Table Recognition in Spreadsheets via a Graph Representation

Elvis Koci, Maik Thiele, Wolfgang Lehner
Department of Computer Science
Technische Universität Dresden
Dresden, Germany
Email: {name.surname}@tu-dresden.de

Oscar Romero
Departament d'Enginyeria de Serveis i Sistemes d'Informació
Universitat Politècnica de Catalunya-BarcelonaTech
Barcelona, Spain
Email: oromero@essi.upc.edu

***Abstract*—Spreadsheet software are very popular data management tools. Their ease of use and abundant functionalities equip novices and professionals alike with the means to generate, transform, analyze, and visualize data. As a result, spreadsheets are a great resource of factual and structured information. This accentuates the need to automatically understand and extract their contents. In this paper, we present a novel approach for recognizing tables in spreadsheets. Having inferred the layout role of the individual cells, we build layout regions. We encode the spatial interrelations between these regions using a graph representation. Based on this, we propose *Remove and Conquer* (RAC), an algorithm for table recognition that implements a list of carefully curated rules. An extensive experimental evaluation shows that our approach is viable. We achieve significant accuracy in a dataset of real spreadsheets from various domains.**

***Keywords*-Spreadsheet; Table Recognition; Table Identification; Graph; Rule-based**

I. Introduction

Spreadsheets are used for a great variability of data management tasks from users with different expertise level. The abundance of usage led to an enormous wealth of structured data contained in spreadsheets. However, the free-for-all nature and the lack of schema (or a specific data model) prevents the reusability and visibility of this information. This results into data silos.

While there is some support to perform spreadsheet data extraction, like [1] and [2], it can not be considered a general purpose solution for arbitrary inputs. Previous work often assumes just one table per sheet. Furthermore, these tables are expected to be well formed and complete.

In this paper, we focus on the task of table recognition for single-table and multi-table spreadsheets (see Figure 1a for an example). We thereby do not rely on any assumptions with what regards the arrangement of tables in these documents. Additionally, we enforce a certain level of flexibility, which allows us to work even with slightly problematic tables (e.g., containing empty cells).

Since spreadsheets are comprehended visually by humans, they contain many formatting and stylistic artifacts. Therefore, an automatic approach needs to infer the structural semantics attached to the various subsections of the user generated content. For our proposed approach, we use a Random Forest classifier to infer the layout role of individual cells in spreadsheets (see Figure 1b), based on previous work outlined at [3].

In a similar fashion to [4], we then use the inferred roles to create the so-called layout regions (see Figure 1c). These group together adjacent cells having the same layout role. As a result, we get coherent blocks of cells, which are easier to handle for the table recognition task.

Here, we propose a graph representation where vertices correspond to the layout regions of a sheet and edges encode the spatial interrelations between these regions. Moreover, we annotate the vertices with additional properties, such as the coordinates of the corresponding region.

Given the dual nature (spatial and semantic) of the problem, graphs can naturally provide an intuitive ground to debug and experiment with different methods for table recognition. In this paper, we present our approach, called Remove and Conquer (RAC). We implement a comprehensive set of rules and heuristics to identify tables, given the graph representation of a sheet. Additionally, in Section VI-D, we discuss our experiments with an alternative approach involving this representation.

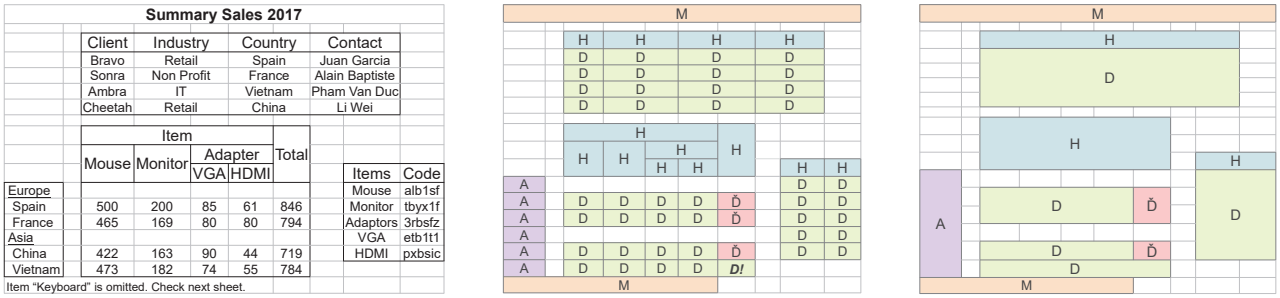
The rest of the paper is organized as follows. Section III provides formal definitions for terms used throughout the proposed approach. In Section IV we describe the creation of the graph representation. A detailed description of RAC is given in Section V. Our experimental evaluation is outlined in Section VI, followed by related work in Section II. We conclude this paper in Section VII.

II. Related Work

There is a considerable number of works tackling layout inference and information extraction in spreadsheets. Recent publications propose approaches involving to some extent machine learning techniques, such as [2], [3], [4], [5], and [6]. Also, we find rule-based approaches, like [7]. Other works make use of domain specific languages, such as [1], [8], and [9]. Furthermore, it is worth mentioning [10], and [11], which discuss systems having spreadsheets as front-end and a relational databases as back-end.

As emphasized in the introduction, most of the aforementioned publications assume single-table worksheets. In this paper, we avoid this assumption and work with arbitrary number of tables and arrangements. In addition, we aim at recognizing tables even in the presents of irregularities, such as misclassifications and empty cells.

An active area of research is spreadsheet debugging (i.e., error detection). In these works, similar to us, individual cells and regions of cells are studied to detect errors.



(a) Initial Worksheet (b) Classify Cells (c) Create Label Regions

Figure 1. Example worksheet containing three tables.

At [12] worksheet contents are decomposed into fragments to assist better debugging of formula cells. A recent work, [13], detects formula errors using neural networks. Also, [14] outlines a rule-based approach to infer unit errors.

Table recognition has been a subject of research in other document formats, as well. The following surveys [15] and [16] provide a comprehensive summary of such works. Here we single out [17]. This work outlines a method for recognizing tables in document images. They make use of a graph model, which is very similar to the one we propose for spreadsheets. Also, we can draw parallels with [18], which proposes an approach for analysis of complex table forms. The authors, in a similar fashion to us, study the geometrical arrangement and the semantics of the individual fields that compose the forms.

III. Preliminaries

A spreadsheet file contains one or more sheets. Each sheet is comprised of a collection of cells organized in rows and columns. This results into a grid-like structure. A cell can be uniquely identified using its row and column number.

In this paper, we adopt the Microsoft Excel terminology, and refer to a sheet from now on as *worksheet*. Moreover, we make use of the following definitions.

Definition 1. Let W denote the set containing all the cells of a worksheet. We define $row : W \mapsto \mathbb{N}_{>0}$ as a function that maps cells in W to their row number. Respectively, $col : W \mapsto \mathbb{N}_{>0}$ returns the column number.

In the proposed approach we will extensively work with subsets of W. However, we are not interested in any arbitrary subset. Instead, we focus on those holding cells from rectangular areas of the worksheet.

Definition 2. A Region, $R \subseteq W$, is the collection of all cells from a rectangular area of the worksheet. There does not exist a cell $c \notin R$ such that $minRow(R) \leq row(c) \leq maxRow(R)$ and $minCol(R) \leq col(c) \leq maxCol(R)$. Here, the functions prefixed with max and min return the boundaries of the region R.

Before identifying tables, we infer the layout role of non-empty cells in the worksheet. As outlined at [3], the latter is defined as a classification task. Each non-empty cell is assigned one of the following layout roles: Header (*H*), Data (*D*), Metadata (*M*), Attributes (*A*), Derived (*Ď*).

Definition 3. Let $label : W \mapsto Labels$, where $Labels = \{Header, Data, Attributes, Metadata, Derived\}$, be a function that maps cells to their assigned layout role. For empty cells $label$ is undefined. We identify them using $empty : W \mapsto \{0, 1\}$. It returns 1 for empty cells, otherwise 0.

With respect to the Wang model [19], Headers correspond to the Boxhead and Stub Head. Attributes are row headers in the Stub. Data is the body of the table. For the rest we provide our definitions. Metadata is additional information about the table, such as the title and footnotes. Derived are aggregations (typically via formulas) of Data.

By grouping together adjacent cells having the same layout role (label), we form larger structures. We refer to them as *Label Regions*. This term was also used at [4], but with a different definition than the one provided below[1].

Definition 4. A Label Region (LR) is a region of a worksheet, where $\forall c, c' \in LR$ the $label(c) = label(c')$ and $empty(c) \neq 1,\ empty(c') \neq 1$.

We briefly describe the procedure for building LRs as follows. In the first pass, we iterate over each row to create sequences of cells having the same label. These form the base LRs. Subsequently, we merge LRs from consecutive rows, if their labels, minimum column, and maximum column match. Figure 1c displays the label regions corresponding to the classified cells in Figure 1b. Note, there can be incorrect classifications in worksheets, such as *D!* at the lower end of Figure 1b.

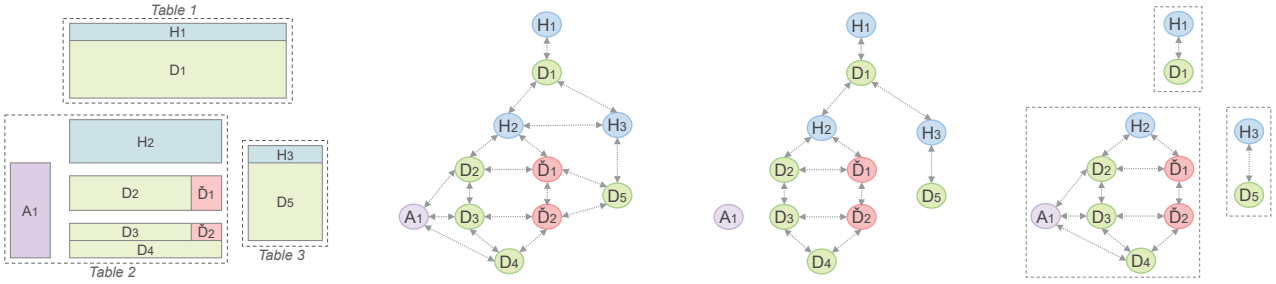
The label regions created from this procedure tend to be wide (i.e., spanning multiple columns). This is favorable for the proposed approach. As outlined in Section V, Header regions play a key role in our analysis. Typically, Headers reside in few rows, but span multiple columns. Thus wide regions, comply better with this behavior.

IV. Building Graphs

Once the regions are constructed we build a graph representation to assist the table recognition task.

Definition 5. Let $G(V, E)$ be the a *directed* graph that captures the spatial interrelations of label regions ($\mathcal{LR}$)

[1]Here, we do not allow overlaps between label regions. As well as, we ensure that there are no empty cells inside a label region.



(a) Load Regions (b) Create Graph (c) Remove Left/Right (d) Output Tables

Figure 2. From Regions to Tables via a Graph Representation

from a worksheet W. There is a one-to-one correspondence between V, the set of vertices, and $\mathcal{LR}$.

Furthermore, we carry the label region characteristics into the graph. For this we define the following functions.

Definition 6. Function $lbl : V \mapsto Labels$ maps vertices to labels. Moreover, $rmin : V \mapsto \mathbb{N}_{>0}$ and $rmax : V \mapsto \mathbb{N}_{>0}$ return respectively the minimum and maximum row number. Equivalently, $cmin$ and $cmax$ do the same for the column numbers.

Next step is to create edges between the vertices of the graph. Here, our aim is to identify spatial relations such as *top of*, *bottom of*, *left of*, and *right of*. In other terms, we capture the relative location of other regions with respect to the selected region in the following four directions: Top, Bottom, Left, and Right. However, we focus only on the nearest neighboring regions for each direction.

To better illustrate the edge creation process, below we outline the steps for the identification of the nearest neighbors on the *Top* for a vertex $v \in V$.

$$T_v = \{u \in V | rmin(v) > rmax(u) \texttt{ and } \texttt{not } (cmin(v) > cmax(u) \texttt{ or } cmax(v) < cmin(u))\}$$

As shown above, we identify all vertices whose maximum row is less than the minimum row of v. On the same time, we enforce that the selected vertices span, at least partially, the same columns as v. To get the nearest vertices we use the following distance functions.

Definition 7. For each direction we define a distance function. Let $tdist := rmin(v) - rmax(u)$ and $bdist := rmin(w) - rmax(v)$ calculate respectively the distance from Top ($u \in T_v$) and Bottom ($w \in B_v$) vertices of v. Likewise, we define $ldist$ for Left and $rdist$ for Right.

Returning to our example, we identify the nearest vertices at the Top for v as follows.

$$NT_v = \{n \in T_v | \ tdist(v, n) = \min_{u \in T_v} tdist(v, u)\}$$

We create a directed edge (v, n), for every $n \in NT_v$. The same can be performed for the nearest neighbors of v on the rest of the directions. Furthermore, using these steps all the vertices of the graph can be analyzed to populate the set of edges E. Figure 2b provides an example graph corresponding to the regions shown in Figure 2a.

Note, we always define two edges for a vertex and its immediate neighbor. More specifically, for an edge $(v, u) \in E$ there exists an equivalent edge $(u, v) \in E$. In Figure 2 we use bidirectional arrows to depict this. Typically, being the nearest neighbor of vertex holds the other way around, from the viewpoint of the neighbors. However, there are also counter examples. In Figure 2, the nearest Top neighbor for H_3 is D_1, but the nearest Bottom neighbor for D_1 is H_2. In this and similar scenarios, we enforce symmetry by creating the equivalent directed edge, which goes the opposite way.

For the proposed approach we need to tell the relation for connected vertices, which is implicit in Figure 2. Therefore, we define the following function.

Definition 8. $dir : E \mapsto \{Top, Bottom, Left, Right\}$ is a function that maps edges to directions. For an edge $(v, u) \in E$ the result of this function communicates the direction that u is nearest neighbor of v.

Finally, we point out that in Figure 2 there are no Metadata regions. We do not consider them for the proposed approach. Metadata could describe multiple tables in a worksheet. As a result, they can not be deterministically assigned to just one table, during our process. Therefore, we leave this as an open question for future work.

V. Remove and Conquer

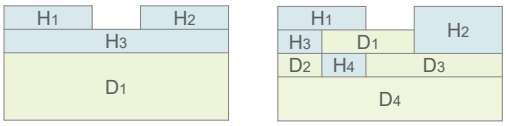
For each worksheet in our dataset we construct a directed graph, as described previously. Our rule-based algorithm, RAC, processes these graphs individually, and outputs a set of proposed tables $\mathcal{P}$. These tables themselves are collections of vertices, which in turn can be used to create the table subgraphs from the original input graph (Figure 2d). Additionally to this, RAC returns, in a separate set U, vertices that could not form tables.

Initially, RAC attempts to divide the vertices into horizontal groups. Next, these are subdivided vertically. The resulting groups should resemble valid tables. Nevertheless, in the final steps RAC attempts to re-assign vertices from U to valid tables.

Vertices from neighboring tables arranged horizontally get connected with Left and Right edges (e.g., *Table 2* and *Table 3* in Figure 2). Such edges can also be found

within the same table. Yet, for the latter we expect Top and Bottom edges as well. Thus, removing those Left and Right (lines 2-4, Figure 4) should mostly impact the inter-table connections. Nevertheless, we take measures to protect the intra-table connections. We avoid removing edges when distance is 1 (i.e., adjacent vertices with no other column in-between them). In lines 23-27 we address the cases where the distance is greater.

In lines 5-21 we process the strongly connected components (SCC) of the graph. These represent what we previously referred to as the horizontal groups. For each component (G^S) we seek to pair Headers with Data, Attributes, and Derived in order to form valid tables. These in turn will be the so called vertical groups.



(a) True Layout Roles (b) Misclassified Cells

Figure 3. The impact of misclassifications

Before going into the details on how RAC splits vertically, we need to address the implications arising from misclassified cells. Consider the examples above. In the left (Figure 3a) regions are formed using the true layout roles of the cells. While, in Figure 3b we use the predicted roles. Here, cells in D_1 and H_4 were incorrectly classified.

RAC needs to infer that all regions in Figure 3b belong to one table, instead of many. The alignment between Header and Data regions provides some hints. The same is true for the size of the regions.

We use these insights starting from line 7 of RAC (Figure 4). We perform our search for tables in an inverse manner, from bottom to top. For this, we sort vertices in descending order of their maximum row, followed by the ascending order of their minimum row. Note, for Figure 3b this means H_2 will be ordered before D_1, H_3 and H_1.

We process each Header h individually, in lines 10-19. If h is not already paired (line 11), we proceed to identify vertices having minimum row greater than or equal to h. The latter represents a set of vertices (including h) with the potential to form a table, denoted as Q. However, we also need to handle scenarios like in Figure 2, where D_5 satisfies the above condition for H_2. Thus, we additionally ensure that the other vertices are connected with h.

Line 13 checks the validity of a Header (discussed in detail later). Vertices paired with a valid Header are subtracted from S', the list of sorted vertices. However, we do not append Q to $\mathcal{P}$ yet. Consider, the scenario in Figure 3a. Pairing H_3 with D_1 could form a seemingly complete table, but it leaves H_1 and H_2 out. Thus, in line 17-19 we identify Headers having no other vertex to pair with (i.e., only h satisfies the conditions in line 12). We append these headers to the Q of the last valid Header, denoted as LQ, if they have columns in common (i.e., are aligned) with vertices in LQ.

There are cases where tables can not be formed. In line 9 we identify strongly connected components that do not contain headers, and store them in U (line 21). The latter holds also vertices that remain still un-paired, after processing all Headers in the component.

Input: G: graph representation of a worksheet
Output: $\mathcal{P}$: proposed tables, U: other undetermined
1: $\mathcal{P} \leftarrow \emptyset$; $U \leftarrow \emptyset$;
2: $E_l \leftarrow \{e \in E | dir(e) = Left$ **and** $ldist(e) > 1\}$
3: $E_r \leftarrow \{e \in E | dir(e) = Right$ **and** $rdist(e) > 1\}$
4: $E \leftarrow E \setminus (E_l \cup E_r)$
5: **for all** $G^S \in getSCC(G)$ **do** // G^S = (S,E_S)
6: $LQ \leftarrow NIL$ // holds Q of last valid Header
7: $S' \leftarrow sortVertices(S)$
8: $S'_H \leftarrow \{v \in S' | lbl(v) = Header\}$
9: **if** $|S'_H| > 0$ **then**
10: **for all** $h \in S'_H$ **do**
11: **if** $h \in LQ$ **then continue**
12: $Q \leftarrow \{s \in S' | rmin(s) \geq rmin(h)$ **and** $hasPath(s, h, E_S)\}$
13: **if** $isValid(h, Q, 0.5)$ **then**
14: $\mathcal{P} \leftarrow \mathcal{P} \cup \{LQ\}$
15: $S' \leftarrow S' \setminus Q$
16: $LQ \leftarrow Q$
17: **else if** $LQ \neq NIL$ **then**
18: **if** $|Q| = 1$ **and** $isAligned(h, LQ)$ **then**
19: $LQ \leftarrow LQ \cup \{h\}$
20: $\mathcal{P} \leftarrow \mathcal{P} \cup \{LQ\}$ // after for all ends
21: $U \leftarrow U \cup S'$ // remaining unpaired
22: $\mathcal{P}, U \leftarrow handleOverlapping(\mathcal{P}, U)$
23: **for all** $u \in U$ **do** // find nearest table left or right
24: $N, dist \leftarrow getNearestVertices(u, (E_l \cup E_r))$
25: $\mathcal{P}' \leftarrow \{P \in \mathcal{P} |\ 0 < |N \cap P|\}$
26: **if** $|\mathcal{P}'| = 1$ **and** $dist \leq 3$ **then**
27: $P \leftarrow P \cup \{u\}$, where $P \in \mathcal{P}'$
28: **return** $\mathcal{P}$, U

Figure 4. Remove and Conquer Algorithm

Occasionally, the minimum bounding rectangle enclosing the regions (vertices) of a proposed table will overlap with that of another one. Consider Table 1 in Figure 2. If it was extending more to the left, it would align with A_1. In such scenario, A_1 would mistakenly be paired with H_1 and D_1. The function in line 22 handles such cases. Initially, it attempts to merge the two overlapping tables, to form a larger valid one. Otherwise, it targets the vertices causing the overlap (referred to as OV). The minimum header row of the lower table is used as boundary. Vertices spanning more rows on the other side of the boundary, rather than in their own table, are considered OV. They are moved from the proposed tables to U, for further processing.

The last steps, lines 24-27, attempt to pair regions in U with the nearest table on their Left or Right. We can make this decision deterministically, only when there is one nearest table. Here, additional constrains could be imposed on the minimum distance (line 26).

The procedure to check the header validity is outlined

Input: h: a Header vertex, Q: vertices to form table with, th: threshold for alignment ratio
Output: $True$ if h is valid, $False$ otherwise

1: **if** $|\{q \in Q | rmin(q) > rmax(h)\}| > 0$ **then**
2: $Q_{\mathcal{H}} \leftarrow \{q \in Q | lbl(q) = Header$ **and** $rmin(q) \leq rmax(h)$ **and** $rmin(q) \geq rmin(h)\}$
3: $X \leftarrow \emptyset$; $X' \leftarrow \emptyset$
4: **for all** $u \in Q_{\mathcal{H}}$ **do**
5: $X \leftarrow X \cup \{x \in \mathbb{N} | cmin(u) \leq x \leq cmax(u)\}$
6: **for all** $v \in Q \setminus Q_{\mathcal{H}}$ **do**
7: $X' \leftarrow X' \cup \{x \in \mathbb{N} | cmin(v) \leq x \leq cmax(v)\}$
8: **return** $\frac{|X \cap X'|}{|X'|} \geq th$ **and** $|X| > 1$
9: **else**
10: **return** $False$

Figure 5. Header Validity Check ($isValid$)

in Figure 5. Initially, we make sure that there are vertices below the specified Header h (line 1 in Figure 5). Also, in line 2 we identify Headers in Q that span one or more rows in the range $[cmin(h), cmax(h)]$. Note, again the latter contains h itself. We calculate the alignment ratio of these Headers with the rest of the vertices (line 3-8). Intuitively, our aim is to ensure that Headers sit on top of the other values in the columns. For the experimental evaluation, we constrain the alignment ratio to be ≥ 0.5. We also ensure that Headers span two or more columns. Single-column Headers suggests lists rather than tables.

VI. Experimental Evaluation

A. Spreadsheet Dataset

We drew a representative sample of 208 worksheets[2], from three corpora; FUSE [20], ENRON [21], and EUSES [22] respectively contribute 128, 56, and 28 worksheets. The resulting dataset was manually annotated, so we are aware of the cell layout roles and the location of tables. In detail, 176 worksheets contain one table, while 32 contain multiple tables (up to 68). For multi-table sheets, we have 29 instances with vertically (top-down) arranged tables and 3 with horizontally (left-right) arranged tables.

Additionally, the dataset was pre-processed as follows. During the region creation we omit rows and columns that are hidden (i.e., not displayed). This is necessary, since we do the same prior to the classification task. Furthermore, we omit columns with width ≤ 2 characters for standard Excel font. Here, the intention is to discard empty columns used for formatting purposes within tables.

B. Objective

The goal of our table recognition approach is to maximize the match between a proposed table P and a true table T. This means maximizing the number of cells they have in common and minimizing the number of cells by which they differ. Here, we only consider non-empty cells, which also constitute our label regions (LRs). Note that a region might contain cells from two distinct true tables. This arises due misclassifications when tables are adjacent.

$$\mu(P,T,V) = \frac{\sum_{u \in P} areaIn(u,T)}{\sum_{u \in P} area(u) + \sum_{v \in V \setminus P} areaIn(v,T)}$$

The formula for calculating the match for a pair P, T is defined above. The function $areaIn$, calculates how much from the area of a vertex (region) is in T. Also, note V is the set of vertices in G, and $P \subseteq V$.

C. Testing RAC

We test RAC on the gold standard (i.e., true cell layout roles are used to create regions), as well as on the classified worksheets. Note, the latter could contain misclassifications. Additionally, we have conceived a stress test for RAC. We induce different levels of random noise into the gold standard (from 1% up to 20%). Here, we used a uniform distribution, i.e. each cell label could be flipped to one of the other four labels with the same chance.

Figure 6 presents the evaluation results on the gold standard and classified cells. T is considered recognized if there exists a P such that they have a match ≥ 0.9. Additionally, the chart below provides individually the results for worksheets with single and multiple tables.

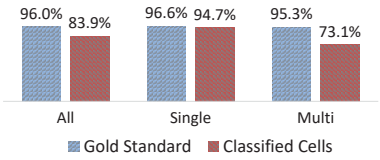


Figure 6. Gold Standard Vs Classified Cells

We note that RAC has considerably high accuracy in the gold standard, while the numbers are lower for the classified cells. Further analysis showed that incorrect classifications often occur in adjacent cells. This has an impact on valid Headers, making it harder to spot them. On the other hand, misclassifications might introduce false Headers. We observe that thin tables (i.e., ≤ 4 columns) are the more vulnerable, since a relatively small number of misclassifications could prevent their recognition.

Another factor is empty columns. They can be found in true tables, as formatting artifacts. In other cases, they are used to separate tables arranged horizontally. Additional domain specific rules are required to better differentiate between these two usages.

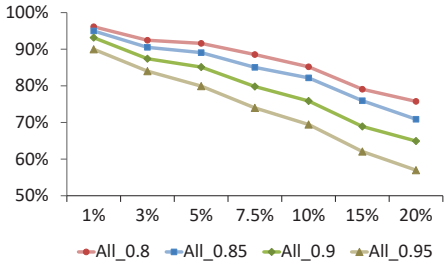


Figure 7. Stress Test

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

The results from the stress test are presented in Figure 7. For each level of induced noise we generate three copies from the gold standard and average the results. Unlike the previous chart, we consider different thresholds (0.8, 0.85, 0.9, and 0.95) for the match between true and proposed tables. The results show that RAC is noticeably resilient to random noise. The performance is considerably good even for high levels of induced error.

D. Alternative Methods

Besides RAC, we experimented with a method incorporating a weighted undirected graph. Again, vertices correspond to layout regions. Edges are weighted using metrics involving the distance and the alignment between layout regions. Then, we make use of the minimum cut algorithm from Stoer-Wagner [23], to iteratively cut weak edges. A stopping criteria eventually terminates this process. The resulting subgraphs are treated as proposed tables. Nevertheless, our evaluation showed that this method recognizes only 86.74% of the tables in the gold standard dataset.

VII. Conclusion

This paper proposes a rule-based table recognition approach for spreadsheets, called Remove and Conquer (RAC). Having inferred the layout roles of the individual cells, from previous work, we build layout regions. Then, we use a graph model to describe their arrangement (i.e., location with respect to each other). The vertices of the graph represent the layout regions together with their properties. Edges are directed and each one of them points to a neighboring vertex (region) in one of the following directions: Top, Bottom, Left, and Right.

This graph representation provides a rich context, upon which we define a set of curated rules. RAC applies these rules to return a list of proposed tables per worksheet. Our experimental evaluation on a diverse corpus of spreadsheets shows encouraging results.

In the future, we aim to expand this work in the following ways. We can define more rules for RAC to capture additional table structures. Alternatively, we can test more automatic methods with the given graph representation. Finally, we plan to enlarge further our dataset with spreadsheet from various sources.

References

[1] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer, "Wrangler: Interactive visual specification of data transformation scripts," in *SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2011, pp. 3363–3372.

[2] Z. Chen, S. Dadiomov, R. Wesley, G. Xiao, D. Cory, M. Cafarella, and J. Mackinlay, "Spreadsheet property detection with rule-assisted active learning," in *CIKM*. ACM, 2017, pp. 999–1008.

[3] E. Koci, M. Thiele, Ó. Romero Moral, and W. Lehner, "A machine learning approach for layout inference in spreadsheets," in *IC3K: volume 1: KDIR*. SciTePress, 2016, pp. 77–88.

[4] E. Koci, M. Thiele, O. Romero, and W. Lehner, "Table identification and reconstruction in spreadsheets," in *CAiSE*. Springer, 2017, pp. 527–541.

[5] Z. Chen and M. Cafarella, "Automatic web spreadsheet data extraction," in *International Workshop on Semantic Search over the Web*. ACM, 2013, p. 1.

[6] M. D. Adelfio and H. Samet, "Schema extraction for tabular data on the web," *Proceedings of the VLDB Endowment*, vol. 6, no. 6, pp. 421–432, 2013.

[7] J. Eberius, C. Werner, M. Thiele, K. Braunschweig, L. Dannecker, and W. Lehner, "Deexcelerator: A framework for extracting relational data from partially structured documents," in *CIKM*. ACM, 2013, pp. 2477–2480.

[8] A. O. Shigarov and A. A. Mikhailov, "Rule-based spreadsheet data transformation from arbitrary to relational tables," *Information Systems*, vol. 71, pp. 123–136, 2017.

[9] F. Hermans, M. Pinzger, and A. Van Deursen, "Automatically extracting class diagrams from spreadsheets," *ECOOP 2010–Object-Oriented Programming*, pp. 52–75, 2010.

[10] M. Bendre, B. Sun, D. Zhang, X. Zhou, K. C.-C. Chang, and A. Parameswaran, "Dataspread: Unifying databases and spreadsheets," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 2000–2003, 2015.

[11] J. Cunha, J. Saraiva, and J. Visser, "From spreadsheets to relational databases and back," in *SIGPLAN workshop on Partial evaluation and program manipulation*. ACM, 2009, pp. 179–188.

[12] T. Schmitz, D. Jannach, B. Hofer, P. W. Koch, K. Schekotihin, and F. Wotawa, "A decomposition-based approach to spreadsheet testing and debugging," in *VL/HCC*. IEEE Computer Society, 2017, pp. 117–121.

[13] R. Singh, B. Livshits, and B. Zorn, "Melford: Using neural networks to find spreadsheet errors," Tech. Rep., January 2017. [Online]. Available: https://www.microsoft.com/en-us/research/publication/melford-using-neural-networks-find-spreadsheet-errors/

[14] R. Abraham and M. Erwig, "Header and unit inference for spreadsheets through spatial analyses," in *VL/HCC*. IEEE, 2004, pp. 165–172.

[15] R. Zanibbi, D. Blostein, and J. R. Cordy, "A survey of table recognition," *Document Analysis and Recognition*, vol. 7, no. 1, pp. 1–16, 2004.

[16] D. Lopresti and G. Nagy, "A tabular survey of automated table processing," in *International Workshop on Graphics Recognition*. Springer, 1999, pp. 93–120.

[17] M. A. Rahgozar and R. Cooperman, "A graph-based table recognition system," *Document Recognition*, vol. 111, pp. 192–203, 1996.

[18] A. Amano and N. Asada, "Complex table form analysis using graph grammar," *Lecture notes in computer science*, pp. 283–286, 2002.

[19] X. Wang, "Tabular abstraction, editing, and formatting," University of Waretloo, Waterloo, Ontaria, Canada, Tech. Rep., 1996.

[20] T. Barik, K. Lubick, J. Smith, J. Slankas, and E. Murphy-Hill, "F use: a reproducible, extendable, internet-scale corpus of spreadsheets," in *Working Conference on Mining Software Repositories*. IEEE Press, 2015, pp. 486–489.

[21] F. Hermans and E. Murphy-Hill, "Enron's spreadsheets and related emails: A dataset and analysis," in *International Conference on Software Engineering-Volume 2*. IEEE Press, 2015, pp. 7–16.

[22] M. Fisher and G. Rothermel, "The euses spreadsheet corpus: a shared resource for supporting experimentation with spreadsheet dependability mechanisms," in *SIGSOFT*, vol. 30, no. 4. ACM, 2005, pp. 1–5.

[23] M. Stoer and F. Wagner, "A simple min-cut algorithm," *Journal of the ACM*, vol. 44, no. 4, pp. 585–591, 1997.