Semantic Table Structure Identification in Spreadsheets

Yakun Zhang
State Key Lab of Computer Sciences, Institute of Software, Chinese Academy of Sciences
University of Chinese Academy of Sciences, Beijing, China
zhangyakun18@otcaix.iscas.ac.cn

Xiao Lv
Microsoft Research
Beijing, China
xilv@microsoft.com

Haoyu Dong
Microsoft Research
Beijing, China
hadong@microsoft.com

Wensheng Dou[*]
State Key Lab of Computer Sciences, Institute of Software, Chinese Academy of Sciences
University of Chinese Academy of Sciences, Beijing, China
wsdou@otcaix.iscas.ac.cn

Shi Han
Microsoft Research
Beijing, China
shihan@microsoft.com

Dongmei Zhang
Microsoft Research
Beijing, China
dongmeiz@microsoft.com

Jun Wei
State Key Lab of Computer Sciences, Institute of Software, Chinese Academy of Sciences
University of Chinese Academy of Sciences, Beijing, China
wj@otcaix.iscas.ac.cn

Dan Ye
State Key Lab of Computer Sciences, Institute of Software, Chinese Academy of Sciences
University of Chinese Academy of Sciences, Beijing, China
yedan@otcaix.iscas.ac.cn

ABSTRACT

Spreadsheets are widely used in various business tasks, and contain amounts of valuable data. However, spreadsheet tables are usually organized in a semi-structured way, and contain complicated semantic structures, e.g., header types and relations among headers. Lack of documented semantic table structures, existing data analysis and error detection tools can hardly understand spreadsheet tables. Therefore, identifying semantic table structures in spreadsheet tables is of great importance, and can greatly promote various analysis tasks on spreadsheets.

In this paper, we propose *Tasi* (Table structure identification) to automatically identify semantic table structures in spreadsheets. Based on the contents, styles, and spatial locations in table headers, Tasi adopts a multi-classifier to predict potential header types and relations, and then integrates all header types and relations into consistent semantic table structures. We further propose TasiError, to detect spreadsheet errors based on the identified semantic table structures by Tasi. Our experiments on real-world spreadsheets show that, Tasi can precisely identify semantic table structures in spreadsheets, and TasiError can detect real-world spreadsheet errors with higher precision (75.2%) and recall (82.9%) than existing approaches.

CCS CONCEPTS

• **Applied computing** → **Spreadsheets**; • **Software and its engineering** → *Software testing and debugging*.

KEYWORDS

Spreadsheet, table structure, error detection

ACM Reference Format:
Yakun Zhang, Xiao Lv, Haoyu Dong, Wensheng Dou, Shi Han, Dongmei Zhang, Jun Wei, and Dan Ye. 2021. Semantic Table Structure Identification in Spreadsheets. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '21), July 11–17, 2021, Virtual, Denmark.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3460319.3464812

1 INTRODUCTION

Spreadsheets contain amounts of valuable data, and have been widely used in various business tasks, e.g., data storage, data analysis, and financial reporting [36]. Existing studies show that there were over 55 million users working with spreadsheets in America in 2012 [43], and more than 50% businesses use spreadsheets [5].

In spreadsheets, data and formulas are organized in a two-dimensional structure. Spreadsheet systems, e.g., Microsoft Excel, provide

[*] Wensheng Dou is also affiliated with Institute of Software Technology, Chinese Academy of Sciences, Nanjing, China. Wensheng Dou is the corresponding author.

ISSTA '21, July 11–17, 2021, Virtual, Denmark
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8459-9/21/07…$15.00
https://doi.org/10.1145/3460319.3464812

flexible methods for users to prepare their spreadsheets. This causes spreadsheet data are usually organized in a semi-structured way.

In a spreadsheet table, headers are used to describe other cells, and contain the table's key structure information. For example, in Figure 1a, "January" in header C2, "Cost" in header C3 and "America" in header B4 together indicate that cell C4 shows the cost in America in January. Note that, headers are usually used for different purposes, thus having different types. For example, header C3 in Figure 1a indicates that cells [C4:C10] show the costs, and headers [C2:E2] index the costs for different months. Further, there exist various relations among headers, e.g., header C1 is the index name of header C2, and header A4 is the parent of header B4. More details about header types and header relations can be found in Section 2.1. In a spreadsheet table, the header types and header relations can represent its semantic table structure.

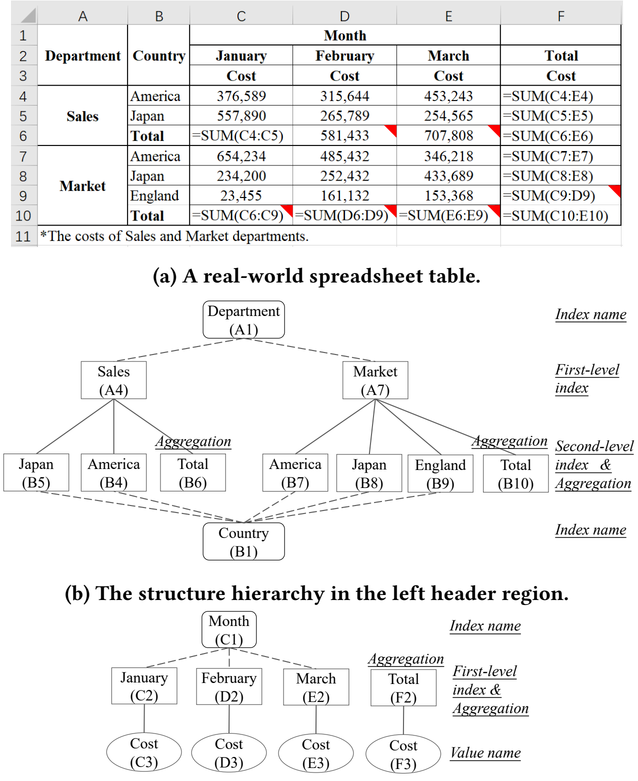
Given a spreadsheet table, human users can easily understand the semantic table structure behind spreadsheet data. However, there are no records documenting the semantic table structures in spreadsheet systems, e.g., Microsoft Excel. It is also challenging to automatically identify semantic table structures from semi-structure spreadsheet data. Existing approaches [15, 18, 21, 22, 28, 29, 34] can only detect certain *coarse-grained* spreadsheet structures, e.g., table regions in TableSense [18], cell types (i.e., header, data, attribute, derived, metadata) in Koci et al. [34], and expandable groups (i.e., similar data and computations) in ExpCheck [21]. These approaches cannot identify the purposes of headers, i.e., *fine-grained* header types in Section 2.1. Further, various header relations are rarely addressed by existing approaches. Chen et al. [15] can only extract one relation among headers, i.e., parent-child relation.

Identifying semantic table structures in spreadsheets is the fundamental step to many spreadsheet analysis tasks. First, spreadsheets contain various errors [39, 40]. We can use semantic table structures in spreadsheets to identify spreadsheet errors. For example, if some cells' computational semantics are inconsistent with their semantic table structures, they may contain errors [8, 19, 22]. Second, by understanding semantic table structures, we can transform the semi-structured data in spreadsheets into relational data. Thus, intelligent data analysis tools, e.g., PowerBI [4] and Ideas in Excel [3] can work well on the relational form of spreadsheet data.

In this paper, we propose *Tasi* (*Ta*ble *s*tructure *i*dentification) to identify semantic table structures in spreadsheets. We observe that the contents, styles and spatial locations in spreadsheet headers can help us precisely identify semantic table structures. In Tasi, we group each two headers in a header pair, and carefully craft 46 features from the contents, styles and spatial locations for each header pair, which can reflect its header types and relation. We adopt a multi-classifier to predict the header types and relation for each header pair. Finally, Tasi integrates the structures of all header pairs into a consistent semantic table structure.

We train Tasi on 2,651 spreadsheet tables and evaluate it on 639 real-world spreadsheet tables. The experimental results show that Tasi can identify semantic table structures effectively. Tasi can correctly identify 75.2% of header types and relations among headers. Specially, for 639 spreadsheet tables, Tasi can identify their semantic table structures in 282 (44.1%) tables without errors.

In order to demonstrate the usefulness of semantic table structures, we further propose TasiError, which can utilize the semantic table structures, and automatically detect two kinds of spreadsheet errors, i.e., missing formulas and formula errors. We evaluate TasiError on the 70 real-world spreadsheets in the CUSTODES dataset [16] with 3,702 errors. TasiError can detect 3,068 (82.9%) errors with a precision of 75.2%, from which 1,172 (31.7%) errors cannot be detected by existing approaches [1, 8, 11, 14, 16, 20, 22].



	A	B	C	D	E	F
1	**Department**	**Country**	**Month**			
2			**January**	**February**	**March**	**Total**
3			**Cost**	**Cost**	**Cost**	**Cost**
4	**Sales**	America	376,589	315,644	453,243	=SUM(C4:E4)
5		Japan	557,890	265,789	254,565	=SUM(C5:E5)
6		**Total**	=SUM(C4:C5)	581,433	707,808	=SUM(C6:E6)
7	**Market**	America	654,234	485,432	346,218	=SUM(C7:E7)
8		Japan	234,200	252,432	433,689	=SUM(C8:E8)
9		England	23,455	161,132	153,368	=SUM(C9:D9)
10		**Total**	=SUM(C6:C9)	=SUM(D6:D9)	=SUM(E6:E9)	=SUM(C10:E10)
11	*The costs of Sales and Market departments.					

(a) A real-world spreadsheet table.

(b) The structure hierarchy in the left header region.

(c) The structure hierarchy in the top header region.

Figure 1: A spreadsheet table extracted from real-world spreadsheets. In Figure 1a, the cells marked by a red right-cornered triangle contain errors.

In summary, we make the following contributions in this paper.

- We propose a learning-based approach, Tasi, to identify semantic table structures in spreadsheets, including header types and header relations.
- We propose TasiError, which can utilize semantic table structures to effectively and precisely detect spreadsheet errors.
- We implement Tasi and TasiError, and evaluate them on real-world spreadsheets. The experimental results show that our approaches perform well in practice.

2 SEMANTIC TABLE STRUCTURE AND ITS APPLICATIONS

A spreadsheet *table* is a rectangular block of cells, which prescribes certain business task [18]. For example, cells [A1:F11] in Figure 1a form a table for storing and analyzing the costs of two departments in a company, i.e., Sales and Market.

A spreadsheet table usually contains two parts [45, 47]. (1) *Header* cells describe other cells in a table. In table [A1:F11] in Figure 1a, cells [A1:B11] and cells [C1:F3] are header cells. Through header A4[1], B4, C1, C2 and C3, we can know 376,589 means the cost of the Sales department in America in January. (2) *Data* cells describe the business data in a table. For example, in Figure 1a, cells [C4:F10] are data cells.

In a spreadsheet table, headers are usually located in the left columns and on the top rows, e.g., cells [A1:B11] in the left columns and cells [C1:F3] on the top rows[2]. We use *top header region* to denote the headers on the top, and *left header region* to denote the headers in the left, and *data region* to denote the data in the table[3].

In this section, we first explain semantic table structures for a spreadsheet table, and then discuss the importance of identifying semantic table structures.

2.1 Semantic Table Structure

Given a spreadsheet table, its *semantic table structure* describes the purposes of its headers and the relations among headers. By inspecting the spreadsheet programming model and amounts of real-world spreadsheets, we summarize five header types and eight header relations as follows.

2.1.1 Header Types. In spreadsheet tables, headers are used to describe other cells and have different purposes. We divide headers into five types, i.e., value name, aggregation, index, index name and other, according to their purposes.

Value name (T_1). A value name is the summary term of related data cells in the data region. A value name can be a measure, such as number, percent and amount, and can also be a unit of measure, such as meter and mL. For example, in Figure 1c, header C3 is a value name.

Index (T_2). An index is a header for indexing data cells in the data region. For example, in Figure 1a, A4, A7, B4, and B5 are indexes. Indexes can have hierarchical structures inside. Take Figure 1b as an example. A4 and A7 are the first-level index headers, and B4 and B5 are second-level index headers in left header region.

Index name (T_3). An index name is a summary term to describe indexes in the header region. For example, in Figure 1b, header B1 is the index name for indexes [B4:B5] and [B7:B9].

Aggregation (T_4). An aggregation indicates some values are calculated from other values. A total is a special aggregation indicating the *sum* operation. For example, in Figure 1b, B6 and B10 are aggregations.

Other (T_5). Some headers in the header regions are not used to describe other cells, e.g., the comment in header A11 in Figure 1a. We categorize these headers as others, since they usually do not reflect a table's semantic structure.

2.1.2 Relations among Headers. Given a header pair $< h1, h2 >$, we summarize eight relations between $h1$ and $h2$. Note that, in a header pair $< h1, h2 >$, we require that, $h1$ is located in the left of $h2$, otherwise, $h1$ is located on the top of $h2$. In this paper, we have this requirement for all header pairs.

$h1$ is the parent of $h2$ (R_1). Headers in a top or left header region can be organized into a hierarchy, e.g., Figure 1b and Figure 1c. If $h1$ and $h2$ are both index headers, and $h1$ is the parent node of $h2$ in the hierarchy, then $h1$ is the parent of $h2$. For example, in Figure 1b, index header A4 is the parent of index header B4. Similarly, **$h2$ can be the parent of $h1$ (R_2).**

Some index headers can form an *index set*, in which all index headers are located in the same level in the hierarchy, and have the same parent if the parent exists. In the left header region in Figure 1b, index header A4 and A7 form an index set {A4, A7}. Index header B4 and B5 have the same parent A4, and form an index set {B4, B5}. In the top header region in Figure 1c, index header C2, D2 and E2 form an index set {C2, D2, E2}.

$h1$ and $h2$ are siblings (R_3). If $h1$ and $h2$ belong to the same index set, they are siblings. For example, in Figure 1b, header B4 and B5 belong to index set {B4, B5}, thus they are siblings.

$h1$ is the index name of $h2$ (R_4). If $h1$ is an index name, $h2$ is an index header, and $h1$ is used to describe $h2$, then $h1$ is the index name of $h2$. For example, in Figure 1c, header C1 is the index name of header C2. Similarly, **$h2$ can be the index name of $h1$ (R_5).**

$h1$ is the aggregation of $h2$'s index set (R_6). For example, in Figure 1a, each cell (e.g., C6) with header B6 is calculated from the corresponding cells with header B4 and B5 (e.g., C4 and C5). B6 is an aggregation header, and B4 and B5 form an index set. So, header B6 is an aggregation of header B4's index set. Similarly, **$h2$ can be the aggregation of $h1$'s index set (R_7).**

No relation of $h1$ and $h2$ (R_8). It means $h1$ and $h2$ do not have any of the above 7 relations. For example, in Figure 1a, header A1 and B1 do not have any of the above 7 relations.

Note that, the headers in the left and top header regions jointly describe relevant data cells in spreadsheets. For example, data cell C4 in Figure 1a is described by its left headers (i.e., A4 and B4) and top headers (i.e., C1, C2, and C3). This relation is different from the above relations. By combining Tasi's semantic structures and data cells' spatial locations, it is straightforward to infer this relation. Therefore, we do not discuss this relation in this paper.

2.2 Applications of Semantic Table Structures

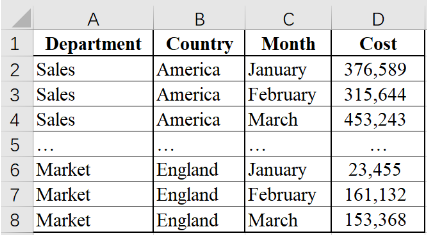
Semantic table structures can be applied on some important spreadsheet analysis scenarios. e.g., error detection, table transformation and data analysis. This motivates us to effectively identify semantic table structures in spreadsheets. We explain some potential applications of semantic table structures as follows.

Error detection in spreadsheets. Spreadsheets contain various errors [13, 32, 40]. Specially, two common errors, i.e., missing formulas and formula errors, can greatly degrade the quality of spreadsheets, causing financial losses [2, 38]. In Figure 1a, C10, D10 and E10 contain three formula errors, since their computations contain the data from Sales department. D6 and E6 in Figure 1a contain two missing formula errors, in which a cell is supposed to contain a formula, but it does not. These errors can lead to wrong data, e.g., C10, D10, E10 and F9 in Figure 1a. These wrong data may cause severe consequences.

[1] We use the upper left cell address to denote a merged cell throughout this paper.

[2] The headers in the left columns and on the top rows may overlap, e.g., A1 and B1 in Figure 1a. In this paper, we only consider the overlapping headers to be included in the left header regions.

[3] Spreadsheets may have weird header layouts, e.g., headers in the right or in the bottom. But, in practice, these weird header layouts are very rare. In our study, we find that almost all (>99%) tables only have headers in the left and on the top. Therefore, we mainly focus on the left and top headers, and ignore other weird header layouts.



	A	B	C	D
1	**Department**	**Country**	**Month**	**Cost**
2	Sales	America	January	376,589
3	Sales	America	February	315,644
4	Sales	America	March	453,243
5	…	…	…	…
6	Market	England	January	23,455
7	Market	England	February	161,132
8	Market	England	March	153,368

Figure 2: Relational table transformed from Figure 1a.

These kinds of spreadsheet errors are hard to be detected since they usually involve the knowledge of intended spreadsheet table semantics, which often requires human judgments or specifications. Note that, the Excel internal error detector [1] only reports F9 as an error correctly, but D10 and E10 as errors with wrong repair suggestions. Some approaches [16, 19, 22] identify equivalence classes based on the R1C1 format[4] and treat outliers as errors in equivalence classes. But they cannot detect errors in C10, D10 and E10, since they contain the same formula in the R1C1 format. However, these errors can be detected by analyzing semantic table structures. We further explain how to utilize semantic table structures to detect these errors in Section 4.

Table transformation. Spreadsheet data are usually organized in a semi-structured format, e.g., Figure 1a. Although human users can easily understand the structures behind spreadsheet data, data analysis tools, e.g., PowerBI [4] and Ideas in Excel [3], cannot understand the precise table structures. Based on the identified semantic table structures, e.g., header types and header relations, we can easily transform semi-structured spreadsheet data into relational data, e.g., the data in Figure 2, which these data analysis tools are good at. Thus, various intelligent analyses can be promoted on spreadsheets.

3 SEMANTIC TABLE STRUCTURE IDENTIFICATION

Figure 3 shows an overview of Tasi. Given a spreadsheet table and its header regions, e.g., left header region and top header region, Tasi first enumerates all its header pairs in each header region (Section 3.1). For a header pair $< h1, h2 >$, Tasi extracts 46 features from its contents, styles and spatial locations (Section 3.2). Then, Tasi predicts their header types and relations together by leveraging a multi-classifier, and obtains the probability of each category for each header pair $< h1, h2 >$ (Section 3.3). Finally, Tasi integrates all header pairs' prediction results, and forms a consistent semantic table structure (Section 3.4).

Note that, instead of predicting header types and header relations independently, we predict them in a header pair together. For two headers in a spreadsheet table, their header types and relations are usually correlated. For example, in Figure 1a, the type of header B1 is an index name, the type of header B4 is an index, and header B1 is the index name of header B4. Since header types and relations should be consistent, it is impossible that header B4 has another header type, e.g., value name.

For a header pair $< h1, h2 >$, we use $< h1.type, h2.type, relation (h1, h2) >$ as its category. As discussed in Section 2.1.1, we have 5 header types. For each header pair $< h1, h2 >$, we have 8 possible relations between $h1$ and $h2$ (Section 2.1.2). Therefore, we can obtain 200 (5*5*8) possible combinations for each header pair. However, not all combinations are valid in real-world spreadsheets. For example, an index header and an aggregation header cannot have the sibling relation. After we remove these invalid combinations, we finally obtain 32 valid prediction categories. That said, we have 32 prediction categories for each header pair.

3.1 Header Pair Enumeration

As discussed in Section 2, a spreadsheet table usually contains at most one top header region and at most one left header region. Given a spreadsheet table and its top and left header regions, Tasi first extracts all headers in each header region. Note that, empty cells in a header region are not considered as headers, e.g., cell F1 in Figure 1a.

For every two headers $h1$ and $h2$ (e.g., header B1 and B4 in Figure 1a) in a header region, if $h1.row < h2.row \mid\mid (h1.row = h2.row \&\& h1.column < h2.column)$, then Tasi forms a header pair $< h1, h2 >$, otherwise, Tasi forms a header pair $< h2, h1 >$. That said, each two headers can only form one pair. If there are N headers in a header region, we can obtain $N * (N - 1)/2$ header pairs. Note that, we only enumerate header pairs in each header region, and do not enumerate header pairs between the left header region and the top header region. For example, header B4 in the left header region and header C2 in the top header region do not form a header pair.

3.2 Feature Extraction

We observe that a table's semantic structure can be characterized by its headers' contents, styles and spatial locations. We incorporate and extend the features proposed by existing works [9, 24, 34], and design the features used in Tasi by trial and error. For each designed feature, if we find that it can contribute to the final result, then we select it, otherwise, we exclude it from consideration. For example, we have tried some style features, e.g., font colors and font types used in existing works [34]. However, we find that they are not helpful for Tasi. Therefore, we exclude them from consideration. Finally, for each header pair $< h1, h2 >$, we extract 38 intra-header features for $h1$ and $h2$, and 8 inter-header features between $h1$ and $h2$. For these 46 features, 36 features are newly designed for Tasi.

Note that, the feature extraction in the top header regions is similar to that in the left header regions. For clarity, we only use headers in the top header regions to explain our feature extract algorithm when possible.

3.2.1 *Intra-Header Features.*

For a header pair $< h1, h2 >$, we extract in total 38 intra-header features from contents, styles and spatial locations, including 2 common features (F#1 and F#2), and 18 features (F#3 - F#20) for each header (36 features in total).

Common features. These features relate to the header region and the table t that header pair $< h1, h2 >$ belongs to. If $< h1, h2 >$ comes from a top header region, we obtain two features: the number of rows occupied by the top header region (**F#1**), and whether table

[4] Spreadsheet systems usually have two built-in formats to represent a cell reference, A1 and R1C1 formats. In the A1 format, a cell at the x-th column and y-th row is denoted as xy, e.g., B2. In the R1C1 format, a cell at m rows below and n columns right to the current cell is denoted as R[m]C[n].

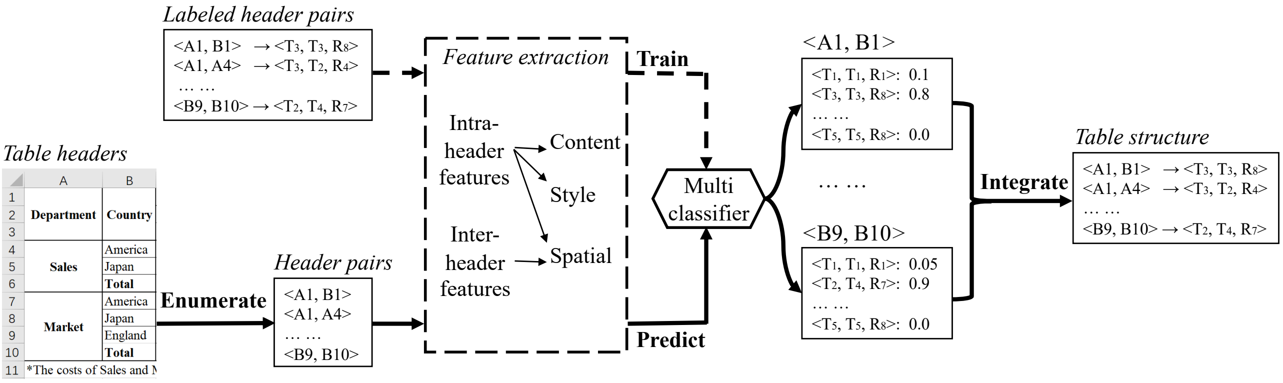


Figure 3: Overview of Tasi. T_i and R_i represent header types and relations in Section 2.1, respectively.

t has a left header region (**F#2**). These features' extraction is similar to the headers in a left header region.

Content features. For each header in a header pair $< h1, h2 >$, we extract 3 features from its content.

- Value name. The value names may be common among different spreadsheet tables, e.g., "Cost" in Figure 1a. We use a key-frequency list of $< key, count >$ to represent the value name distribution from an existing spreadsheet corpus [18], where key is a value name, and $count$ shows the occurrence count of value name key. We use the frequency of the value name as one feature (**F#3**).
- Aggregation caption. If the text in the header contains tokens like "total", "sum" and "max", this feature (**F#4**) is set to 1, e.g., header F2 in Figure 1a.
- Aggregation formula. For a header h in a top header region, if some formula cells in the same column as h reference a range of cells, this feature (**F#5**) is set to 1, e.g., F2 in Figure 1a.

Style features. The styles of a header can provide valuable indicators for the identification process. We define 3 style features for each header in a header pair.

- Merged cell. If a header h is a merged cell, we use the numbers of rows (**F#6**) and columns (**F#7**) occupied by the merged cell as two features. If h is not a merged cell, these two features are set to 1.
- Indents. If a header's text contains indents, this feature (**F#8**) is set to the number of indents, otherwise, 0.

Spatial features. These features include the locations of a header and its neighbors. We define 12 spatial features for each header in a header pair.

- Row ratio & column ratio. The row index $index_r$ of a header h in its table t (**F#9**) and the normalized row index $index_r$ / $t.totalRow$ (**F#10**) are considered as two features. Similarly, the column index $index_c$ of h (**F#11**) and the normalized column index $index_c / t.totalColumn$ (**F#12**) are considered as another two features.
- Header number. For top headers, the numbers of headers in different rows can be useful to infer header hierarchy. For example, row 1 in Figure 1a has less headers than row 2, and this indicates that some headers in row 1 may be parents of some headers in row 2. Let $headers_h$ be the number of headers in the same row as header h in a top header region, and loc_h be the index location where h is located in all headers in the same row as header h. We consider $headers_h$ (**F#13**) and $loc_h / headers_h$ (**F#14**) as two features.
We further compare the numbers of headers in row $h.row$ and its neighboring rows. Let $prevHeaders$ and $nextHeaders$ be the number of $h.row$'s previous row (i.e., $h.row - 1$) and next row (i.e., $h.row + 1$), respectively. We consider two features, which are $(prevHeaders - headers_h)/headers_h$ (**F#15**) and $(nextHeaders - headers_h)/headers_h$ (**F#16**). If h is located in the first row or the last row in the header region, the corresponding feature is set to 0.
- Upper left header. For a header h, we assume its upper left header as h_{ul}. For example, in Figure 1a, header D3's upper left nearest header is header C2. If h_{ul} exists and h_{ul} is a merged cell, we use the number of rows (**F#17**) and the number of columns (**F#18**) occupied by h_{ul} as two features. If h_{ul} is not a merged cell, these two features are set to 1. We use h_{ul}'s row index (**F#19**) and column index (**F#20**) as another two features. These features can reflect the possible parent-child relation. If h_{ul} does not exist, these four features are set to -1.

3.2.2 Inter-Header Features. For a header pair $< h1, h2 >$, we extract in total 8 spatial features, which can reflect the relation between $h1$ and $h2$.

- Index difference. For $h1$ and $h2$, we consider the differences between their row indexes (row_d) (**F#21**) and column indexes (col_d) (**F#22**) as two features. Further, we consider the smaller difference of row_d and col_d as a feature (**F#23**).
- Relative location. For $h1$ and $h2$, if $h1$ is located in the left and top of $h2$, this feature is set to 1 (**F#24**).
- Header number difference. Similar to feature F#21 and F#22, the number difference in row $h1.row$ and $h2.row$ can help to infer header hierarchy between $h1$ and $h2$. Let $headers_{h1}$ be the number of headers in row $h1.row$ and $headers_{h2}$ be the number of headers in row $h2.row$. We consider $headers_{h1} - headers_{h2}$ as a feature (**F#25**).

- Thick border. If there is a thick border between $h1$ and $h2$ in the header pair, this feature is set to 1 (**F#26**).
- Blank row & column. If there exists a blank column (i.e., all cells in the column are empty) between $h1$ and $h2$ in the top header region, this feature is set to 1 (**F#27**).
- Aggregation relation. For $< h1, h2 >$ in a top header region, if some formula cells in the same column as $h1$ reference a range of cells, we assume that the type of header $h1$ is an aggregation. If the referenced cells contain the cells in the same column as $h2$, this feature is set to 1, otherwise, 0 (**F#28**). For example, in Figure 1a, the type of header F2 is an aggregation. The referenced cells in F4 contain the cells in column E. So, header E2 is aggregated by header F2.

3.3 Training and Prediction

Training. For each labeled spreadsheet table, we extract all its header pairs (Section 3.1). For each header pair $< h1, h2 >$, Tasi extracts 46 features in Section 3.2, and predicts its category $< h1.type$, $h2.type$, $relation(h1, h2) >$. In the training step, we aim to correctly predict most header pairs' categories. We use a multi-classifier θ_{model}, e.g., Random Forest [12], Logistic Regression [35] and Decision Tree [42], to train a multi-classifier model.

Prediction. Given a header pair $< h1, h2 >$, Tasi extracts 46 features from its contents, styles and spatial locations, and uses the trained multi-classifier to predict the probability of each category Note that, we only use 32 valid categories, and ignore other invalid categories.

3.4 Generating Semantic Table Structure

For each header pair, we can learn the probability of each category. If we simply use the category with the highest probability for each header pair, we may obtain inconsistent semantic table structure. For example, two header pairs $< h1, h2 >$ and $< h2, h3 >$ may have different header types for header $h2$. Therefore, we need to generate a consistent semantic table structure which has the highest potential to be the correct one.

Algorithm 1 presents how to generate the consistent semantic table structure. Basically, we iteratively add a header pair hp into the current structure ($curStructure$). For each header pair hp, we evaluate all its possible predict results (Line 18–24). We choose its predicted result, which has the highest probability and is compatible with $curStructure$ (Line 19–20), and add it into the current structure ($curStructure$ in Line 21). To speed up this process, if the current structure has bad performance, we prune a non-optimal solution (Line 15–16). For all consistent semantic table structures, we keep the best one (Line 10–13).

The predication result of a header pair $< h1, h2 >$ is compatible with the current solution $curStructure$ if it satisfies the following two conditions. (1) $h1$ and $h2$ have the same header types as ones in $curStructure$. (2) The relation of $< h1, h2 >$ does not conflict with other relations in $curStructure$. For example, if $h1$ is a parent of $h2$ and $h2$ is a parent of $h1$, then these two relations conflict.

Algorithm 1: Generating consistent table structure.

```
   Input: $headerPairs$ (Header pair list)
   Output: $structure$
1  $curStructure \leftarrow Stack()$;
2  $bestStructure \leftarrow NULL$;
3  $bestEval \leftarrow MAX\_VALUE$;
4  getStructure(1);
5  return $bestStructure$;
6
7  Function getStructure($i$) do
8      $eval \leftarrow eval(curStructure)$;
9      if $i > headerPairs.length$ then
10         if $eval < bestEval$ then
11             $bestStructure \leftarrow curStructure$;
12             $bestEval \leftarrow eval$;
13         return;
14     else
15         if $eval > bestEval$ then
16             return;
17     $hp = headerPairs.get(i)$;
18     for $j \leftarrow 1; j \leq \theta_{maxResult}; j++$ do
19         $predictResult \leftarrow hp.getBestPredictResult(j)$;
20         if $compatible(curStructure, predictResult)$ then
21             $curStructure.push(predictResult)$;
22             getStructure($i + 1$);
23             $curStructure.pop()$;
24     end
25 end
```

A semantic table structure is better if it can satisfy the following conditions. (1) The possibility of each header pair's prediction result is as high as possible. (2) There are as many relations among headers as possible. Let N be the number of headers in a header region, HP_r be all header pairs with certain relations in the current solution, HP_n be all header pairs with no relation in the current solution, $hp.pr.prob$ denotes the probability of a header pair hp's prediction result ($hp.pr$). We use the following function to evaluate a semantic table structure.

$$eval = \sum_{hp \in HP_r} -log(hp.pr.prob) + \theta_w * \frac{(HP_n.size)^2}{N * (N-1)/2}$$

If there are N headers in a left or top header region, we can obtain $N * (N-1)/2$ header pairs. The evaluation function $eval$ is divided into two parts. In the first part, we intend to maximize the probabilities of header pairs with certain relations. Note that, the value range for $hp.pr.prob$ is $[0, 1]$. Thus, the greater the prediction result of a header pair is, the smaller the evaluation is. In the second part, we intend to reduce the number of header pairs with no relation, since no relation can only provide limited information for semantic table structure. Thus, we punish the pairs with no relation. We use θ_w to balance these two parts.

Note that, in Algorithm 1, we only select the top $\theta_{maxResult}$ prediction results for each header pair (Line 18), instead of all 32 valid prediction results. The reasons are as follows. (1) It is potential that all header pairs can be predicted correctly with higher predict probabilities for each header pair. (2) It is time-consuming to search all possible solutions in Algorithm 1, and return a good solution in a limited time (1s in our experiment). (3) If $\theta_{maxResult}$ is too large, it increases the possibility of selecting a prediction result with smaller prediction probability, which can increase the probability of generating a wrong semantic table structure.

3.5 Parameters

Our algorithm takes three parameters as input. (1) **Multi-classifier model (θ_{model}):** θ_{model} represents the classifier adopted in Section 3.3. We adopt Random Forest [12], Logistic Regression [35] and Decision Tree [42] in scikit-learn [6] for θ_{model} in our experiment. (2) **Selected max predication results ($\theta_{maxResult}$):** In Algorithm 1, we select the top $\theta_{maxResult}$ prediction results for each header pair. (3) **The weight in evaluation function (θ_w):** We use θ_w to balance the two parts in the evaluation function.

In the experiment in Section 5.1.2, we evaluate the combinations of these parameters using 5-fold cross validation on the training set, and obtain the best performance. The experimental result shows that Tasi can obtain the best performance when θ_{model} = Random Forest, $\theta_{maxResult} = 3$, and $\theta_w = 0.8$.

4 STRUCTURE-BASED ERROR DETECTION

As discussed in Section 2.2, spreadsheets contain many missing formulas and formula errors. In this section, we discuss two common error patterns, range errors and inconsistent errors among indexes, which are the common cases of missing formulas and formula errors. We propose, TasiError, which can utilize semantic table structures to detect these two error patterns.

Note that, we only explore two error patterns about missing formulas and formula errors to present the usefulness of semantic table structures. Based on semantic table structures, readers can explore more error patterns about missing formulas and formula errors. For example, in Figure 1a, header B6 is an aggregation, which indicates that D6 should have a formula. The index set {A4, A7} indicates that [C4:F6] and [C7:F10] should have similar computations. These can help detecting the formula error in C10.

4.1 Range Errors

Error description. Spreadsheet formulas usually use a range of cells as input. For example, cell F4 in Figure 1a references a cell range [C4:E4]. Since the cells in a cell range are computed together, they should share the similar computational semantics. For example, C4, D4 and E4 represent the costs of Sales department in America in each month. These kinds of cells are usually indexed by an index set, e.g., index set {C2, D2, E2}. We observe two error scenarios in real-world spreadsheets. (1) Some cells in a cell range are indexed by an index set, while other cells in it are not indexed by the same index set. For example, in the cell range [C6:C9] used by C10 in Figure 1a, C7, C8 and C9 are indexed by the index set {B7, B8, B9}, whereas C6 is not indexed by this index set. We infer that the cell range contains more cells than expected, thus introducing a formula error. (2) A cell range only contains partial cells indexed by an index set. For example, in the cell range [C9:D9] used by F9 in Figure 1a, C9 and D9 are indexed by the index set {C2, D2, E2}. However, E9 is also indexed by the index set, but not included. We infer that the cell range misses a cell, thus introducing a formula error.

Error detection. Given a spreadsheet formula, we extract its cell ranges. If a formula uses more than one cell range, e.g., SUM(A1:A10, B1:B10), we check them one by one. We only check cell ranges that contain one row of cells (i.e., row-based cell range), e.g., [C4:E4] in F4, or one column of cells (i.e., column-based cell range), e.g., [C6:C9] in C10. The only difference between row-based cell ranges and column-based cell ranges is their directions. Thus, for clarity, we only use row-based cell ranges to explain our detection algorithm.

We use index sets identified by Tasi to detect range errors. Given a row-based cell range, we first identify the index sets in the top header region that indexes cells in the cell range. For example, for the cell range [C9:D9] in F9, we can find that C9 and D9 are indexed by index set {C2, D2, E2} in the top header region. Then, we extract all cells in row 9 (in which the cell range [C9:D9] locates) indexed by this index set, and obtain a cell set, i.e., {C9, D9, E9}. If there is a difference between this cell set and the cell range, we detect a formula error. For this example, cell range [C9:D9] does not contain E9, thus introducing a formula error.

4.2 Inconsistent Errors among Indexes

Error description. The cells indexed by an index set usually share the similar computational semantics. For example, in Figure 1a, header C2, D2 and E2 form an index set, which indexes the costs in each month. For each row (in row 4−10), the cells indexed by the index set {C2, D2, E2} share the similar computational semantics. For example, in row 6, C6, D6 and E6 are used to compute the total costs of Sales department for each month. If their computational semantics are different, some cells may contain errors. For example, D6 and E6 do not have a formula like C6, thus introducing two missing formula errors. Similarly, F9 introduces a formula error since it does not have a formula like F7 and F8, which are indexed by the index set {B7, B8, B9}.

Error detection. Since the detection algorithms on row-based index sets (e.g., {C2, D2, E2}) and column-based index sets (e.g., {B7, B8, B9}) are similar, we take row-based index sets as an example. Note that, our algorithm can only handle row (column)-based index sets in which each index can only index one column (row) of cells, e.g., row-based index set {C2, D2, E2} and column-based index set {B7, B8, B9}. We cannot handle other types of index sets, e.g., {A4, A7}.

Given a row-based index set *is*, we put cells indexed by *is* in each row of the table into a cell group. Take row-based index set {C2, D2, E2} as an example. We can obtain a cell group {C4, D4, E4} in row 4, and a cell group {C6, D6, E6} in row 6, etc. The cells in each cell group should share the similar computational semantics.

Note that, these cell groups are similar to cell arrays in AmCheck [20] & CACheck [22], and cell clusters in CUSTODES [16]. Different from them, we use index sets to obtain these cell groups. Furthermore, Koci et al. [34] classify cells into five cell types, i.e., header, data, attribute, metadata and derived, but cannot infer header types and header relations. Chen et al. [15] can identify parent-child relation, but cannot infer header types. Therefore, these approaches cannot correctly infer index sets.

We adapt the basic idea in CACheck and CUSTODES to detect inconsistent errors in cell groups. We describe it as follows.

(1) If there are not any formula cells in a cell group, e.g., {C4, D4, E4}, we ignore the cell group, since we cannot infer its computational semantics.

Table 1: The statistics of spreadsheets in the ground truth.

Item	Total	Train	Test
Spreadsheet	**3,110**	2,488	622
Rows on average	**75**	72	85
Columns on average	**19**	19	19
Cells on average	**649**	639	691
Formulas on average	**137**	138	133
Table	**3,290**	2,651	639
Left header region	**2,648**	2,130	518
Top header region	**3,204**	2,579	625
Left header pair	**1,188,364**	929,617	258,747
Top header pair	**235,185**	190,859	44,326

Table 2: The statistics of table structures in the ground truth.

	Item	Total	Train	Test	Cor.	Acc.
Type	T1	**11,570**	9,300	2,270	1,546	68.1%
	T2	**73,815**	58,994	14,821	12,715	85.8%
	T3	**333**	275	58	15	25.9%
	T4	**770**	597	173	53	30.6%
	T5	**3,195**	2,486	709	226	31.9%
Relation	R1	**12,460**	9,547	2,913	1,119	38.4%
	R2	**0**	0	0	0	-
	R3	**1,011,892**	803,040	208,852	174,633	83.6%
	R4	**3,784**	3,134	650	293	45.1%
	R5	**61**	56	5	0	0.0%
	R6	**1,243**	968	275	17	6.2%
	R7	**6,558**	5,274	1,284	627	48.8%
	R8	**387,551**	298,457	89,094	51,300	57.6%

(2) As errors normally occur in minority, they can be detected as outliers in a cell group. We first count the number of cells that use the same formula pattern in the R1C1 format. Then, we choose the formula pattern that can cover most cells as the formula pattern of the cell group (fp). If we identify more than one formula pattern, we select the first one as fp. For example, in the cell group {C6, D6, E6}, its formula pattern is SUM(R[-2]C:R[-1]C). In the cell group {F7, F8, F9}, its formula pattern SUM(RC[-3]:RC[-1]).

(3) If a formula in a cell group is different from fp, it contains a formula error. For example, in the cell group {F7, F8, F9}, F9's formula is different from the formula pattern SUM(RC[-3]:RC[-1]), and thus contains a formula error.

(4) For a data cell in the cell group, we apply the formula pattern fp on the cell, and then check whether the value can be computed by the formula pattern fp. If yes, we consider the data cell contains a missing formula error. For example, in the cell group {C6, D6, E6}, D6 and E6 can be computed by the formula pattern SUM(R[-2]C:R[-1]C). Thus, D6 and E6 contain two missing formulas.

5 EVALUATION

Our evaluation studies the following three research questions.

RQ1: *How effective is Tasi in identifying semantic table structures in spreadsheets?*

RQ2: *How effective is TasiError in detecting spreadsheet errors?*

RQ3: *How is TasiError compared with existing techniques, e.g., CUSTODES [16], CACheck [19], and ExceLint [11]?*

To answer RQ1, we evaluate Tasi on our built ground truth and analyze its performance (Section 5.1). To answer RQ2 and RQ3, we evaluate TasiError on the CUSTODES dataset [16], and further compare TasiError with existing approaches (Section 5.2). We have made our datasets and experimental results available online at https://github.com/tcse-iscas/Tasi.

5.1 Evaluation on Tasi

5.1.1 Dataset Construction. To the best of our knowledge, no spreadsheet corpora have documented their semantic table structures. The commonly-used corpora, EUSES [25], Enron [27] and FUSE [10], do not provide their semantic table structures. During collaborating with Microsoft, we have the opportunity to collect a larger and newer spreadsheet dataset compared with existing ones, e.g., EUSES (created before 2005, 4,498 spreadsheets), Enron (created before 2001, 15,770 spreadsheets) and FUSE (created before 2015, 249,376 spreadsheets). To obtain recently-used real-world spreadsheets, we crawl spreadsheet files from the Internet through a web crawler developed by the Bing search engine team, and obtain all spreadsheet files according to file suffixes, i.e., .xls and .xlsx. Finally, we obtain 4,290,022 spreadsheets, and use them as our experimental subject.

It is time- and cost- consuming to reconstruct semantic table structures for all spreadsheets. So, we randomly sample 3,500 spreadsheets from these spreadsheets, and reconstruct their semantic table structures. In the sampling process, we do not bias the spreadsheets in any special domains. We also exclude the spreadsheets that are not written in English, which Tasi cannot support.

We employ nine experienced data labelers in Speechocean (a third-party artificial intelligence data provider) [7] to label the semantic table structures of the sampled spreadsheets. The entire dataset building process includes two steps: labeling and confirmation. In the labeling step, each sampled spreadsheet is labeled by one labeler. The labeler first identifies all tables in it. If a table contains only one row / column, we assume that it has limited information, e.g., no headers or data, and exclude it from consideration. If the labeler cannot fully understand a table's semantics, we exclude it from consideration. Thus, we ensure the accuracy of our dataset. If the labeler confirms that a table has similar structures with an already-labeled table, we also exclude it from consideration. Thus, we ensure that our dataset has diverse table structures. For the remaining tables, the labeler identifies its top header region and left header region, and then all header types and header relations. In the confirmation step, to avoid errors introduced by individual labelers, the labeling results are verified and revised by another experienced labeler until two labelers reach an agreement. The above labeling process takes about 148 man-days.

Through the above sampling and labeling process, we obtain 3,290 tables from 3,110 spreadsheets[5]. The basic statistics of these spreadsheets are shown in column Total in Table 1, and their header types and relations are shown in column Total in Table 2. Note

[5] Since we have to go through a compliance and privacy certification before publishing the whole dataset, we provide 50 spreadsheets and their related annotations for now.

that, different header types and relations are imbalanced in Table 2, because header types and relations should be imbalanced in practice. For example, index types (T_2) are usually more common than other types, and we do not obtain any header pair with the relation of "$h2$ is $h1$'s parent" (R_2).

5.1.2 Training. We randomly split 3,110 spreadsheets into two parts, 80% as training dataset and 20% as testing dataset. The statistics about two datasets are shown in column Train and Test in Table 1 and Table 2, respectively. We use the training dataset (column Train in Table 2) to obtain the multi-classifier model and parameter setting in Tasi.

Parameter setting. As discussed in Section 3.5, Tasi takes three parameters as input. For θ_{model}, there are the three candidates, i.e., Random Forest [12], Logistic Regression [35] and Decision Tree [42]. For $\theta_{maxResult}$, we use four candidates, i.e., 1, 3, 5, 10. For θ_w, we use six candidates, i.e., 0.0, 0.2, 0.4, 0.6, 0.8 and 1.0.

For these three parameters, we can obtain 72 (3*4*6) candidate combinations. We use 5-fold cross-validation on the training dataset (column Train in Table 2) for each combination and calculate the accuracy indicator by the generated semantic table structures, according to Section 3.4. Finally, we obtain the following best candidates: θ_{model} = Random Forest, $\theta_{maxResult} = 3$, $\theta_w = 0.8$. For this candidate, the accuracy of header pairs on the training dataset (column Train in Table 2) is 77.6%.

5.1.3 Experimental Result. We use the trained Tasi to validate Tasi's performance on the testing dataset (column Test in Table 2).

For the 303,073 header pairs in the testing dataset, Tasi can correctly identify the structures of 227,989 header pairs. That said, the accuracy of the structure identification of header pairs is 75.2%. Specifically, Tasi can identify the structures of 201,797 (78.0%) header pairs in the left header regions, and 26,192 (59.1%) header pairs in the top header regions. Note that, for a header pair $< h1, h2 >$, if $h1$ and $h2$'s types and relations are all correctly identified, we consider the structure of this header pair is correctly identified. We further show the detailed results for header types and relations in column Cor. (i.e., correct) and Acc. (i.e., accuracy) in Table 2.

For all 639 tables in the testing dataset, Tasi can identify their structures without any errors in 282 tables. That said, the accuracy of semantic table structure identification is 44.1%. Specially, for 518 left header regions, Tasi can identify their structures without any errors in 363 (70.1%) header regions; for 625 top header regions, Tasi can identify their structures in 339 (54.2%) header regions. Due to the complexity of table structures, it is challenging to fully identify semantic table structures without any errors. The accuracy needs to be further improved in the future.

For all 639 tables, we further compute the accuracy of header pairs in each table. Figure 4 shows the distribution of accuracy of header pairs for all tables, top header regions and left header regions. We rank these tables according to their accuracies of header pairs from low to high. We can see that, Tasi performs better on the left header regions than the top header regions. We infer that the left header regions are usually more well-structured than the top header regions.

We further investigate whether our semantic table structure evaluation function in Section 3.4 can introduce inaccuracy. That said, the incorrect semantic table structure can obtain a smaller evaluation score than the correct one. We find that 17 left header regions and 38 top header regions cannot be correctly generated due to this reason. A better evaluation function could improve Tasi's accuracy greatly.

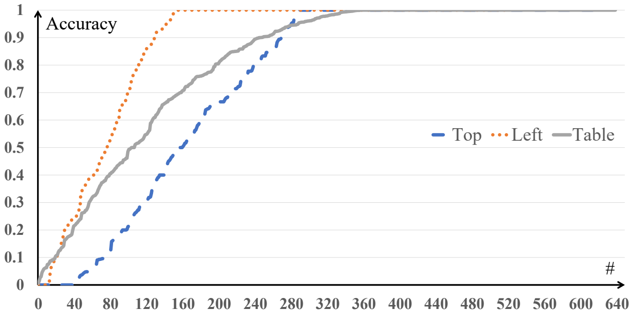


Figure 4: The distribution of accuracy of header pairs for all tables, top header regions and left header regions.

Based on the above analyses, we can draw the following conclusion to RQ1: *Tasi is effective in identifying semantic table structures in real-world spreadsheets.*

To the best of our knowledge, Tasi is the first approach to identify the header types and relations (Section 2.1). Koci et al. [34] can classify spreadsheet into 5 coarse-grained types, i.e., header, data, metadata, derived and attribute. It cannot identify the different types of headers. Chen et al. [15] can only identify the parent-child relation among headers, and cannot identify header types and other relations. Therefore, we do not compare with these approaches.

5.2 Evaluation on TasiError

5.2.1 Experimental Setting. To evaluate TasiError, we adopt the CUSTODES dataset [16] as our experimental subject due to the following reasons. First, the spreadsheets in Section 5.1 do not have annotations for spreadsheet errors. Second, CUSTODES dataset has been used by many error detection approaches [11, 16, 44], and should represent real-world spreadsheet errors. Third, TasiError and CUSTODES detect the same types of spreadsheet errors, i.e., missing formulas and formula errors. Thus, We can have a fair comparison with existing approaches by using the same dataset.

The CUSTODES dataset contains 70 spreadsheets from eight categories (Category in Table 3), 291 worksheets and 1,974 missing formulas and formula errors. Note that, this dataset does not overlap with the dataset in RQ1. During our study, we find that some missing formulas and formula errors in these spreadsheets are not marked out in the original CUSTODES dataset. Note that, similar omission has also been reported by ExceLint [11]. Thus, we re-label the missing formulas and formula errors in these 70 spreadsheets. We invite five master students, who are not the authors of this paper, to help labeling errors. To make re-labeling errors accurate, five master students carefully inspect these spreadsheets, and understand their structures and semantics, and further identify all missing formulas and formula errors. Note that, we only explain the concepts of missing formulas and formula errors to labelers, and do not bias to any error patterns. For each spreadsheet, all participants carefully

Table 3: The comparison of spreadsheet error detection.

Category	**Ground truth**		**TasiError**						**CUSTODES**		**CACheck**		**ExceLint**		**Excel**	
			Range Error		**Index Error**		**Total**									
	SS	**Error**	**DT**	**TP**	**DT**	**TP**	**DT**	**TP**	**DT**	**TP**	**DT**	**TP**	**DT**	**TP**	**DT**	**TP**
cs101	1	6	0	0	1	1	1	1	3	3	6	6	4	4	1	0
database	14	1,998	91	24	1,888	1,869	1,979	1,893	1,140	1,066	882	842	62	14	563	40
financial	23	993	502	315	371	329	873	644	651	324	515	336	94	28	1,204	327
forms3	2	10	37	2	2	1	39	3	29	7	7	1	20	1	464	2
grades	7	210	41	41	81	79	122	120	316	95	176	99	11	4	322	43
homework	7	58	249	7	11	11	260	18	71	46	77	34	23	11	1,238	8
inventory	9	86	200	50	60	12	260	62	144	23	50	40	15	11	391	37
modeling	7	341	241	26	306	301	547	327	89	18	101	30	20	8	797	24
Total	**70**	**3,702**	**1,361**	**465**	**2,720**	**2,603**	**4,081**	**3,068**	**2,443**	**1,582**	**1,814**	**1,388**	**249**	**81**	**4,980**	**481**

cross-validate their labeling results. If any inconsistency occurs, they discuss it and finally reach a consensus. Thus, we can avoid potential errors in the labeling process. This re-labeling and cross-checking process takes about 10 days. Note that, all newly annotated errors belong to the two types of errors, i.e., missing formulas and formula errors. We do not introduce new types of errors into the CUSTODES dataset, e.g., format errors.

To use Tasi on these spreadsheets, we further label the tables, left header regions and top header regions in them. In total, we find 506 tables in these 70 spreadsheets, 486 left header regions, and 458 top header regions. Note that, it is not difficult to mark out tables and header regions for human users, since human users can easily understand the functions of tables, and divide the spreadsheets into tables according to different functions, and further divide the tables into header regions and data regions.

The first three columns in Table 3 show the statistics of errors in the 70 spreadsheets. Our manual inspection finds 3,702 errors (Error), in which, 1,744 errors are omitted in the original CUSTODES dataset. Note that, 16 cells in the original CUSTODES dataset are wrongly labeled as faulty, thus we exclude them in our dataset. For all 3,702 errors, 1,308 errors have lead to wrong data.

5.2.2 *Detection Result.* We first use Tasi to extract the semantic table structures for all tables in these 70 spreadsheets. Then, we run TasiError to detect spreadsheet errors based on the semantic table structures identified by Tasi.

Table 3 shows the statistics of errors detected by TasiError (TasiError). In total, TasiError detects 4,081 errors (TasiError/DT), of which, 3,068 (75.2%) are true (TasiError/TP). TasiError misses 634 errors, i.e., the recall of TasiError is 82.9%. Therefore, the F1-measure of TasiError is 78.8%.

Table 3 also shows the statistics of detected range errors and inconsistent errors among indexes. We can see that TasiError obtains lower precision (34.2%) in detecting range errors, and higher precision (95.7%) in detecting inconsistent errors among indexes. The range error detection can be further improved in the future.

False positives. TasiError reports 1,013 false positives. We further investigate the reasons of these false positives. (1) TasiError works on the structures identified by Tasi. Wrongly identified semantic table structures can make TasiError fail. For range errors, this causes 474 false positives. For inconsistent errors among indexes, this causes 27 false positives. (2) The rules adopted by TasiError can introduce false positives. For range errors, TasiError introduces 422 false positives. For example, a formula needs to calculate multiple index sets according to the semantics. For inconsistent errors among indexes, TasiError introduces 90 false positives. For example, we may wrongly choose the formula pattern in a cell group, when the correct formula does not cover most of its cells.

False negatives. TasiError misses 634 errors. There are four reasons for these false negatives. (1) 156 false negatives are caused by wrongly identified semantic table structures by Tasi. (2) Some formulas are not supported by our implementation, e.g., If and Ref. 10 false negatives belong to this case. (3) TasiError cannot obtain the correct formula pattern for a cell group. 343 false negatives belong to this case. (4) TasiError cannot detect formula errors when the formula references cells in different rows (columns) in the column-(row-) based cell groups. 125 false negatives belong to this case.

Based on the above analyses, we can draw the following conclusion to RQ2: *By using semantic table structures identified by Tasi, TasiError can effectively detect errors with a precision of 75.2% and a recall of 82.9%.*

5.2.3 *Comparison with Existing Approaches.* To evaluate the effectiveness of TasiError in detecting spreadsheet errors, we compare TasiError with six error detection approaches, i.e., CUSTODES [16], CACheck [22], ExceLint [11], Excel internal error detector [1], AmCheck [20], UCheck [8] & Dimension [14], which are publicly available and represent the state of the art. For fair comparison, we slightly revise these approaches and make them use our annotated tables and header regions when necessary. The detection results of these tools are shown in Table 3. Specially, for the 3,068 errors detected by TasiError, 1,172 errors cannot be detected by CUSTODES, CACheck, ExcelLint, Excel internal error detector, AmCheck, and UCheck & Dimension. Note that, 463 (12.5%) errors can be detected by existing approaches, but cannot be detected by TasiError. This is caused by two reasons. First, Tasi wrongly identifies table structures, causing 107 errors missed. Second, TasiError's detection patterns cannot detect 356 errors.

Table 3 shows their detailed comparison results. We do not show the detailed results for AmCheck and UCheck & Dimension in

Table 3 due to space limitation. For CUSTODES, its precision, recall, and F1-measure are 64.8%, 42.7%, and 51.5%, respectively. For CACheck, its precision, recall, and F1-measure are 76.5%, 37.5%, and 50.3%, respectively. For ExceLint, its precision, recall, and F1-measure are 32.5%, 2.2%, and 4.1%, respectively. For Excel internal error detector, its precision, recall, and F1-measure 9.7%, 13.0%, and 11.1%. For AmCheck, its precision, recall and F1-measure are 58.2%, 34.0%, and 42.9%, respectively. For UCheck & Dimension, its precision, recall, and F1-measure are 2.1%, 1.1%, and 1.4%, respectively.

We further compare TasiError with existing approaches in the original CUSTODES dataset. From the previous experiment, we can see that CUSTODES and CACheck have the best performance among existing approaches. Thus we only compare TasiError with CUSTODES and CACheck here. For the 1,974 errors in the original CUSTODES dataset, TasiError, CUSTODES, and CACheck achieve precisions of 38.6%, 64.8%, and 74.5%, respectively. Note that, TasiError has the lowest precision here because 1,493 real errors detected by TasiError are omitted in the original CUSTODES dataset, thus wrongly considering as false positives. TasiError, CUSTODES, and CACheck achieve recall of 79.8%, 80.2%, and 68.5%. We can see that TasiError achieves a comparable performance with CUSTODES and CACheck. For the 1,744 additional errors, CUSTODES and CACheck can only detect 49 errors, whereas TasiError can detect 1,493 errors.

We further investigate why TasiError performs better than existing approaches, and summarize them as follows. First, the structures related to some errors (e.g., range errors) are not identified by existing approaches, thus they miss these errors. Whereas, Tasi and TasiError can detect and utilize these structures. Second, the detection rules in existing approaches can omit some errors. For example, CUSTODES, CACheck, and AmCheck wrongly consider C10, D10 and E10 in Figure 1a as correct. However, TasiError can utilize index sets to detect these errors.

Based on the above analyses, we can draw the following conclusion to RQ3: *TasiError significantly outperforms CUSTODES, CACheck, ExceLint, AmCheck, UCheck & Dimension and Excel internal detector. This demonstrates that semantic table structures can greatly help error detection in spreadsheets.*

6 DISCUSSION

In this section, we discuss the limitations and threats to validation.

6.1 Limitations

User-specified tables and header regions. In this work, we mainly focus on semantic table structure identification, and assume that users can provide tables and header regions in spreadsheets. Note that, there have already been some works that can automatically identify tables and header regions in a spreadsheet, e.g., TableSense [18], RAC [33], Dong et al. [17], Koci et al. [34] and Gol et al. [26]. These approaches can be integrated into Tasi, achieving an end-to-end solution for semantic table structure identification in spreadsheets.

Limited language support. For now, Tasi only supports spreadsheets written in English. We can find there are many domain-specific vocabularies, irregular abbreviations and words in spreadsheets, which we cannot handle effectively. In the future, supporting different languages will be greatly helpful.

6.2 Threats to Validity

Representativeness of experimental subjects. One threat to the external validity is the representativeness of our experimental subjects used in the evaluation. To evaluate Tasi, we crawl lots of real-world spreadsheets from the Internet. To evaluate TasiError, we select the CUSTODES dataset, which has been widely used in spreadsheet researches [11, 16, 44]. Thus, we believe our experimental subjects can represent the real-world spreadsheets in practice.

Dataset construction. Semantic table structures and errors in our studied spreadsheets are not well documented. It is also impossible to inspect the semantic table structures and errors with the help of their original authors. Thus, we invite expert labelers and master students to inspect table structures and errors in our experimental subjects. To alleviate potential mistakes, all data are cross-checked by them.

7 RELATED WORK

Spreadsheet structure analysis. Spreadsheets usually have flexible structures. Various approaches are proposed to identify different kinds of spreadsheet structures. (1) Table identification: TableSense [18] proposes an enhanced convolutional network model to detect table regions. RAC [33] proposes a rule-based approach using graph representation of spreadsheet layout to detect table regions. These works cannot understand semantic table structures. (2) Cell classification: some works categorize spreadsheet cells into header, data, metadata, derived and attribute using different learning-based approaches, e.g., Koci et al. [34], Gol et al. [26] and Sun et al. [41]. Nagy et al. [37] propose a rule-based method to divide the tables into headers and data in web tables. Although these works can identify headers, but they cannot distinguish different header types. (3) Special structure identification: ExpCheck [21] inspects cell formats and semantic information to identify similar cell groups. Chen et al. [15] propose a two-phase semi-automatic system to extract parent-child relation among headers. TableCheck [19] and LTC [48] propose to identify table clones. Gyro [28] extracts class diagrams based on common usage patterns. VEnron [23] analyzes structure changes among spreadsheet evolution. These works cannot identify fine-grained header types and their relations, whereas Tasi can.

Spreadsheet error detection. Spreadsheets contain various errors [39, 40]. Many spreadsheet error detection approaches have been proposed to detect errors in spreadsheets. Hermans et al. propose to detect inter-worksheet errors [29] and data clone related inconsistencies [30]. Some approaches detect errors based on certain structures in spreadsheets, e.g., AmCheck [20], CACheck [22], CUSTODES [16], Melford [44], ExcelLint [11], TableCheck [19], EmptyCheck [46], WARDER [31], UCheck [8] and Dimension [14]. However, these approaches mostly focus on partial structures in spreadsheets. As we discussed in Section 5.2, by using semantic table structures, TasiError can greatly improve the capability of detecting spreadsheet errors.

8 CONCLUSION

Understanding semantic table structures is the fundamental step to perform various spreadsheet analysis tasks, e.g., data analysis and error detection. In this paper, we propose Tasi, to detect semantic table structures in spreadsheets, including header types and relations among headers. We further propose a structure-based error detection approach, TasiError, to detect spreadsheet errors based on semantic table structures identified by Tasi. Our experiments on real-world spreadsheets show that Tasi can identify semantic table structures effectively. By utilizing semantic table structures identified by Tasi, TasiError can detect much more errors than existing approaches precisely.

In the future, we plan to pursue the following research directions. First, we can explore how to incorporate users' efforts to fix structure identification errors, achieving a more effective solution. Second, we can utilize Tasi to transform spreadsheets into relational data for easy data analyses. Third, TasiError can be improved by developing more structure-based error detection patterns.

ACKNOWLEDGEMENTS

We thank Huiyu Bao, Fanzhang Peng, Jiayi Zheng, Qianwang Dai, Tao Wang and Jiansen Song for their contributions in labeling spreadsheet errors and validating experimental results. This work was partially supported by National Natural Science Foundation of China (61702490, 62072444), Microsoft Research Asia Collaborative Research Program, Frontier Science Project of Chinese Academy of Sciences (QYZDJ-SSW-JSC036) and Youth Innovation Promotion Association at Chinese Academy of Sciences.

REFERENCES

[1] 2021. *Error detector in Excel 2019*. Retrieved May 8, 2021 from https://www.dummies.com/software/microsoft-office/detecting-and-correcting-errors-in-excel-2019-formulas/

[2] 2021. *European Spreadsheet Risks Interest Group*. Retrieved May 8, 2021 from http://www.eusprig.org/horror-stories.htm

[3] 2021. *Ideas in Excel*. Retrieved May 8,2021 from https://support.office.com/en-ie/article/ideas-in-excel-3223aab8-f543-4fda-85ed-76bb0295ffc4

[4] 2021. *Power BI | Interactive Data Visualization BI Tools*. Retrieved May 8, 2021 from https://powerbi.microsoft.com

[5] 2021. *Rethinking Spreadsheets and Performance Management*. Retrieved May 8, 2021 from https://www.cutimes.com/2013/07/31/rethinking-spreadsheets-and-performance-management/?slreturn=20200726064739

[6] 2021. *scikit-learn: Machine Learning in Python*. Retrieved May 8, 2021 from https://scikit-learn.org

[7] 2021. *speechocean: A company for AI data provider*. Retrieved May 8, 2021 from http://en.speechocean.com/welcome.html

[8] Robin Abraham and Martin Erwig. 2007. UCheck: A Spreadsheet Type Checker for End Users. *Journal of Visual Languages and Computing* 18, 1 (2007), 71–95.

[9] Marco D Adelfio and Hanan Samet. 2013. Schema Extraction for Tabular Data on the Web. *Proceedings of the VLDB Endowment (VLDB)* 6, 6 (2013), 421–432.

[10] Titus Barik, Kevin Lubick, Justin Smith, John Slankas, and Emerson Murphy-Hill. 2015. FUSE: A Reproducible, Extendable, Internet-Scale Corpus of Spreadsheets. In *Proceedings of IEEE/ACM Working Conference on Mining Software Repositories (MSR)*. 486–489.

[11] Daniel W. Barowy, Emery D. Berger, and Benjamin Zorn. 2018. ExceLint: Automatically Finding Spreadsheet Formula Errors. In *Proceedings of International Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA)*. 148:1–148:26.

[12] Leo Breiman. 2001. Random Forests. *Machine learning* 45, 1 (2001), 5–32.

[13] Jonathan P Caulkins, Erica Layne Morrison, and Timothy Weidemann. 2007. Spreadsheet Errors and Decision Making: Evidence from Field Interviews. *Journal of Organizational and End User Computing* 19, 3 (2007), 1–23.

[14] Chris Chambers and Martin Erwig. 2009. Automatic Detection of Dimension Errors in Spreadsheets. *Journal of Visual Languages & Computing* 20, 4 (2009), 269–283.

[15] Zhe Chen and Michael Cafarella. 2014. Integrating Spreadsheet Data via Accurate and Low-Effort Extraction. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 1126–1135.

[16] Shing-Chi Cheung, Wanjun Chen, Yepang Liu, and Chang Xu. 2016. CUSTODES: Automatic Spreadsheet Cell Clustering and Smell Detection Using Strong and Weak Features. In *Proceedings of International Conference on Software Engineering (ICSE)*. 464–475.

[17] Haoyu Dong, Shijie Liu, Zhouyu Fu, Shi Han, and Dongmei Zhang. 2019. Semantic Structure Extraction for Spreadsheet Tables with a Multi-task Learning Architecture. In *Proceedings of Workshop on Document Intelligence on Neural Information Processing Systems (NeurIPS)*.

[18] Haoyu Dong, Shijie Liu, Shi Han, Zhouyu Fu, and Dongmei Zhang. 2019. Tablesense: Spreadsheet Table Detection with Convolutional Neural Networks. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, Vol. 33. 69–76.

[19] Wensheng Dou, Shing-Chi Cheung, Chushu Gao, Chang Xu, Liang Xu, and Jun Wei. 2016. Detecting Table Clones and Smells in Spreadsheets. In *Proceedings of ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*. 787–798.

[20] Wensheng Dou, Shing-Chi Cheung, and Jun Wei. 2014. Is Spreadsheet Ambiguity Harmful? Detecting and Repairing Spreadsheet Smells due to Ambiguous Computation. In *Proceedings of International Conference on Software Engineering (ICSE)*. 848–858.

[21] Wensheng Dou, Shi Han, Liang Xu, Dongmei Zhang, and Jun Wei. 2018. Expandable Group Identification in Spreadsheets. In *Proceedings of International Conference on Automated Software Engineering (ASE)*. 498–508.

[22] Wensheng Dou, Chang Xu, Shing-Chi Cheung, and Jun Wei. 2017. CACheck: Detecting and Repairing Cell Arrays in Spreadsheets. *IEEE Transactions on software Engineering (TSE)* 43, 3 (2017), 226–251.

[23] Wensheng Dou, Liang Xu, Shing-Chi Cheung, Chushu Gao, Jun Wei, and Tao Huang. 2016. VEnron: A Versioned Spreadsheet Corpus and Related Evolution Analysis. In *Proceedings of International Conference on Software Engineering (ICSE)*. 162–171.

[24] Julian Eberius, Katrin Braunschweig, Markus Hentsch, Maik Thiele, Ahmad Ahmadov, and Wolfgang Lehner. 2015. Building the Dresden Web Table Corpus: A Classification Approach. In *Proceedings of IEEE/ACM International Symposium on Big Data Computing*. 41–50.

[25] Marc Fisher and Gregg Rothermel. 2005. The EUSES Spreadsheet Corpus: A Shared Resource for Supporting Experimentation with Spreadsheet Dependability Mechanisms. In *Proceedings of the Workshop on End-user Software Engineering*. 1–5.

[26] Majid Ghasemi Gol, Jay Pujara, and Pedro Szekely. 2019. Tabular Cell Classification Using Pre-Trained Cell Embeddings. In *Proceedings of IEEE International Conference on Data Mining (ICDM)*. 230–239.

[27] Felienne Hermans and Emerson Murphy-Hill. 2015. Enron's Spreadsheets and Related Emails: A Dataset and Analysis. In *Proceedings of International Conference on Software Engineering (ICSE)*, Vol. 2. 7–16.

[28] Felienne Hermans, Martin Pinzger, and Arie Van Deursen. 2010. Automatically Extracting Class Diagrams from Spreadsheets. In *Proceedings of European Conference on Object-Oriented Programming (ECOOP)*. 52–75.

[29] Felienne Hermans, Martin Pinzger, and Arie van Deursen. 2012. Detecting and Visualizing Inter-Worksheet Smells in Spreadsheets. In *Proceedings of International Conference on Software Engineering (ICSE)*. 441–451.

[30] Felienne Hermans, Ben Sedee, Martin Pinzger, and Arie van Deursen. 2013. Data Clone Detection and Visualization in Spreadsheets. In *Proceedings of International Conference on Software Engineering (ICSE)*. 292–301.

[31] Yicheng Huang, Chang Xu, Yanyan Jiang, Huiyan Wang, and Da Li. 2020. WARDER: Towards Effective Spreadsheet Defect Detection by Validity-Based Cell Cluster Refinements. *Journal of Systems and Software (JSS)* 167 (2020), 1–19.

[32] Andrew J Ko, Robin Abraham, Laura Beckwith, Alan Blackwell, Margaret Burnett, Martin Erwig, Chris Scaffidi, Joseph Lawrance, Henry Lieberman, Brad Myers, et al. 2011. The State of the Art in End-user Software Engineering. *ACM Computing Surveys (CSUR)* 43, 3 (2011), 1–44.

[33] Elvis Koci, Maik Thiele, Wolfgang Lehner, and Oscar Romero. 2018. Table Recognition in Spreadsheets via a Graph Representation. In *Proceedings of IAPR International Workshop on Document Analysis Systems (DAS)*. 139–144.

[34] Elvis Koci, Maik Thiele, Óscar Romero Moral, and Wolfgang Lehner. 2016. A Machine Learning Approach for Layout Inference in Spreadsheets. In *Proceedings of International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*. 77–88.

[35] S Lee. 2005. Application of Logistic Regression Model and its Validation for Landslide Susceptibility Mapping using GIS and Remote Sensing Data. *International Journal of Remote Sensing* 26, 7 (2005), 1477–1491.

[36] Ephraim R McLean, Leon A Kappelman, and John P Thompson. 1993. Converging End-user and Corporate Computing. *Commun. ACM* 36, 12 (1993), 78–90.

[37] George Nagy and Sharad Seth. 2016. Table Headers: An Entrance to the Data Mine. In *Proceedings of International Conference on Pattern Recognition (ICPR)*. 4065–4070.

[38] Ray Panko. 2006. Facing the Problem of Spreadsheet Errors. *Decision Line* 37, 5 (2006), 8–10.

[39] Raymond R Panko. 2008. Spreadsheet Errors: What We Know. What We Think We can Do. *arXiv preprint arXiv:0802.3457* (2008).

[40] Stephen G Powell, Kenneth R Baker, and Barry Lawson. 2008. A Critical Review of the Literature on Spreadsheet Errors. *Decision Support Systems* 46, 1 (2008), 128–138.

[41] Kexuan Sun Harsha Rayudu Jay Pujara. 2021. A Hybrid Probabilistic Approach for Table Understanding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*.

[42] S Rasoul Safavian and David Landgrebe. 1991. A Survey of Decision Tree Classifier Methodology. *IEEE transactions on Systems, Man, and Cybernetics* 21, 3 (1991), 660–674.

[43] Christopher Scaffidi, Mary Shaw, and Brad Myers. 2005. Estimating the Numbers of End Users and End User Programmers. In *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 207–214.

[44] Rishabh Singh, Benjamin Livshits, and Benjamin Zorn. 2017. Melford: Using Neural Networks to Find Spreadsheet Errors. *Tech. Rep.* (2017).

[45] Xinxin Wang and Derick Wood. 1993. Tabular Abstraction for Tabular Editing and Formatting. In *Proceedings of International Conference for Young Computer Scientists*. 17–29.

[46] Liang Xu, Shuo Wang, Wensheng Dou, Bo Yang, Chushu Gao, Jun Wei, and Tao Huang. 2018. Detecting Faulty Empty Cells in Spreadsheets. In *Proceedings of International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 423–433.

[47] Richard Zanibbi, Dorothea Blostein, and James R Cordy. 2004. A Survey of Table Recognition. *Document Analysis and Recognition* 7, 1 (2004), 1–16.

[48] Yakun Zhang, Wensheng Dou, Jiaxin Zhu, Liang Xu, Zhiyong Zhou, Jun Wei, Dan Ye, and Bo Yang. 2020. Learning to Detect Table Clones in Spreadsheets. In *Proceedings of ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. 528–540.