1. Table creation in TIRS

Input: Set of $\mathcal{LRDS}$ (D), set of $\mathcal{LRHS}$ (H), set of $\mathcal{LRAS}$ (A)
Output: Set of extracted tables from the worksheet (T)

```
1  begin
2      T ← ∅;
3      UF ← ∅, UD ← D ;                    // UF: unpaired LRFs,UD: unpaired LRDs
4      O ← {Horizontal, Vertical};
5      foreach o in O do
6          if o == Horizontal then UF ← H else UF ← UF ∪ A;
7          foreach d in UD do
8              f ← GetNext(UF), newtbl ← false;
9              while f ≠ null and newtbl == false do
10                 if IsValidPair({d},{f},o) then                    // S1: line 10-12
11                     (T, UF, UF) ← Construct({d},{f}, UD, UF, T, o);
12                     newtbl = true;
13                 else if IsDataBreaker(d,f) then                   // S3: line 13-16
14                     (d1, d2) ← BreakInTwoParts(d,f);
15                     (T, UF, UF) ← Construct({d2},{f}, UD, UF, T, o);
16                     d ← d1;
17                 f ← GetNext(UF);
18             if newtbl == false then                               // S2: line 18-20
19                 CF ← GetCompositeFence(d, UF, o);
20                 if IsValidPair({d}, CF, o) then
21                     (T, UF, UF) ← Construct({d}, CF, UD, UF, T, o)
22     UH ← UF ∩ H, UA ← UF ∩ A;          // Extract unpaired Headers & Attributes
23     foreach o in O do
24         if o == Horizontal then UF ← UH else UF ← UF ∪ UA;
25         foreach f in UF do                                        // S4: line 23-25
26             CD ← GetCompositeData({f}, o, UD);
27             if IsValidPair(CD,{f}, o) then
28                 (T, UD, UF) ← Construct(CD,{f}, UD, UF, T, o);
29         foreach MF in MergeByOrientation(UF, o) do                // S5: line 26-28
30             CD ← GetCompositeData(MF, o, UD);
31             if IsValidPair(CD, MF, o) then
32                 (T, UF, UF) ← Construct(CD, MF, UD, UF, T, o)
33     return T;
34 Function Construct(SD, SF, UD, UF, T, o):   // SD: Selected LRDs, SF: Selected LRFs
35     table ← CreateTable(SD, SF, o);
36     TT ← T, TUD ← UD, TUF ← UF;          // Temporary variables in this function
37     (TUD, TUF) ← FilterOutPaired(table, TUD, TUF);
38     ConT ← HandleTableBreakers(table, TUF, o);          // Trivial case ConT = {table}
39     foreach t in ConT do
40         foreach u in {TUD ∪ TUF} do
41             if IsInside(table,u) or IsOverlap(table,u) then AddOtherRegion(table,u) ;
42         (TUD, TUF) ← FilterOutPaired(table, TUD, TUF);
43         TT ← TT ∪ {t};
44     return (TT, TUD, TUF);
```

Algorithm 1 provides a high level view from the execution of table creation steps (S1–S5). For each individual step S1 to S5, we first process horizontal and then vertical fences. Our empirical analysis showed the former are by far more common, thus we prioritize them. Additionally, we give priority to Headers over Attributes. It is fair to claim that Headers represent more "secure" fences, since less misclassification involve this label compared to Attributes [10]. Another details is that of S4 and S5 being executed only after all the types of fences are processed by steps S1–S3.

Furthermore, to avoid any inconsistencies, after the table creation we execute a series of operations. We incorporate regions that partially overlap or fall inside (complete overlap) the table (line 34–35). We exclude the paired regions from the next iterations (line 31 and 36). Also, we call function *HandleTableBreakers*, which basically is a batch execution of step S3.

Finally, at line 35 we use function *AddOtherRegion*. At this point of the algorithm we can not tell what the role of the fully or partially overlapping region is, since we already have paired the main components of the table. Therefore, we keep such regions at a special set called "Other".

4 Experimental Evaluation

In the following subsections, we discuss the evaluation of our proposed approach. Firstly, we present the dataset that was used for our experiment. Afterwards, we define how we measure the success of our method, and present the results of our evaluation.

4.1 Dataset of Annotated Tables

For our experiments we have considered spreadsheets from three different sources. EUSES [8] is one of the oldest and most frequently used copora. It has 4,498 unique spreadsheets, which are gathered through Google searches using keywords such as "financial" and "inventory". The ENRON corpus [9] contains over 15,000 spreadsheets, extracted from the Enron email archive. This corpus is of a particular interest, since it provides access to real-world business spreadsheets used in industry. The third corpus is FUSE [3] that contains 249,376 unique spreadsheets, extracted from Common Crawl[6].

From these three corpora, we randomly selected and annotated 216 spreadsheet documents. This translates into 465 individual worksheets. The annotations were performed by experts from our group, using a tool we developed in our previous work [10]. Each non-empty cell was assigned one of the five predefined labels (see Fig. 1). Additionally, we recorded tables as ranges of cells (storing the address of the top-left and bottom-right cells). Thus, for each annotated cell we can tell the table it belongs to.

For the evaluation of TIRS, we used 858 annotated tables. Out of this, 541 come from FUSE, 222 from ENRON, and 95 from EUSES. We should note that we omitted from our analysis 26 worksheets (40 "tables"). These worksheets contain only Data, and no fences. During the annotation phase we marked these Data as valid tables. However, later we decided to exclude them, since they do not comply anymore to our table definition (see Sect. 3.1).

4.2 Evaluation Objectives and Metrics

An extracted table T_e is considered a match to an annotated table T_a when they share at least 80% of their cells, considering only the Data, Header, and

[6] http://commoncrawl.org/.

Attribute regions (as mentioned in Sect. 3, these regions form the base of tables). To perform the comparison, we represent both T_e and T_a as rectangles. For a pair T_e and T_a we evaluate the spatial match using the formula below, where $\gamma = 0.8$.

$$match(T_e, T_a) = \frac{overlap(area(T_e), area(T_a))}{max(area(T_e), area(T_a))} \geq \gamma \tag{4}$$

We should note the reasons behind the omission of Derived and Metadata regions. Firstly, as can be seen from our table definition, they are not a must for its existence. Secondly, Metadata and Derived are not necessarily always related to a single table. During the annotation phase, we encountered Metadata that provide information relevant to multiple tables in the worksheet. Also, in our dataset a small number of Derived regions contain aggregations coming from several tables. Such regions, related to multiple tables, emerge "orphan" from our annotation phase, since we avoid assigning a table to them. Clearly, there is the need for more sophisticated ways to handle Metadata and Derived, but for the moment we exclude them from our analysis.

4.3 Evaluation Results

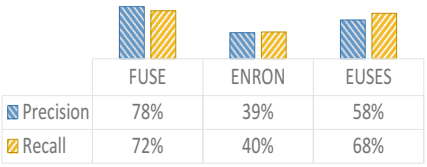
We present our evaluation per corpus, per number of misclassifications in the worksheet, and finally per table arrangements. The latter is related to the way tables are stacked in the worksheet.

We use precision and recall [14] to evaluate how good our approach is at identifying spreadsheet tables. In our context, precision measures the percentage of extracted tables (T_e), i.e., that match an annotated table (T_a). While, recall measures the percentage of T_a that were matched by our method.

Additionally, we compare the number of T_e with the number of T_a in the worksheet. The ratio where these numbers are equal is recorded by the "Equal" metric. The "Not Equal" metric records the cases these numbers differ (i.e., we extracted more or fewer tables than the actual number of tables in the worksheet).

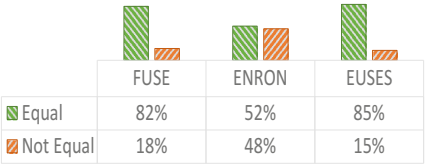
As seen in Fig. 5(a), our approach performs considerably well for FUSE tables, but poorly for ENRON tables. During an empirical examination, we noted that tables from ENRON tend to have a more complex structure, when compared to the other two corpora. We believe this to be the main reason for low scores in this corpus. This claim is further enforced by the results of the cell classification evaluation [10], where ENRON worksheets exhibit more misclassifications.

For EUSES, considering also Fig. 5(b), we are able to match well the actual number of tables in worksheets, but in terms of precision and recall we do not achieve that high scores. It seems that for a number of cases the extracted tables do not overlap significantly ($\geq$80%) with the annotated tables in the worksheet.



	FUSE	ENRON	EUSES
Precision	78%	39%	58%
Recall	72%	40%	68%

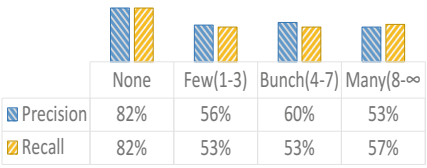
(a) Precision and Recall



	FUSE	ENRON	EUSES
Equal	82%	52%	85%
Not Equal	18%	48%	15%

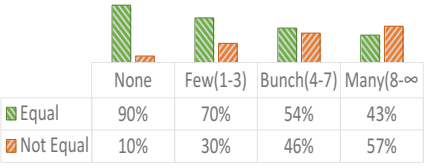
(b) Matching Number of Tables

Fig. 5. Results per corpus



	None	Few(1-3)	Bunch(4-7)	Many(8-∞
Precision	82%	56%	60%	53%
Recall	82%	53%	53%	57%

(a) Precision and Recall



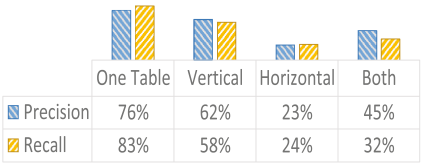
	None	Few(1-3)	Bunch(4-7)	Many(8-∞
Equal	90%	70%	54%	43%
Not Equal	10%	30%	46%	57%

(b) Matching the Number of Tables

Fig. 6. Results per number of misclassifications

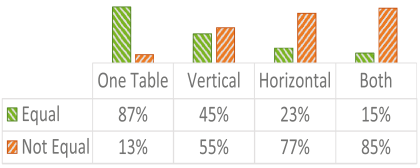
Figure 6 shows that our method performs well when there are no misclassifications. In contrast to what we expected, precision and recall do not follow a decreasing trend as we move to worksheets with more incorrect classifications. This is not the case for the *equal* and the *not equal* metrics.

We believe that in the case of precision and recall factors other than the number of incorrect classifications have strong influence. The graphs presented in Fig. 7 seem to support this claim.



	One Table	Vertical	Horizontal	Both
Precision	76%	62%	23%	45%
Recall	83%	58%	24%	32%

(a) Precision and Recall



	One Table	Vertical	Horizontal	Both
Equal	87%	45%	23%	15%
Not Equal	13%	55%	77%	85%

(b) Matching the Number of Tables

Fig. 7. Results per table arrangements

We observe that our method performs well for worksheets that contain one table, as shown in Fig. 7. We can also say, that precision and recall are tolerable for tables stacked vertically (row-wise). However, for horizontal alignments (column-wise) our scores are quite low. Probably, this impacts the performance for worksheets that contain both horizontal and vertical alignments of tables.

We believe the fact that we give more priority to horizontal fences, as mentioned mentioned at the end of Sect. 3.3, can explain the results in Fig. 7. Clearly,

we have biased TIRS towards tables stacked vertically. This is for a good reason, since they appear more frequently.

(a) Precision and Recall (b) Match # of Tables

Fig. 8. Overall results

In Fig. 8 we provide the overall results from our evaluation. In general, it is difficult to assess the performance of our approach, since to the best of our knowledge there is no similar work to directly compare these results with (see also Sect. 5). On the one hand, the precision and recall measures, provided in Fig. 8(a), are lower than expected. On the other hand, the results presented in Fig. 8(b) are satisfactory.

Closing the evaluation remarks, we point out that cells wrongly classified as Data and Header play a considerable role in the performance of TIRS. As we previously mentioned in this section, we exclude Metadata and Derived from our evaluation, and only consider the other three remaining labels. However, misclassifications might introduce Data and Header where there should have been Derived and Metadata cells. The classification results, presented in our previous paper [10], show that 98.7% of the misclassified Derived were labeled Data. For incorrectly classified Metadata, 43% were mistakenly marked as Data and 44% as Header. Such misclassifications often increase the size of T_e, and make it difficult to identify a match (true positive). In other words, T_e and T_a might share all the cells of the true table base (i.e., the annotated Fence and Data regions), but few incorrectly classified cells in T_e effectively reduce the ratio of their overlap (see Eq. 4).

5 Related Work

In this section, we review some of the related work on table identification and layout inference in spreadsheets. At [5] the authors present their work on what they call data frame spreadsheets (i.e., containing attributes or metadata regions on the top/left and a block of numeric values). Using linear-chain, conditional random field (CRF), they perform a sequential classification of rows in the worksheet, in order to infer its layout. Their next immediate focus is extracting the hierarchies found on the top (Header) and left (Attribute) regions. They proceed with the extraction of the data in the form of relational tuples, based on the information they inferred about the structure of data frame. In contrast to us, the authors do not distinguish the individual tables in the worksheet, but rather

assume only data-frame like spreadsheets. At [2], the authors present their work on schema extraction for Web tabular data, including spreadsheets. They extensively evaluated various methods for table layout inference, all operating at the row level. The CRF classifier combined with their novel approach for encoding cell features into row features (called "logarithmic binning") achieves the highest scores. Though the authors discuss how the inferred layout could be used to extract the schema for the tables in a spreadsheet, they do not provide an experimental evaluation of their claims. Nevertheless, we borrow from them and enforce with our work the idea that the header and data are instrumental for identifying and processing tables. The paper [1] presents work on header and unit inference for spreadsheets. Unlike us, the authors take a software engineering perspective. They utilize the inferred table structure to identify unit errors in spreadsheets. The authors have defined a set of heuristics based spatial-analysis algorithms, and a framework that allows them to combine the results from these algorithms. Unlike in our work, their spatial use cell features (e.g., content type and formula referencing), rather a pre-assigned labels from a classification task. Additionally, they have evaluated their approach in two datasets, containing 10 and 12 spreadsheets, respectively. They report few errors regarding the header inference, which is one of their main targets. However, the authors do not discuss how their framework performs on the table level. At [6], the authors present DeExcelerator, a framework which takes as input partially structured documents, including spreadsheets, and automatically transforms them into first normal form relations. For spreadsheets, their approach works based on a set of simple rules and heuristics that resulted from a manual study on real-world examples. Their framework operates on different granularity levels (i.e., row, column, and cell), considering the content, formating, and location of the cell/s. They evaluated the performance of their system on a sample of 50 spreadsheets extracted from data.gov, using human judges (10 database students). In contrast, we performed our evaluation in a much larger dataset covering a broader spectrum of spreadsheets.

6 Conclusions and Future Work

In this paper we presented TIRS, a heuristics based framework for table identification in spreadsheet. Unlike related work, we utilized the location and the labels assigned to the cells from a classification method we developed in a previous work. We introduced the concept of label regions and their representation as minimum bounding rectangles. The latter is a vital tool for defining a rich set of heuristics, such as the ones used in TIRS. For our evaluation, we used a large dataset of tables, covering various domains and formats. The results show that we achieve satisfactory performance in the sample of worksheets from FUSE and in worksheets that contain one table. The lowest scores come from ENRON worksheets and worksheets that contain horizontally stacked tables.

We see two possible actions to improve our approach in the future. Firstly, we can come up with more specialized heuristics, taking into account also the

domain of the spreadsheets. Here, in addition to the labels, we could utilize various cell features in a similar fashion as in related work. Secondly, we can enrich TIRS with more sophisticated techniques, coming from fields such as machine learning and statistics.

Acknowledgments. This research has been funded by the European Commission through the Erasmus Mundus Joint Doctorate "Information Technologies for Business Intelligence - Doctoral College" (IT4BI-DC).

References

1. Abraham, R., Erwig, M.: Header and unit inference for spreadsheets through spatial analyses. In: VL/HCC 2004, pp. 165–172. IEEE (2004)
2. Adelfio, M.D., Samet, H.: Schema extraction for tabular data on the web. VLDB **6**(6), 421–432 (2013)
3. Barik, T., Lubick, K., Smith, J., Slankas, J., Murphy-Hill, E.: FUSE: a reproducible, extendable, internet-scale corpus of spreadsheets. In: MSR 2015 (2015)
4. Caldwell, D.R.: Unlocking the mysteries of the bounding box (2005)
5. Chen, Z., Cafarella, M.: Automatic web spreadsheet data extraction. In: SSW 2013, p. 1. ACM (2013)
6. Eberius, J., Werner, C., Thiele, M., Braunschweig, K., Dannecker, L., Lehner, W.: DeExcelerator: a framework for extracting relational data from partially structured documents. In: CIKM 2013, pp. 2477–2480. ACM (2013)
7. Eckerson, W.W., Sherman, R.P.: Strategies for managing spreadmarts. Bus. Intell. J. **13**(1), 23–24 (2008)
8. Fisher, M., Rothermel, G.: The EUSES spreadsheet corpus: a shared resource for supporting experimentation with spreadsheet dependability mechanisms. In: SIGSOFT 2005, vol. 30, pp. 1–5. ACM (2005)
9. Hermans, F., Murphy-Hill, E.: Enron's spreadsheets and related emails: a dataset and analysis. In: Proceedings of ICSE 2015. IEEE (2015)
10. Koci, E., Thiele, M., Romero, O., Lehner, W.: A machine learning approach for layout inference in spreadsheets. In: KDIR (2016)
11. Mohanty, H., Bhuyan, P., Chenthati, D.: Big Data: A Primer. Springer India, New Delhi (2015)
12. O'Leary, D.E.: Embedding ai and crowdsourcing in the big data lake. IEEE Intelligent Systems **29**(5), 70–73 (2014)
13. Papadias, D., Theodoridis, Y.: Spatial relations, minimum bounding rectangles, and spatial data structures. International Journal of Geographical Information Science **11**(2), 111–138 (1997)
14. Ting, K.M.: Precision and recall. In: Encyclopedia of machine learning, pp. 781–781. Springer (2011)