A Genetic-based Search for Adaptive Table Recognition in Spreadsheets

Elvis Koci*†, Maik Thiele*, Oscar Romero†, Wolfgang Lehner*

*Fakultät Informatik
Technische Universität Dresden
Dresden, Germany
name.surname@tu-dresden.de

†Departament d'Enginyeria de Serveis i Sistemes d'Informació
Universitat Politècnica de Catalunya-BarcelonaTech
Barcelona, Spain
{ekoci, oromero}@essi.upc.edu

***Abstract*—Spreadsheets are very successful content generation tools, used in almost every enterprise to create a wealth of information. However, this information is often intermingled with various formatting, layout, and textual metadata, making it hard to identify and interpret the tabular payload. Previous works proposed to solve this problem by mainly using heuristics. Although fast to implement, these approaches fail to capture the high variability of user-generated spreadsheet tables. Therefore, in this paper, we propose a supervised approach that is able to adapt to arbitrary spreadsheet datasets. We use a graph model to represent the contents of a sheet, which carries layout and spatial features. Subsequently, we apply genetic-based approaches for graph partitioning, to recognize the parts of the graph corresponding to tables in the sheet. The search for tables is guided by an objective function, which is tuned to match the specific characteristics of a given dataset. We present the feasibility of this approach with an experimental evaluation, on a large, real-world spreadsheet corpus.**

***Index Terms*—Spreadsheet, Table Recognition, Graph Model, Graph Partitioning, Genetic Search, Weight Tuning**

I. Introduction

Spreadsheets are powerful content generation tools, assisting novices and professionals alike. They contain data that are roughly relational, but accompanied by various formatting, layout, and textual metadata. Thus, spreadsheet contents are designed primarily for human consumption, and often carry implicit information. Due to these reasons, automatic table recognition in spreadsheets has been a challenging task.

Particularly hard are the cases where multiple tables co-exist with other non-table structures (such as lists and comments). All together, these structures can be arranged in a variety of ways within the sheet, increasing further the complexity. In addition, we need to take into account the presence of formatting and stylistic artifacts. For instance, the structure of the individual tables can be distorted by empty cells, which are used for visual padding or implicit/missing values.

Current solutions to these challenges, such as [1]–[3], are mostly heuristics-based and limited by the imagination of the domain experts creating the rules. Furthermore, these rules need to be manually adjusted, before being utilized (transferred) in other datasets. Therefore, in this paper we propose a mostly automatic approach, which is capable of identifying near-optimum solutions, based on previously seen examples (i.e., training on a sample of annotated sheets).

In detail, we propose an approach that adopts a graph model to represent the contents of a sheet. Given this, the identification process can be formulated as a search for the optimal partitioning of the input graph, in which the resulting parts correspond exactly to the tables of the sheet. We define an objective function to determine the merit of candidate partitionings. This function is tuned to match the characteristics of a given dataset. Finally, we use genetic algorithms [4], [5] to search for the global optimum partitioning.

The subsequent parts of this paper are organized as follows: We discuss the related work in Section II. Section III introduces concepts used throughout this paper. In Section IV, we formally define the proposed table recognition approach. Our experimental evaluation is outlined in Section V. We conclude this paper in Section VI.

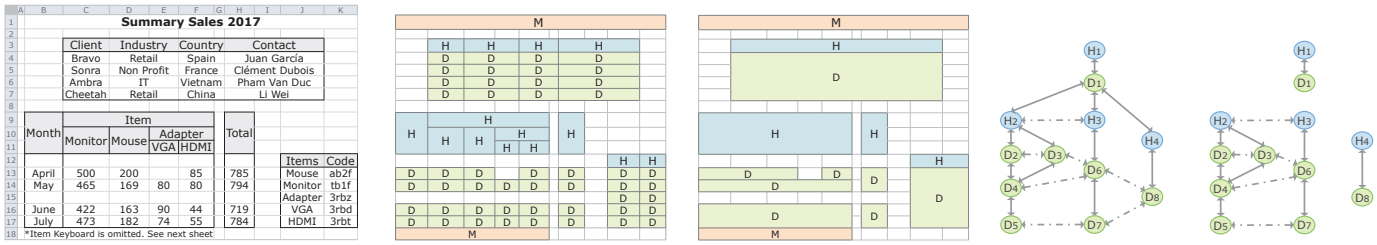
II. Related Work

There is a considerable number of works tackling layout inference and table recognition in spreadsheets Here, we mention those that are rule- and heuristics-based, like [2], [3], [6]. Other works make use of domain specific languages, such as [1], [7]. Recent publications apply to large extent machine learning techniques [8]–[11], but the overall approach still dependents on heuristics and/or generic assumptions. Instead, we avoid any assumption with regards to the number of tables, formatting, and arrangements in the sheet. Moreover, to the best of our knowledge, we are the first to employ genetic search and graph partitioning techniques for table identification in spreadsheet documents.

It is worth mentioning works proposing methods dealing with formula debugging in spreadsheets. At [12]–[14], similar to us, individual cells and regions of cells are examined, with the purpose of detecting logical errors in formulas.

The surveys [15], [16] provide a comprehensive summary of table recognition techniques, for different document formats. Authors of [17] discuss the use of genetic algorithms for image enhancement and segmentation. Also, we single out [18],

This work is supported by the German Federal Ministry of Education and Research (BMBF, 01IS14014A-D), by funding the competence center for Big Data "ScaDS Dresden/Leipzig"



(a) Original sheet (b) Labeled Cells (c) Label Regions (d) Sheet Graph (e) Table Subgraphs

Fig. 1: Building the Graph Representation

which proposes a method for table recognition in document images. It utilizes a graph model similar to the one in this work, but the rest is based on rules and heuristics.

Finally, in literature there is a large body of publications discussing graph partitioning. The surveys [19], [20] provide an extensive review of existing techniques. At [21], they focus entirely on the use of genetic algorithms for graph partitioning.

III. Preliminaries

A. Label Regions

This work utilizes the concept of *Label Region* (LR) [2], [3], which is a strictly rectangular area of a sheet, enclosing adjacent non-empty cells, having the same layout role (label).

Figures 1 a-c illustrate the creation of LRs. We ensure that each non-empty cell has a label and it belongs to one and only one LR. Here, we use three labels: *Header* (H), *Data* (D), and *Metadata* (M). Headers are the titles of the columns, describing the values below them. Metadata cells provide additional information about the sheet as a whole, or about a subset of its values. Some typical examples are titles, footnotes, and comments. Lastly, Data cells are the basic unit of information. Unlike the other two roles, Data cells do not describe values other than the ones they themselves hold.

The label of a cell can be determined using a classifier [9]–[11]. Regardless of the algorithm, the cited works make use of cell features such as style, formatting, content type, text properties, formula references. On top of that, it is possible to analyzing the nearby cells, to capture the surrounding context.

B. Graph Representation

At [3] we introduced a graph representation that captures the spatial interrelations between the label regions (LRs) of a sheet. In this work, we make use of the same representation, but apply a different approach.

The nodes of the graph correspond to the LRs of the sheet. Moreover, each node is annotated with features from the corresponding LR, such as its label and coordinates (i.e., the rows and columns that the region spans). Edges are introduced for neighboring LRs that span the same column/s or row/s, i.e., they partially or completely align, either horizontally or vertically. Furthermore, these edges are enriched with additional attributes, such as the direction of alignment and the distance between the label regions. Figure 1.d illustrates the graph representation for the example sheet on the left. Edges connecting vertically aligned regions are dashed, while those horizontally aligned are solid.

As can be seen in Figures 1.d-e, Metadata regions are omitted. These regions, unlike Header and Data, can relate to one or more tables. Understanding these relations requires an analysis that goes beyond structure (which is the scope of this work). Therefore, we do not consider Metadata regions in the resulting graph model. Moreover, we omit them for the rest of this paper, and we treat them as future work.

IV. Proposed Approach

As outlined by [19] and [20], graph partitioning and clustering are well studied problems with many applications. Using the representation described in Section III-B, we formulate the task of table recognition in spreadsheets (TRS) as a graph partitioning problem (GPP).

The input for TRS-GPP is an undirected (sheet) graph $G = (V, E)$. Here, V is the set of nodes (label regions), and E is the set of edges (spatial relations). We partition V into disjointed subsets, where $V_1 \cup ... \cup V_k = V$ s.t. $V_i \cap V_j = \emptyset$ for all $i \neq j$. Typically, the number of partitions k for GPP is fixed in advance [19]. This is not feasible in our case, since we are not aware of the number of tables in the sheet, beforehand. Thus, in this work, k can take any of the values in $\{1, ..., |V|\}$.

Intuitively, the overall goal of TRS-GPP is to find the partitioning that uniquely and precisely distinguishes the true tables of a sheet. We search for this on the basis of an objective function (see Section IV-C), measuring the quality of any arbitrary partitioning (i.e., a candidate solution). The search procedure attempts to find the global optimum, which is the solution with the best quality.

A. Header Groups

Before proceeding with the quality metrics, we introduce the concept of *Header group*, denoted as $\mathcal{H}$. Due to layout artifacts, such as empty cells, a single table may contain multiple Header regions (e.g. the bottom-left table, in Figure 1). Therefore, we create distinct groups of Headers, using the Data rows as separators. In other words, Headers end up to the same group, unless there is a data row between them.

We define the function `hgps`, which returns Header groups for any arbitrary partition V_i. Clearly, if the partition does not contain Headers, this function will return an empty set. In other cases, we might get multiple groups, which we handle as described in the following section.

B. Quality Metrics

In this section, we propose a set of metrics, to measure the quality of a partitioning $P = \{V_1, ..., V_k\}$. Intuitively, this quality is directly related to the partitions that compose P. Therefore, we define metrics that capture the appropriate characteristics (related to being a table or not) for a partition $V_i \in P$. We formulate these metrics such that they measure negative properties, since we later use minimization to identify the optimal solution. This means, the lower the metrics' values are, the more table-like are the partitions.

Below, we provide the formal definition of the proposed metrics. We use the terms region and node interchangeably, to refer to the original label regions (LRs). Additionally, we make use of the following functions: $data$ and $heads$, respectively, to get the Data and Header nodes of a partition, $area$ to get the number of cells in a node (region), $rows$ and $cols$ to get the set of row and column indices that the region covers, $cwidth$ to get the width of a column given its index, and $rheight$ to get the height of a row given its index.

M1-2. *Negative Data/Header Alignment Ratio (ndar/nhar):* For a partition V_i, we measure the horizontal alignment of Data with the top Header group ($\mathcal{H}^{top}$). The latter has the smallest row index, among all $\mathcal{H}$ in V_i.

For Data and $\mathcal{H}^{top}$, the alignment ratio is high, when they share most of their column indices. We inverse this measurement, to capture the negative cases. Thus, the closer the value is to one, the lower is the alignment. In the equation below, *ndar* measures the ratio from the perspective of the Data regions. Thus, we divide by $|C^d|$.

$$C^{ht} = \bigcup_{v \in \mathcal{H}^{top}} \texttt{cols}(v), \qquad C^{d} = \bigcup_{u \in \texttt{data}(V_i)} \texttt{cols}(u),$$

$$ndar = \begin{cases} 1 - \frac{|C^{ht} \cap C^{d}|}{|C^{d}|}, & if\ |C^{d}| \geq 1\ and\ |C^{ht}| \geq 1 \\ 0, & otherwise \end{cases}$$

We use *nhar* to measure the alignment from the perspective of $\mathcal{H}^{top}$. Therefore, in this case, we would replace the fraction below with: $1 - \frac{|C^{ht} \cap C^{d}|}{|C^{ht}|}$.

M3-4 *Is Data/Header Partition (dp/hp):* Notice that, for the previous two metrics, we omit the cases where the partitions contain either Data or Header, but not both. For these cases we introduce two separate boolean metrics. Here, we illustrate the calculation of *dp*. We handle *hp*, similarly.

$$dp = \begin{cases} 1 & if\ \texttt{heads}(V_i) = \emptyset\ and\ |\texttt{data}(V_i)| \geq 1 \\ 0, & otherwise \end{cases}$$

M5. *Is All In One Column (ioc):* This is another boolean metric. If $\mathcal{H}^{top}$ and Data cover altogether only one column, ioc returns 1, otherwise 0. This metric pushes towards tables that span at least two columns.

$$ioc = \begin{cases} 1, & if\ |C^{d}| = 1\ and\ |C^{ht}| = 1\ and\ |C^{d} \cap C^{ht}| = 1 \\ 0, & otherwise \end{cases}$$

M6. *Count Other Valid Headers (#ovh):* Besides $\mathcal{H}^{top}$, there might be other *valid* Header groups, in the partition. We consider those valid that (cumulatively) span more than one column. The presence of other valid $\mathcal{H}$ suggests multiple tables in the same partition.

$$\#ovh = \left|\{\mathcal{H} \in \texttt{hgps}(\texttt{V}_\texttt{i}) \setminus \mathcal{H}^{top} : \ \Big|\bigcup_{v \in \mathcal{H}} \texttt{cols}(v)\Big| \geq 2\}\right|$$

M7. *Data Above Header Ratio (dahr)*: Measures the portion of Data cells found above $\mathcal{H}^{top}$ in a given partition. In other terms, we identify Data cells with a row index less than $\min \bigcup_{v \in \mathcal{H}^{top}} \texttt{rows}(v)$. Intuitively, for typical tables, it is expected that all Data cells are below $\mathcal{H}^{top}$. However, for arbitrary partitions, especially in multi-table sheets, Data cells could be found above it.

$$dahr = \frac{\#dcells\_above}{\sum_{u \in \texttt{data}(V_i)} \texttt{area}(u)}$$

M8. *Average Width for Adjacent Empty Columns (avg_waec):* We group adjacent empty columns, measure the commutative width[1] per group, and subsequently average the widths. In the equation below, we denote as $\mathcal{C}^{emt}$ the list containing these (*aec*) groups. Intuitively, this metric identifies empty columns that act as separators of content. This could imply that some nodes of the partition do not belong with others.

$$avg\_waec = \frac{\sum_{i=1}^{|\mathcal{C}^{emt}|} \sum_{j=1}^{|\mathcal{C}_i^{emt}|} \texttt{cwidth}(\mathcal{C}_{ij}^{emt})}{|\mathcal{C}^{emt}|}$$

M9. *Average Height for Adjacent Empty Rows (avg_waer):* The presence of empty rows, similar to empty columns, could imply separation of contents.

$$avg\_waer = \frac{\sum_{i=1}^{|\mathcal{R}^{emt}|} \sum_{j=1}^{|\mathcal{R}_i^{emt}|} \texttt{rheight}(\mathcal{R}_{ij}^{emt})}{|\mathcal{R}^{emt}|}$$

M10. *Overlap Ratio (ovr):* Unlike the other metrics, this one is calculated at partitioning level, P. We identify overlaps between the individual partitions composing P, and measure their area. We divide the sum of overlaps with the used area of the sheet (i.e. the minimum bounding box enclosing all regions). Below, C_i and C_j represent the sets of column indices for partition V_i and V_j, respectively. Equivalently, for row indices we use R_i and R_j.

$$ovr = \frac{\sum_{i=1}^{|P|-1} \sum_{j=i+1}^{|P|} |C_i \cap C_j| * |R_i \cap R_j|}{\left|\bigcup_{v \in V} cols(v)\right| * \left|\bigcup_{v \in V} rows(v)\right|}$$

C. Objective Function

In Equation 1, we define the function measuring the fitness of the individual partitions that comprise a candidate partitioning $P = \{V_1, ..., V_k\}$.

$$\texttt{fit}(V_i, M, \mathbf{w}) := \sum_{j=1}^{9} M_j(V_i) * \mathbf{w}_j \tag{1}$$

As shown, the fitness for a partition V_i is calculated as a weighted sum of metrics' values. With M we denote the

[1] In Excel, column width is measured in units of 1/256th of a standard font character width, while row height is measured in points

list of implemented metrics. In Equation 1, we use only the first nine metrics, since these apply at the partition level (as outlined in Section IV-B). While $\mathbf{w}$ is a vector that holds the corresponding weights for the metrics.

$$\texttt{obj}(P, M, \mathbf{w}) := \mathbf{w}_{10} * M_{10}(P) + \sum_{V_i \in P} \texttt{fit}(V_i, M, \mathbf{w}) \quad (2)$$

Equation 2 provides the definition for the objective function. We sum up the fitness of the individual partitions. Moreover, we make use of the *Overlap Ratio* metric (M_{10}), which is calculated at the level of the partitioning.

D. Weight Tuning

One of the challenges of the proposed approach is determining the optimal weights for the objective function (see Equation 2). Intuitively, some metrics are more crucial than others. However, it is rather difficult to manually ascertain the exact importance of each metric in relation to the rest. Therefore, optimization algorithms are needed to tune the weights automatically. Here, we make use of Sequential Quadratic Programming (SQP) [22], for constrained minimization.

1) Tuning Sample: We tune the weights based on a sample drawn from a dataset of sheet graphs (see Section V-A). The intention is to guide the optimization algorithm in finding weights that favor valid partitions, while penalizing false ones. Therefore, for each one of the selected graphs, we determine the target partitioning, which is the one that corresponds to the true tables in the sheet. Furthermore, we randomly generate multiple alternative partitionings (i.e., false instances).

2) Tuning Function: We denote the sample used for optimization as $\mathcal{S} = (\mathcal{U}, \mathcal{T})$, where $\mathcal{U}$ holds all the generated alternative partitionings, and $\mathcal{T}$ holds the corresponding target partitionings (i.e., a one-to-one mapping).

$$\arg\min_{\mathbf{w}} \sum_{j=1}^{|\mathcal{U}|} \frac{1 + obj(\mathcal{T}_j, M, \mathbf{w})}{1 + obj(\mathcal{U}_j, M, \mathbf{w})}, \text{such that: } 0 \leq \mathbf{w} \leq 10^3$$

The equation above defines the function used for weight tuning. As can be seen, it is a summation of fractions, designed for minimization. In most of the fractions the numerators will get smaller values than the denominators (as mentioned in Section IV-B, we measure negative properties). Regardless, we still need to increase the gap between target and alternatives. The challenge is to find weights that penalize alternatives, without affecting many targets.

Note that we add $+1$ to denominators and numerators, in order to avoid exceptions of division by zero, during the automatic search for optimal weights. Furthermore, we constrain the possible values for the weights in the interval $[0, 10^3]$. This is to avoid extremely large or extremely low weights, which might not be realistic, but rather reflecting the peculiarities of the current sample.

E. Genetic Search

Given Equation 2, we are able to identify the fittest partitioning for each input sheet-graph. For small graphs (currently set to ≤ 10 nodes), we perform exhaustive search. However, for larger graphs a more efficient mechanism is required. For these cases, we use genetic algorithms [4], [5], which can yield near optimum solutions in reasonable time. In this work, we make use of the *edge encoding* as described in [21]. We represent the edges of an input graph with a boolean valued list of size $|E|$. When the corresponding value is set to true, the edge is activated, otherwise not. Dis-activating and re-activating edges leads to various graph partitionings. Intuitively, *edge encoding* ensures that we always get connected components of the input graph. This is favorable, since it translates to partitions that enclose neighboring regions, rather than arbitrary ones.

Nevertheless, there are challenges to this formulation. The search space increases exponentially with the number of edges $2^{|E|}$. Thus, identifying the right combination of edges becomes more demanding for larger graphs. We find such graphs (up to $7,484$ edges) in our dataset. They occur due to many implicit/missing values (empty cells) in tables, which inflate the number of nodes (LRs), and with that the number of edges.

The pseudocode at Algorithm 1 describes the implemented genetic search. An initial population, composed of boolean-valued lists, is generated randomly. The size of this population is not fixed, but rather calculated with a function, which takes into account the number of edges in the input graph. In subsequent iterations, new individuals (*Children*) are created from the population of the previous generation (*Parents*). This step is performed using one of the following genetic operations: *random mutation* (inverts boolean values of a Parent, with an independent probability *indpb* = 0.1), and *uniform crossover* (combines values from two Parents, with *indpb* = 0.5). We pick individuals for the next generation with *tournament selection* of size = 3, considering both Children and Parents. The *hallOfFame* (hof) carries the individual having the smallest score (i.e., the fittest) among all examined candidates.

We select genetic operators and parameter values following recommended practices [23]. After extensive experimentation, we favored those that push towards more diverse generations (Children). In this way, we cover a larger search space and decrease the chances of premature convergence.

Furthermore, we note the optional *seed* individual in Algorithm 1. We adopt the heuristic approach proposed at [3], to create this seed. It represents a good candidate solution,

Algorithm 1: Identifying the Optimum Partitioning

Input: sheet-graph: $G = (V, E)$, metrics: M, weights: $\mathbf{w}$, population size: $npop = \texttt{ceil}(log_{10}(|E|) * 100)$, #generations: $ngen = 200$, crossover probability: $cxpb = 0.5$, mutation probability: $mutpb = 0.5$, #offsprings to generate: $\lambda = npop$, #individuals to select: $\mu = npop$, and $seed$ individual (optional)
Output: The partitioning with the lowest objective function score

```
1 begin
2     Pop ← createInitialPopulation(G, npop, seed);
3     hof ← updateHallOfFame(Pop, G, M, w);
4     for i ∈ {1, ..., ngen} do
5         Children ← createOffsprings(Pop, λ, cxpb, mtpb);
6         hof ← updateHallOfFame(Children, G, M, w);
7         Pop ← selectFittest(Pop ∪ Children, μ, G, M, w)
8     return hof
```

which is fed to the initial population. Such hybrid approaches are common in practice, as suggested by [24]. They often yield better results than pure genetic ones. We make $\frac{npop}{2} - 1$ copies of the seed, and apply random mutations on them, with *indpb*=0.1. These copies, together with the seed, compose half of the initial population. The rest is randomly generated.

V. Evaluation

We perform 10-fold cross-validation, with 10 weight-tuning rounds per training sample, followed by 10 runs of genetic search for test graphs with $|E| > 10$. To ensure statistical significance, we repeat the whole process three times. Each time, we shuffle the dataset using a different (numeric) seed.

A. Annotated Dataset

The considered dataset[2] is composed of files that were randomly selected from the ENRON corpus [25]. For each file, we annotate the first sheet (from left to right) containing table/s. We skip sheets having transposed tables (i.e., header on the left), and horizontally attached tables (i.e., no empty column in-between them), as being outside of our scope.

In total, we annotated 674 single-table sheets, and 140 having multiple tables (refer to Figure 2.a). Overall, this sums up to $1\,158$ annotated tables. In multi-table sheets, the arrangement is predominantly vertical (top to bottom), but we also find 34 with horizontal (left to right), and 17 having both arrangements. Moreover, we notice empty columns (173 sheets) and empty rows (364 sheets) within tables, besides those found between tables (respectively 51 and 102 sheets).

We generate two graph datasets. The first one, the gold standard, contains the graphs corresponding to the annotated sheets. The second one is a modified dataset, used only for training, where 0.1% (min 1 cell) of the cells in the annotated sheets are randomly re-labeled or omitted (i.e., induced noise).

Note that in this work we do not consider cell classification, but rather work directly with the annotations. Furthermore, we omit label regions (LRs) found outside of annotated tables, when generating graph representations (refer to Section III-B).

B. Cross-Validation Folds

Having these datasets, we assess the performance via a 10-fold cross-validation (CV). We ensure that multi- and single-table graphs are balanced among the folds. With regards to training, the modified dataset is an additional option. Each iteration of the CV we train on the corresponding graphs from the modified dataset, instead of the ones from the gold standard. Nevertheless, testing still happens on the original left-out fold. Intuitively, the second approach exposes the tuning mechanism to small irregularities, to avoid overfitting.

C. Tuning Rounds

Before tuning the weights, we identify the target and generate alternative partitionings for the training graphs (see Section IV-D). We limit the number of alternative partitionings to $10 * \#tables$, per graph. This balances the contribution of multi-table and single-table graphs in the tuning sample.

Furthermore, we ensure reliable weights by performing 10 rounds of tuning, each time generating new alternative partitionings, from the training graphs. The weights, resulting from different rounds, are averaged. For this, we consider the error rate of each round, measured as the portion of alternative partionings having a lower obj function value than the corresponding targets. Altogether, rounds with higher error rates contribute less to the averaged weights.

D. Search and Assessment

Figure 2.b shows the distribution of edge counts $|E|$, for graphs in the gold standard. There are 380 (47%) graphs with $|E|$ in $(0, 10]$, for which we use exhaustive search. For the rest, $|E| > 10$, we apply genetic search.

While exhaustive search is deterministic, the genetic one can return different candidate solutions, on multiple runs, for the same input graph. Therefore, we perform genetic search 10 times, and then average the accuracy of the results.

The accuracy is assessed by comparing the target partitioning to the output (hof). For a pair, annotated (target) table and hof partition, we calculate the agreement. We measure this using the *Jaccard index* (i.e., #cells in common over #cells in union). Subsequently, we identify the best match (highest Jaccard index) for an annotated table. We mark as recognized tables for which we find an agreement of ≥ 0.9. Note that we consider from hof only partitions having both Data and Header nodes. The rest we regard as non-valid candidates.

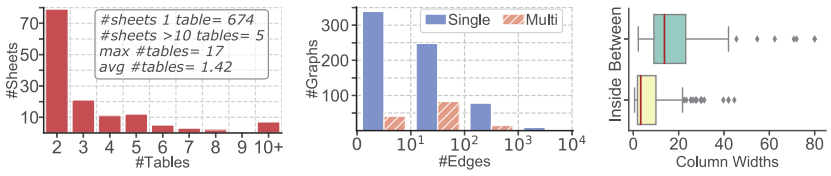
E. Evaluation Results

Table I summarizes the cross-validation results, which are averaged for three runs (i.e., each time re-shuffling the dataset). We start by discussing the first two approaches, heuristic (H) and genetic-exhaustive (GE) search. The former has higher accuracy with multi-table sheets, as it was designed for more generic cases. For example, the heuristic approach treats all empty columns as table separators, which is favorable when multiple tables are arranged horizontally. Nevertheless, GE performs significantly better on single-table sheets, since it anticipates occasional irregularities inside these tables (e.g., missing values, empty columns/rows). At the same time, this flexibility is disadvantageous for some multi-table sheets.

Besides that, our analysis revealed that the accuracy of GE depends on the number of edges. Specifically, we determined that GE achieves an accuracy of only 19% for multi-table graphs with $|E| > 100$. However, for smaller such graphs, with $10 < |E| \leq 100$ and $|E| \leq 10$, the accuracy is notably higher, at 81% and 97% respectively.

The introduction of noise (GE+N), during the tuning phase, has a slightly positive effect for single-table sheets, but also slightly negative for multi-table ones. We observe that the induced noise leads to lower weights for *"count other valid headers"* metric (M6), which causes some true Headers to be interpreted as non-valid. Particularly, we notice a decrease in accuracy for sheets with multiple vertically arranged tables.

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



(a) Multi-Table Sheets (b) Edge Counts (c) Horizontal Gaps

Fig. 2: Analysis of the Evaluation Results

TABLE I: % of Recognized Tables

	ALL	SINGLE	MULTI
H	85.7	86.9	83.9
GE	84.1 ± 0.4	92.8 ± 0.1	72.2 ± 1.2
GE+N	84.3 ± 0.4	93.72 ± 0.2	71.3 ± 0.8
GE+H	89.0 ± 0.3	90.7 ± 0.2	86.6 ± 0.6
GE+N+H	89.6 ± 0.1	91.5 ± 0	87.0 ± 0.3

Feeding GE search with a heuristic *seed* leads to a considerably higher number of recognized tables. This hybrid method (GE+H) overcomes the drawbacks of the individual approaches, to achieve noticeably better results for multi-table sheets. Specifically, GE+H has an accuracy of 71%, 91%, and 97%, respectively in multi-table graphs with $|E| > 100$, $10 < |E| \leq 100$, and $|E| \leq 10$. Nevertheless, the *seed* can be misleading in some cases, as we see the accuracy decreasing for single-tables. To this aids the induced noise (GE+N+H), improving the result for single-table sheets and overall.

In addition to the aforementioned, the heuristic *seed* speeds up the genetic search. GE+H requires on average 4 generations to find the "best" candidate (i.e., *hof* does not change afterwards) compared to 13 for GE. About 70% of the GE+H runs determine *hof* from the first generation, since the seed corresponds to the best candidate. For the rest, the search proceeds to find other partitionings with lower (better) score.

We conclude this discussion, with an analysis of horizontal gaps (i.e., adjacent empty columns), which we determined to be the main factor for the remaining unrecognized tables. We studied the width distribution for such gaps, inside and between tables (in Figure 2.c, outliers >80 omitted). There is a notable overlap between these distributions, suggesting that width alone is not informative enough to recognized the true horizontal table separators. In future works, we will consider additional metrics to better capture the circumstances in the sheet, allowing the approach to make more informed decisions.

VI. Conclusions

In this paper, we present our novel approach for table recognition in spreadsheets. Unlike previous works, our method requires little to no involvement of domain experts. Instead, it adapts to specific spreadsheet datasets, based on a sample of annotated files. We represent each sheet with a graph model, which carries layout and spatial information about the contents. On top of this, we define metrics to capture the underlying table characteristics. These metrics become an integral part of an objective function, which is tuned to favor tables similar to the provided examples. Subsequently, in the remaining sheets of the dataset, we recognize tables by partitioning the corresponding graph representation. The search is performed either exhaustively or with the help of a genetic algorithm, depending on the size of the graph.

References

[1] 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.

[2] E. Koci, M. Thiele, O. Romero, and W. Lehner, "Table identification and reconstruction in spreadsheets," in *CAiSE'17*, pp. 527–541.

[3] E. Koci, M. Thiele, W. Lehner, and O. Romero, "Table recognition in spreadsheets via a graph representation," in *DAS'18*, pp. 139–144.

[4] C. M. Anderson-Cook, "Practical genetic algorithms," 2005.

[5] D. E. Goldberg, *Genetic algorithms*. Pearson Education India, 2006.

[6] 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'13*, pp. 2477–2480.

[7] F. Hermans, M. Pinzger, and A. Van Deursen, "Automatically extracting class diagrams from spreadsheets," *ECOOP'10*, pp. 52–75.

[8] 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.

[9] 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.

[10] Z. Chen and M. Cafarella, "Automatic web spreadsheet data extraction," in *International Workshop on Semantic Search over the Web*, 2015, p. 1.

[11] M. D. Adelfio and H. Samet, "Schema extraction for tabular data on the web," *VLDB'16*, pp. 421–432.

[12] R. Abraham and M. Erwig, "Header and unit inference for spreadsheets through spatial analyses," in *Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2004, pp. 165–172.

[13] 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.

[14] R. Singh, B. Livshits, and B. Zorn, "Melford: Using neural networks to find spreadsheet errors," Tech. Rep., January 2017.

[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. W. Embley, M. Hurst, D. Lopresti, and G. Nagy, "Table-processing paradigms: a research survey," *International Journal of Document Analysis and Recognition (IJDAR)*, vol. 8, no. 2-3, pp. 66–86, 2006.

[17] M. Paulinas and A. Ušinskas, "A survey of genetic algorithms applications for image enhancement and segmentation," *Information Technology and control*, vol. 36, no. 3, 2007.

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

[19] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz, "Recent advances in graph partitioning," in *Algorithm Engineering*. Springer, 2016, pp. 117–158.

[20] S. E. Schaeffer, "Graph clustering," *Computer science review*, vol. 1, no. 1, pp. 27–64, 2007.

[21] J. Kim, I. Hwang, Y.-H. Kim, and B.-R. Moon, "Genetic approaches for graph partitioning: a survey," in *GECCO'11*, pp. 473–480.

[22] P. T. Boggs and J. W. Tolle, "Sequential quadratic programming," *Acta numerica*, vol. 4, pp. 1–51, 1995.

[23] A. E. Eiben and S. K. Smit, "Evolutionary algorithm parameters and methods to tune them," in *Autonomous search*. Springer, 2011, pp. 15–36.

[24] C. Grosan and A. Abraham, "Hybrid evolutionary algorithms: methodologies, architectures, and reviews," in *Hybrid evolutionary algorithms*. Springer, 2007, pp. 1–17.

[25] F. Hermans and E. Murphy-Hill, "Enron's spreadsheets and related emails: A dataset and analysis," in *ICSE*. IEEE Press, 2015, pp. 7–16.