Expandable Group Identification in Spreadsheets

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

Shi Han
Microsoft Research, China
shihan@microsoft.com

Liang Xu
University of Chinese Academy of Sciences
State Key Lab of Computer Science,
Institute of Software, Chinese Academy of Sciences, China
xuliang12@otcaix.iscas.ac.cn

Dongmei Zhang
Microsoft Research, China
dongmeiz@microsoft.com

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

ABSTRACT

Spreadsheets are widely used in various business tasks. Spreadsheet users may put similar data and computations by repeating a block of cells (a *unit*) in their spreadsheets. We name the unit and all its expanding ones as an *expandable group*. All units in an expandable group share the same or similar formats and semantics. As a data storage and management tool, expandable groups represent the fundamental structure in spreadsheets. However, existing spreadsheet systems do not recognize any expandable groups. Therefore, other spreadsheet analysis tools, e.g., data integration and fault detection, cannot utilize this structure of expandable groups to perform precise analysis.

In this paper, we propose *ExpCheck* to automatically extract expandable groups in spreadsheets. We observe that continuous units that share the similar formats and semantics are likely to be an expandable group. Inspired by this, we inspect the format of each cell and its corresponding semantics, and further classify them into expandable groups according to their similarity. We evaluate ExpCheck on 120 spreadsheets randomly sampled from the EUSES and VEnron corpora. The experimental results show that ExpCheck is effective. ExpCheck successfully detect expandable groups with F1-measure of 73.1%, significantly outperforming the state-of-the-art techniques (F1-measure of 13.3%).

CCS CONCEPTS

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

KEYWORDS

Spreadsheet, expandable group

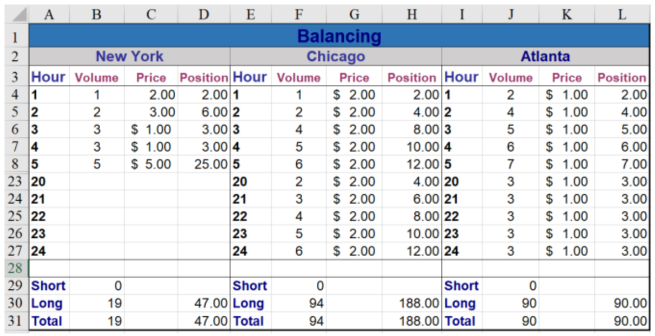
ACM Reference format:
Wensheng Dou, Shi Han, Liang Xu, Dongmei Zhang, and Jun Wei. 2018. Expandable Group Identification in Spreadsheets. In *Proceedings of the 2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE'18), September 3-7, 2018, Montpellier, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3238147.3238222

1 INTRODUCTION

Spreadsheets have been widely used for various business tasks, including data management, decision support, financial reporting, and so on. It was estimated that there were over 55 million users in the United State working with spreadsheets in 2012 [28] and 50-80% of businesses use spreadsheets [30].

In spreadsheet systems, data and computations are organized into a two-dimensional structure. Spreadsheet systems usually provide great flexibility in editing spreadsheets, and various spreadsheet structures may be created by spreadsheet users. Furthermore, many spreadsheets are designed to be interpreted by human, and have flexible structures. Thus, these spreadsheets often cannot be consumed by other spreadsheet analysis tools, e.g., complex data analysis, visualization, and fault detection [27]. For example, Power BI [31] mainly works on data with clearly defined schemas, such as database tables. However, spreadsheet data are often in more flexible forms without a schema specified. Figure 1 shows a typical spreadsheet excerpt extracted from the VEnron corpus [13]. This spreadsheet cannot be simply treated as relational data. Thus, it cannot be directly consumed by relational-data-based analysis tools, unless we identify its structure, such as the title hierarchy (rows 1-3) and repeating structure (e.g., cells [A2:D31] and [E2:H31], cells [A4:L4] and [A5:L5]), and then transform the data into the canonical form of relational data. Therefore, it is important to understand the structure of spreadsheets.

In spreadsheets, users usually put similar data and computations by repeating a block of cells (a *unit*). For example, in Figure 1, each of rows 4-27 represents the data for an hour. So, each of these rows forms a unit, while the units in rows 5-27 repeat the same structure of the unit in row 4. On the other hand, cells [A2:D31] form a unit and represent the data for New York. Similarly, cells [E2:H31] form another unit and represent the data for Chicago.



	A	B	C	D	E	F	G	H	I	J	K	L
1	Balancing											
2	New York				Chicago				Atlanta			
3	Hour	Volume	Price	Position	Hour	Volume	Price	Position	Hour	Volume	Price	Position
4	1	1	2.00	2.00	1	1	$ 2.00	2.00	1	2	$ 1.00	2.00
5	2	2	3.00	6.00	2	2	$ 2.00	4.00	2	4	$ 1.00	4.00
6	3	3	$ 1.00	3.00	3	4	$ 2.00	8.00	3	5	$ 1.00	5.00
7	4	3	$ 1.00	3.00	4	5	$ 2.00	10.00	4	6	$ 1.00	6.00
8	5	5	$ 5.00	25.00	5	6	$ 2.00	12.00	5	7	$ 1.00	7.00
23	20				20	2	$ 2.00	4.00	20	3	$ 1.00	3.00
24	21				21	3	$ 2.00	6.00	21	3	$ 1.00	3.00
25	22				22	4	$ 2.00	8.00	22	3	$ 1.00	3.00
26	23				23	5	$ 2.00	10.00	23	3	$ 1.00	3.00
27	24				24	6	$ 2.00	12.00	24	3	$ 1.00	3.00
28												
29	Short	0			Short	0			Short	0		
30	Long	19		47.00	Long	94		188.00	Long	90		90.00
31	Total	19		47.00	Total	94		188.00	Total	90		90.00

Figure 1: A motivating spreadsheet excerpt extracted from VEnron [13]. In this excerpt, three units [A2:D31], [E2:H31] and [I2:L31] form an expandable group, and 24 units [A4:L4]-[A27:L27] form an expandable group. Note that, rows 9-22 are hidden due to space limitation.

These two units are repeated. We name a unit and all its repeating units as an *expandable group*. Expandable groups are the key and fundamental structure to organize and manage data and computations in spreadsheets. The structure of expandable groups can help transform non-relational data in spreadsheets into relational data, as well as detect spreadsheet faults in expandable groups (more details in Section 2). However, once a spreadsheet is created by users, there is not any clue stored in the spreadsheet to indicate which cells can form an expandable group.

In an expandable group, e.g., [A4:L4]-[A27:L27] in Figure 1, all units, share the same / similar formats and semantics at their corresponding cells. For example, the corresponding cells in units [A4:L4] and [A5:L5] share the similar formats and semantics, although their contents look different. (1) Some corresponding cells (e.g., A4 and A5) are both index numbers; some corresponding cells (e.g., B4 and B5) are both input cells; and some corresponding cells (e.g., L4 and L5) prescribe the same formula pattern (as shown in Figure 2). (2) All corresponding cells have the same or similar headers. For example, cell A4 has the first-level header "Hour" and the second-level header "New York", and cell E4 has the first-level header "Hour" and the second-level header "Chicago". Although the second-level headers of cells A4 and E4 are different, they have the similar semantics, since "New York" and "Chicago" are two cities in US and belong to the same category. (3) All corresponding cells share the similar formats. For example, cells G4 and G5 have the "$" symbol that represents US dollar.

In this paper, we focus on automatically identifying expandable groups in spreadsheets. The key challenge is about how to determine which cells can form expandable groups. Our tool, *ExpCheck*, works based on two observations: (1) The corresponding cells in each unit of an expandable group usually share common features, e.g., cell type, data format, and formula pattern. (2) For the corresponding rows / columns in each unit of an expandable group, their headers share the same or similar semantics. For the first observation, we extract various format features, and compare them. For the second observation, we extract headers of a table, and compare their semantic similarity based on Word2Vec [26].

We implement ExpCheck as a prototype tool and evaluate its performance using the EUSES [16] and VEnron [13] corpora, which are two of the most widely-used corpora for spreadsheet research. The experimental results show that ExpCheck can detect expandable groups effectively, with a precision of 69.3%, recall of 77.3%, and F1-measure of 73.1%. As a comparison, Abraham's spreadsheet template inference approach [2], which also infers expandable groups, can only detect expandable groups with a precision of 25.7%, recall of 8.9%, and F1-measure of 13.3%. This result shows that ExpCheck significantly outperforms existing approaches by 59.8% in F1-measure. Such a big improvement is critical to effectively analyze spreadsheet structure.

ExpCheck substantially differs from expandable group analysis in Abraham's spreadsheet template inference approach [2]. Their work considers two units as expandable only when their corresponding cells share the same formula patterns and types. Thus, it may miss some expandable groups when they do not have formulas or detect incomplete expandable groups when some formulas are missing. For example, their work considers that units [A4:L4]-[A24:L24] form an expandable group due to cells L4:L24 share the same formula pattern in the R1C1 style (shown in Figure 2). Thus, three units [A25:L25]-[A27:L27] are not considered as parts of an expandable group. Instead, ExpCheck detects expandable groups by inspecting the format and semantic similarity among units. As such, ExpCheck can detect the expandable group [A4:L4]-[A27:L27]. ExpCheck also differs from other spreadsheet structure analysis tools [8][10][11][12] in the types of detected structures. Unlike ExpCheck, AmCheck/CACheck [11][12] and CUSTODES [8] aggregate cells into clusters based on their formula similarity in a row or column. For example, AmCheck/CACheck aggregate cells [L4:L27] into a cluster, and CUSTODES can further aggregates cells B31, D31, F31, H31, J31, L31 into a cluster. TableCheck [10] leverages the header information to detect table clones, in which corresponding cells are labelled by the same headers. However, all these approaches cannot present the repeating structure of expandable groups. Thus, they cannot work on expandable group detection.

In summary, this paper makes the following contributions.

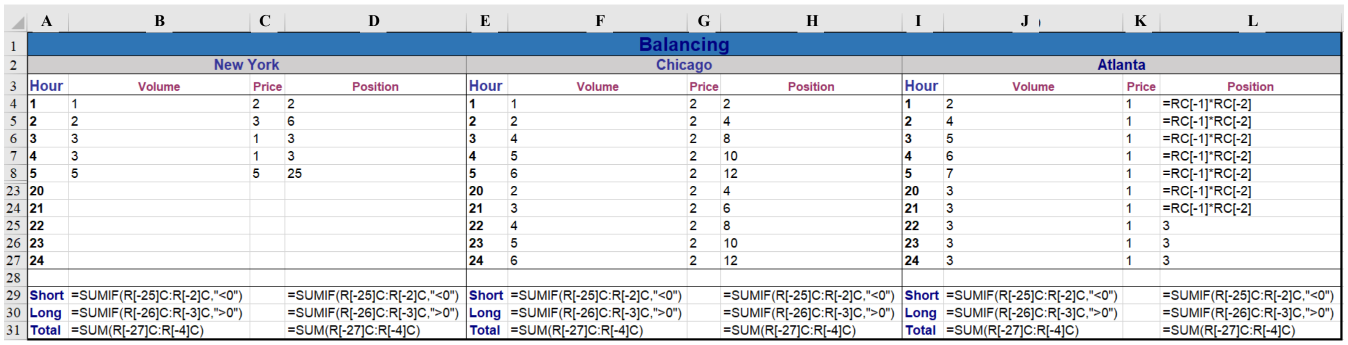
- We propose a novel approach, ExpCheck, to detect expandable groups by inspecting the format and semantic similarity among a block of cells.
- We implement and evaluate ExpCheck on 120 real-life spreadsheets, randomly sampled from the EUSES and VEnron corpora. The experimental results show that ExpCheck can detect expandable groups effectively, and significantly outperforms the state-of-the-art techniques.

2 MOTIVATION AND OVERVIEW

In this section, we explain the key concept of expandable groups, and discuss the importance of expandable group identification. Finally, we briefly introduce how to detect expandable groups.

2.1 Expandable Groups

The concept of expandable groups was first introduced as the core component in ViSTL [15], which is a formal language to model spreadsheet tables. To ease presentation, we use the spreadsheet excerpt in Figure 1 as an illustrative example, which is extracted from the VEnron corpus [13] and performs the balancing analysis



	A	B	C	D	E	F	G	H	I	J	K	L
1	Balancing											
2	New York				Chicago				Atlanta			
3	Hour	Volume	Price	Position	Hour	Volume	Price	Position	Hour	Volume	Price	Position
4	**1**	1	2	2	**1**	1	2	2	**1**	2	1	=RC[-1]*RC[-2]
5	**2**	2	3	6	**2**	2	2	4	**2**	4	1	=RC[-1]*RC[-2]
6	**3**	3	1	3	**3**	4	2	8	**3**	5	1	=RC[-1]*RC[-2]
7	**4**	3	1	3	**4**	5	2	10	**4**	6	1	=RC[-1]*RC[-2]
8	**5**	5	5	25	**5**	6	2	12	**5**	7	1	=RC[-1]*RC[-2]
23	**20**				**20**	2	2	4	**20**	3	1	=RC[-1]*RC[-2]
24	**21**				**21**	3	2	6	**21**	3	1	=RC[-1]*RC[-2]
25	**22**				**22**	4	2	8	**22**	3	1	3
26	**23**				**23**	5	2	10	**23**	3	1	3
27	**24**				**24**	6	2	12	**24**	3	1	3
28												
29	Short	=SUMIF(R[-25]C:R[-2]C,"<0")		=SUMIF(R[-25]C:R[-2]C,"<0")	Short	=SUMIF(R[-25]C:R[-2]C,"<0")		=SUMIF(R[-25]C:R[-2]C,"<0")	Short	=SUMIF(R[-25]C:R[-2]C,"<0")		=SUMIF(R[-25]C:R[-2]C,"<0")
30	Long	=SUMIF(R[-26]C:R[-3]C,">0")		=SUMIF(R[-26]C:R[-3]C,">0")	Long	=SUMIF(R[-26]C:R[-3]C,">0")		=SUMIF(R[-26]C:R[-3]C,">0")	Long	=SUMIF(R[-26]C:R[-3]C,">0")		=SUMIF(R[-26]C:R[-3]C,">0")
31	Total	=SUM(R[-27]C:R[-4]C)		=SUM(R[-27]C:R[-4]C)	Total	=SUM(R[-27]C:R[-4]C)		=SUM(R[-27]C:R[-4]C)	Total	=SUM(R[-27]C:R[-4]C)		=SUM(R[-27]C:R[-4]C)

Figure 2: The spreadsheet excerpt in Figure 1 is now given in the R1C1 style[1]. In the R1C1 style, the corresponding cells in an expandable group usually share the same formula pattern, e.g., L4 vs. L5, and B31 vs. F31.

for three cities, i.e., New York, Chicago and Atlanta. Formally, we describe expandable groups as follows.

Definition 1: An *expandable group* is a triple {*orient*, *unit*, *end*}, where, *orient* is the direction in which the group expands, *unit* is a rectangular area of cells and represents the first block of the group, and *end* is the boundary of the group. It means that, *unit* is expanded along the *orient* to the *end*. In an expandable group, the corresponding cells among all its units prescribe common formats and semantics.

Consider the spreadsheet excerpt in Figure 1. Cells [A2:D31] as the first unit, horizontally expand to column L, and form an expandable group {*hex*, [A2:D31], L}, in which *hex* represents that this group is *horizontally expandable*, and the second element [A2:D31] is the first unit of the expandable group, and L is the last column of the expandable group. Note that, although unit [A2:D31] represents the data for New York and unit [E2:H31] represents the data for Chicago, they only differ in the city names, and their corresponding cells share the same / similar format and semantics: (1) Formula cells in the corresponding cells among units often have the same formula pattern in the R1C1 style[1], e.g., cells B29:B31 and F29:F31 in Figure 2. (2) The corresponding cells among units often have the same format, e.g., cells C6:C8 and G6:G8 in Figure 1 use the same data format, which starts with a symbol "$".

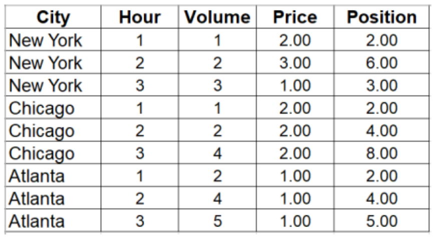
Based on the expanding direction, expandable groups can be further classified into two categories: *vex group* and *hex group*.

vex group (Vertically expandable group): A vex group expands its *unit* in the vertical direction. We represent it as {*vex*, *unit*, *end*}. In Figure 1, cells [A4:L4], as the first unit, expand to row 27, and form a vertically expandable group {*vex*, [A4:L4], 27}.

hex group (Horizontally expandable group): A hex group expands its *unit* in the horizontal direction. We represent it as {*hex*, *unit*, *end*}. In Figure 1, cells [A2:D31] as the first unit, expand to column L, and form a horizontally expandable group {*hex*, [A2:D31], L}.

Expandable groups represent the basic and common structure of a spreadsheet table. To better understand the table structure, we

[1]In spreadsheets, a *cell reference* can be represented in two built-in styles: *A1* and *R1C1*. In the *A1* style, a cell at the *r*-th row and *c*-th column is denoted as *cr* in the relative reference (e.g., A4), and \$*c*\$*r* in the absolute reference (e.g., \$A\$4). In the *R1C1* style, a cell at *x* rows below and *y* columns right to the current cell is denoted as R[*x*]C[*y*] in the relative reference, and a cell at the *x*-th row and *y*-th column is notated as R*x*C*y* in the absolute reference.



City	Hour	Volume	Price	Position
New York	1	1	2.00	2.00
New York	2	2	3.00	6.00
New York	3	3	1.00	3.00
Chicago	1	1	2.00	2.00
Chicago	2	2	2.00	4.00
Chicago	3	4	2.00	8.00
Atlanta	1	2	1.00	2.00
Atlanta	2	4	1.00	4.00
Atlanta	3	5	1.00	5.00

Figure 3: The ideal relational table for the spreadsheet excerpt shown in Figure 1. Note that, only the data in rows 4-6 of Figure 1 are shown due to space limitation.

need to minimize it. Thus, for an expandable group, it should satisfy the following conditions. (1) Its basic unit does not contain any expandable groups. (2) All its units cannot be included by other expandable groups. For example, {*vex*, [A4:L4], 26} satisfies Definition 1, but we do not consider it as an expandable group, since it should contain row 27. {*vex*, [A4:L5], 27} satisfies Definition 1, but we do not consider it as an expandable group, since its basic unit [A4:L5] contains a vex group {*vex*, [A4:L4], 5}. We require that all expandable groups satisfy the above conditions. Note that, a cell can belong to a vex group and a hex group in the same time. E.g., cell A4 belongs to {*vex*, [A4:L4], 27} and {*hex*, [A2:D31], L}.

2.2 Potential Applications of Expandable Groups

The structure of expandable groups can be further applied on important spreadsheet analysis scenarios, e.g., fault detection, and spreadsheet data transformation. This motivates us to effectively detect expandable groups in spreadsheets. We explain these potential application scenarios in the following.

Faults related to expandable groups. In an expandable group, the corresponding cells among units usually share the same formats and semantics. The inconsistencies among the corresponding cells usually indicate faults. The above example contains two different types of faults. (1) ***Missing formulas.*** The missing formula occurs when a cell is supposed to contain a formula, but it does not. For the expandable group {*vex*, [A4:L4], 27}, cells [L4:L27] should follow the same computational semantics, i.e., they have the same formula pattern as shown in Figure 2. However, cells [L25:L27] do not have formulas. In the preparation of this spreadsheet, users may fill the data directly other than their formulas. We can see that cells [L25:L27] should have formulas. According to the expandable group {*hex*, [A2:D31], L}, we can see that cells [D4:D27] and [H4:H27] also suffer from missing formulas. (2) ***Wrong data formats.*** The wrong data format occurs when a cell is supposed to have certain format, but it does not. For example, according to the expandable group {*hex*, [A2:D31], L} in Figure 1, we can see that cells [C4:C5] should have the same format with cells [G4:G5] and [K4:K5], which starts with a symbol "$".

Data transformation between spreadsheets and relational data. Spreadsheet data are usually not organized in a relational way. The spreadsheet excerpt in Figure 1 shows a representative non-relational spreadsheet. This kind of non-relational data cannot be easily consumed by other data analysis tools, e.g., Power BI [31] and Insights in Excel [32]. However, once we can understand these expandable groups, we can easily transform the spreadsheet into relational data, as shown in Figure 3. This new relational data has the same semantics with the original data in Figure 1. However, it can be easily consumed by other data analysis tools. This also highlights the importance of expandable group detection.

2.3 ExpCheck Overview

Detecting expandable groups needs to address three main challenges. The first challenge is how to judge the boundary of the first unit of an expandable group, e.g., cells [A2:D31] in Figure 1. Second, how can we determine the set of features for automated expandable group detection? Different users may have their own styles to tabulate their spreadsheets. Third, in which situation can two continuous units be expanded? Some corresponding cells may have different formats, e.g., cells C5 and C6 in Figure 1. Note that all corresponding cells with the same formats do not necessarily suggest that they are expandable. Even we assume that cells [C3:C27] and [D3:D27] had the same format (i.e., if cells [C6:C8] do not start with a symbol "$"), they do not form an expandable group, since they have different semantics.

For the first challenge, we observe that in an expandable group, its units usually occupy a whole row / column of a table. For example, in Figure 1 we consider cells [A4:L4] as a potential unit. We can further generate larger unit by considering multiple rows / columns into a unit. For the second challenge, although different users may use different styles, units in an expandable group usually share the same formats. In this case, we define a set of features for each cell, and then validate whether two units share the same features. For the third challenge, we do not require that all corresponding cells must share the same formats and semantics. Our feature model in Section 3.3 can tolerate some differences. Further, we need to check the semantics of cell headers, and check whether corresponding headers share the similar semantics and belong to the same category. For example, in Figure 1, the headers "New York", "Chicago" and "Atlanta" are all US cities and belong to the same category. We adopt Word2Vec [26] to calculate the semantic similarity among headers. High similarity of two headers indicates that they should belong to the same category.

3 EXPANDABLE GROUP IDENTIFICATION

Given a spreadsheet, ExpCheck analyzes it and reports all detected expandable groups. ExpCheck works in three steps First, it classifies cells into four types, i.e., *data*, *formula*, *label* and *empty*. Based on cell types, it further identifies tables in the spreadsheet, which contain semantically related data and computations (Section 3.1). Second, for each table, it extracts the first few rows / columns, which can be used as headers to describe the data in the table (Section 3.2). Third, it extracts expandable groups based on format and semantic features (Section 3.3).

3.1 Table Identification

We observe that spreadsheet users may put some unrelated tables into a worksheet (a spreadsheet may contain multiple worksheets). An expandable group usually lies into one table, and tables can be used as the boundary of expandable groups. Therefore, we need to identify tables first. In spreadsheets, different tables are usually circumscribed by empty cells. Thus, we first determine each cell's type. Then, we identify tables based on cell types.

3.1.1 Cell Classification. We follow the approaches described by Hermans et al. [19] and Abraham et al. [1] to classify cells into four types: (1) data cells: numerical cells with plain values; (2) formula cells: cells that contain formulas; (3) label cells: cells that contain strings; and (4) empty cells.

Our cell classification algorithm works as follows. First, all numerical cells with formulas are marked as *formula cells*. Second, all numerical cells without formulas are marked as *data cells*. Third, all non-formula cells referenced by formula cells are marked as *data cells*, no matter whether they are empty or not. Fourth, for all remaining cells, they are classified as *label cells* when not empty, and *empty cells*, otherwise.

Take the spreadsheet excerpt in Figure 1 as an example. Our cell classification marks cells [A4:K27] and [L25:L27] as *data cells*, cells [L4:L24] and other cells with formulas as *formula cells*. Cells [A28:L28], [C29:C31], [G29:G31] and [K29:K31] are marked as *empty cells*. All cells with strings are marked as *label cells*, e.g., A3.

3.1.2 Extracting Tables. In spreadsheets, a table represent a cell area, in which related information is put together. Tables are usually connected cell areas divided by empty cells. A connected cell area is defined as a rectangle containing data, label and formula cells [1][19]. In a connected cell area, two cells are connected if they can touch horizontally, vertically or diagonally. The basic algorithm to find connected areas is as follows. (1) It finds the left-most upper-most non-empty cell that is not contained by any cell area yet. The cell is considered as the initial cell of a new cell area. (2) This cell area is expanded by checking all cells that are connected to it in all directions. If one of the cells are not empty, the cell area is expanded to include this cell. (3) When all cells connected to the cell area are empty, we get a new cell area. Simply, existing work [1][19] consider these connected cell area as tables.

However, the above algorithm usually obtains only part of a table, rather than a complete table. Take the spreadsheet excerpt in Figure 1 as an example. This spreadsheet excerpt only includes one table [A1:L31]. Due to empty cells in this table, e.g., empty cells [A28:L28], the above algorithm divides it into 5 connected cell areas, i.e., [A1:L27], [A29:B31], [D29:F31], [H29:J31] and [L29:L31]. We can see that the table [A1:L31] is divided into 5 pieces. Thus, the above table identification approach cannot obtain complete tables. In our investigation on the spreadsheets from the EUSES and Enron corpora, such case is very common. It is because a table usually has several pieces, and spreadsheet users are used to putting empty cells to separate them.

To tackle the issues caused by empty cells, we further design two new strategies as follows to identify tables.

a) The borders of two cells are indicators about whether the two cells are connected. Consider the spreadsheet excerpt in Figure 1. The table [A1:L31] is surrounded by the bold border, which is different from the default border in the table. Thus, when we check whether cells L27 and L28 are connected, we check whether these two cells have the same bold border in the right. If they do, we consider them as connected.

b) Although some tables are separated into pieces by empty cells, they may share the same headers, and thus they should be put into one table. For example, two pieces [A1:L27] and [A29:L31] are separated by row 28. The second piece [A29:L31] shares the same column headers as [A1:L27], i.e., rows 1-3. Thus, these two pieces should be combined into one. If cells [A29:L31] have their own column headers, we will not combine them into one.

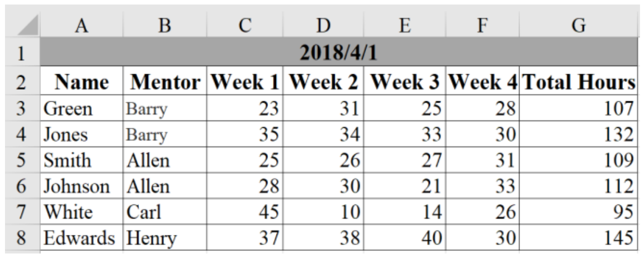
We use the example in Figure 1 to show how to design the second strategy. After we obtain the connected cell area [A1:L27] based on the basic table identification algorithm, we skip row 28 in which all cells are empty and obtain a new cell area initialized as [A29:L29]. This initialized cell area has the same start and end columns with [A1:A27]. We then use the basic table identification algorithm to expand this cell area, and obtain a new cell area [A29:L31]. We further check whether this cell area [A29:L31] has column headers based on the header inference in Section 3.2. In this example, cell area [A29:L31] does not have column headers and has the same width as [A1:L27]. Therefore, we combine it into the cell area [A1:A27]. Otherwise, we do not combine these two cell areas. We can also check neighboring cells in the right, which is similar to checking neighboring cells in the bottom.

3.2 Header Identification

Although spreadsheet systems allow users to put table headers in any place, by investigating amounts of spreadsheets in the EUSES and Enron corpora, we observe that, in most cases, spreadsheet users put the headers in the first few rows / columns of a table. Figure 4 shows a typical example in which the first two rows are used as column headers of the table. Note that, although cells [A3:A8] are label ones, they are not considered as row headers of the table.

The headers of a table are usually used to describe the data in the table. We have two observations that can help locate headers for a table. (1) The headers of a table can usually be found on the left or top of a table as illustrated in Figure 4. Especially, we assume the headers are usually located in the first few (e.g., 4) rows / columns in a table. (2) Spreadsheet users usually use different formats for headers and data region. For example, the first row has grey background, and the first two rows have bold fonts in Figure 4. They are different from the data region, i.e., cells [A3:G8].

Based on the above observations, we design a novel approach to detect headers in a table. We assume that, for a table, at most *max_header* rows and columns can be used as headers. Based on our inspection on the EUSES and Enron corpora, we set *max_header* to 4. This can cover almost all cases. Our header inference approach works as follows.



	A	B	C	D	E	F	G
1	2018/4/1						
2	Name	Mentor	Week 1	Week 2	Week 3	Week 4	Total Hours
3	Green	Barry	23	31	25	28	107
4	Jones	Barry	35	34	33	30	132
5	Smith	Allen	25	26	27	31	109
6	Johnson	Allen	28	30	21	33	112
7	White	Carl	45	10	14	26	95
8	Edwards	Henry	37	38	40	30	145

Figure 4: A spreadsheet excerpt that computes the working hours in April 2018. In this excerpt, row 2 is the first-level column headers, and row 1 is the second-level column headers. Cells [A3:A8] are data, and not considered as headers.

a) Type-based header inference. To infer the column headers of a table, we start from the first row of the table, and check whether the checked row can be used as column headers, until we have checked *max_header* rows, or the checked row is not considered as column headers. If the checked row contains at least one label cell, and all other cells in it are empty, we consider it as column headers of the table. It is similar for row header inference. Based on this rule, we obtain that the first 2 columns (A & B) in Figure 4 are row headers. There is no column header, since the date in the first row is consider as data, not a label.

b) Format-based header inference. As discussed earlier, headers usually have different format from the data region, and the data region usually have the same format. Based on this observation, we use the following rule to detect headers: For any given three continuous rows or columns ($x1$, $x2$ and $x3$), if $x1$ has different format from $x2$, and $x2$ has the same format with $x3$, we consider $x1$ and rows / columns before $x1$ as headers, $x2$ and $x3$ as data region. We check this rule from the first row / column of the table, until we find three rows / columns that satisfy the above rule, or *max_header* rows / columns are reached. For example, in Figure 4, row 2 has different format from row 3, and row 3 & row 4 have the same format. Thus, rows 1 and 2 are considered as column headers. Here, we use the format features discussed in Section 3.3.2 to calculate whether 2 rows / columns have the same format.

Note that, any of the above two strategies may fail in some cases. For the first strategy, it does not work when there are data cells in the headers, e.g., the date in row 1 in Figure 4. For the second strategy, if the first two data rows (e.g., rows 3 and 4) have slightly different format, it may fail. Thus, we combine the results from above strategies, and consider all detected headers from these two strategies as headers. For the table in Figure 4, we get the first two rows as column headers, and first two columns as row headers.

c) Mixed header inference. The above process may wrongly judge situations. For example, in Figure 4, cells [A3:B8] are considered as row headers. In fact, they are data. In this case, we use the third strategy to further refine headers. As discussed earlier, header cells usually have different format from data cells. If we observe that the cells in headers have the same format as data cells, it should be considered as data, rather than headers. For example, in Figure 4, cells [B3:B8] have the same format with the data cells [C3:C8]. Thus, we consider [B3:B8] as data, rather than headers. Through this strategy, we finally obtain that, in Figure 4, the first two rows are used as column headers, and there are no row headers.

After the above three strategies, we may obtain multiple rows that are used as column headers. We consider the row nearest to data area as the first-level column headers, and the row next to the first-level column headers as the second-level column headers, and so on. It is similar for row headers. Take the spreadsheet in Figure 4 as an example, row 2 is considered as the first-level column headers, and row 1 is considered as the second-level column headers.

Note that our above header identification algorithm differs from previous approaches [1][19]. First, previous approaches mainly use the idea in the first strategy. Thus, they suffer from the issues we discuss earlier. For example, they consider cell A3 as the row header of cell C3. Worse, they consider cell B3 as the header of cell C3, too. Second, they try to find the headers for each individual cell, and do not consider all data cells as a whole. Thus, the inferred headers may locate in different rows / columns. Our approach avoids these issues.

3.3 Expandable Group Detection Algorithm

Our expandable group detection algorithm is inspired by two key observations: (1) For each unit in an expandable group, their corresponding cells share the same or similar formats. (2) For each unit in an expandable group, their corresponding header cells share the same or similar semantics. Thus, in the following, we first explain how to compare two corresponding columns among two units in a hex group, and then explain our expandable group detection algorithm in detail. Note that, for hex and vex groups, they have the similar detection algorithm. The only difference between hex and vex group detection is their directions. Thus, for clarity, we only use hex group detection to explain our algorithm.

3.3.1 Expandable Group Detection. ExpCheck's expandable group detection algorithm takes a table as input and returns all expandable groups in the table. Algorithm 1 shows how to detect hex groups in a table. The algorithm works as follows. (1) It first detects hex groups in which its unit only have one column (Lines 2-3). (2) For the table, it starts to check possible unit from its first column and generates a unit for the first column (Lines 4-6). (3) It tries to horizontally expand the unit to the right. Each time, it generates a candidate unit2 that has the same size as unit, and checks whether two units (unit and unit2) have similar formats and semantics. If yes, unit can be horizontally expanded to the right. Otherwise, unit cannot be further expanded to the right (Lines 7-16). (4) Once we find a unit can be horizontally expanded at least once, we find a hex group, and add it into groups (Lines 17-23). (5) The algorithm continues to find hex group with a unit of width columns (Lines 5-24). (6) Once all potential units with width columns have been checked, we check whether all detected hex groups have covered most columns. If yes, we do not search for hex groups with larger width. Otherwise, we increase width, and try to find new hex groups (Lines 25-27).

Note that, when there are multiple-level column headers, we generate the *unit* from the first-level column headers first, and then try to include the higher-level column headers (Line 33). We take the spreadsheet excerpt in Figure 1 as an example to further explain the above algorithm. For this example, the input table is [A1:L31]. Since there are three levels of column headers (rows 1-3), we first consider the first-level column headers (row 3) to be

Algorithm 1. hex group detection algorithm.

```
Input: table (spreadsheet table), startRow (the first row that
       can be included in a hex group)
Output: groups (all detected hex groups).
 1:  groups = EMPTY;
 2:  width = 1; // Number of columns in the unit
 3:  while width < table.colNum / 2 do
 4:    startCol = table.firstCol; // Start from first column
 5:    while startCol < table.lastCol do
 6:      unit = generateUnit(startCol, witdth, 0);
 7:      repeat = 1; // Number of expanding units
 8:      while TRUE do
 9:        // Generate the potential expanding unit
10:        unit2 = generateUnit(startCol, width, repeat);
11:        if (unit2 == NULL || !isSimilar(unit, unit2)) then
12:          break;
13:        else
14:          repeat++;
15:        end if
16:      end while
17:      if (repeat > 1) then  // Find a hex group
18:        end = startCol + repeat * width - 1;
19:        groups.add({hex, unit, end});
20:        startCol += repeat * width;
21:      else
22:        startCol ++
23:      end if
24:    end while
25:    if terminate(groups) then
26:      break;
27     else
28:      width++;
29:    end if
30:  end while
31:  return groups;
32:
33:  // Generate a potential expanding unit
34:  method generateUnit(startCol, width, repeat)
35:    unit.firstRow = startRow;
36:    unit.lastRow = table.lastRow;
37:    unit.firstCol = startCol + repeat * width;
38:    unit.lastCol = startCol + (repeat + 1) * width;
39:    if (unit.lastCol > table.lastCol) then
40:      return NULL;    // No unit generated
41:    end if
42:    return unit;
43:  end method
```

included in *unit*. (1) When *width* is 1, it starts to check whether units [A3:A31] and [B3:B31] are similar. For this case, they are not similar (we will explain why in Section 3.3.2). Thus, we start to check units [B3:B31] and [C3:C31], and so on. Finally, we cannot detect any hex group when *width* is 1. Thus, we gradually increase *width* to 2, 3, and 4. Finally, when *width* is 4, we find units [A3:D31], [E3:H31] and [I3:L31] are similar. Thus, we get a hex group {*hex*, [A3:D31], L}. For now, we find this hex group have covered all columns in table [A1:A31], thus, we do not further increase *width* to 5. After this, we further check whether the second-level column headers (row 2) can be included into hex groups. Since cells [A2:D2] is a merged cell, we directly set *width* to 4, and check whether [A2:D31] can be expandable. In this case, we obtain a new hex group {*hex*, [A2:D31], L}. Since this new hex group covers {*hex*, [A3:D31], L}, we only keep this new hex group. Next, we try to include the third-level column headers (row 1). But we cannot find any hex group in this case. Finally, our algorithm returns {*hex*, [A2:D31], L}.

3.3.2 Similarity among Two Units. The key part of our expandable group detection is how to inspect whether two units share the similar formats and semantics. Algorithm 2 shows how to judge whether two units have similar semantics and formats in a hex group. The algorithm takes two units that potentially belong to a hex group as input. For each pair of corresponding columns of the two units, it compares each pair of cells in them (Lines 3-14). If the cells belong to the column headers, we need to compare whether they have the similar semantics (Lines 6-10). If their semantics are different, they cannot be considered as similar (Lines 7-9). For all the cells, we compare whether they have the same formats (Lines 11-13). To tolerance the possible difference in an expandable group, we require that a fixed percent (min_similar_cells) of cells have the same formats. In our experiment, we set min_similar_cells as 60%, which indicates that most cells have the same formats.

The features used in ExpCheck are collected from existing literatures [6][7][25]. If two cells do not have the same value for a feature, we consider them as dissimilar. Note that, if a cell is empty, we cannot obtain its features. In this case, we assume that the empty cell can have any kind of features and is similar to other non-empty cells. Thus, we can ignore the noise caused by empty cells. We divide all features into 4 categories as follows.

a) Header: A header presents the semantic information of a groups of cells. For example, in Figure 1, "New York" shows the property of cells [A3:D31], and "Hour" in cell A3 shows cells [A4:A27] are 24 hours. For any corresponding headers in an expandable group, they should have similar semantics and can be categorized into the same category. By calculating the semantic similarity of two headers based on Word2Vec [26], we judge whether two headers can belong to the same category.

Generally, a header is a word or a short string. We use *h1* and *h2* to denote two headers, and function *word_similarity* to denote the cosine similarity among two words in Word2Vec. Thus, the similarity of two headers can be calculated as follows:

$$h1 =< w1_i, w1_2, \dots w1_m >, h2 =< w2_i, w2_2, \dots w2_n >$$

$$sim1 = \sum_{i=1}^{i=m} \max_{1\le j\le n}(word\_similarity(w1_i, w2_j))$$

$$sim2 = \sum_{i=1}^{i=n} \max_{1\le j\le m}\left(word\_similarity(w2_i, w1_j\right)$$

$$sim(h1, h2) = \frac{sim1 + sim2}{m + n}$$

When the similarity of two headers *sim*(*h1*, *h2*) exceeds a threshold (*min_header_similarity*), we consider that they are similar. To guarantee high accuracy in our approach, the threshold is set to strong similarity. In our experiment, we set *min_header_similarity* as 0.3.

b) Cell type: As discussed earlier, cells can be divided into four types: *data*, *label*, *formula*, and *empty*. For two cells, if one is *label* cell, and another is *data* or *formula* cell, we consider them as dissimilar. For other cases, we consider them as similar. Thus, we can tolerate some inconsistent cases. For example, in Figure 1, we consider cells D4 and L4 as similar, even though cell D4 does not have a formula.

c) Cell formula: Formulas are a common feature to explain a cell's computational semantics. For two formula cells, if they have the same formula pattern, i.e., they have the same formula in the R1C1 style, they are considered as similar, otherwise not.

d) Format: The formats of a cell present how a cell is shown in a table. Ideally, the corresponding cells in an expandable group should have consistent formats. The format features we use are as follows.

- **Merge.** We record if a cell is merged, and its merged region, if any. For two cells, if they are both merged, and their merged regions have the same size, they are considered as similar. If they are not merged either, they are considered as similar, too. Otherwise, they are considered as dissimilar.
- **Fill color.** We check whether two cells are filled by the same color or not.
- **Font.** The font features describe whether two cells have the same font name, size, color, weight of bold, etc. It also records italic, underlines. If there is any difference among these features, two cells are considered as dissimilar.
- **Data format.** Data format presents how a data is shown. E.g., cells C6 and D6 are shown in different format in Figure 1.
- **Alignment.** The type of horizontal and vertical alignment, and the number of indentations.

Algorithm 2. Similarity analysis among two units in hex groups.

```
Input: unit1 and unit2 (two units have the same size and poten-
       tially belong to the same expandable group).
Output: TRUE or FALSE (whether two units are similar).
 1:  for (col = 1; col <= unit1.colNum; col++) do
 2:    simCell = 0; // Number of similar cells in column col
 3:    for (row = 1; row <= unit1.rowNum; row++) do
 4:      cell1 = unit1.getCell(row, col);
 5:      cell2 = unit2.getCell(row, col);
 6:      if (cell1 is a header) then
 7:        if (!similarSemantic(cell1, cell2)) then
 8:          return FALSE;
 9:        end if
10:      end if
11:      if (sameFormat(cell1, cell2)) then
12:        simCell++
13:      end if
14:    end for
15:    pert = simCell / unit1.rowNum;
16:    if (pert < threshold)
17:      return FALSE;
18:    end if
19:  end for
20:  return TRUE;
```

4 IMPLEMENTATION

This section briefly explains some necessary implementation details. Our ExpCheck uses the Apache POI library [33] to read Excel files. ExpCheck loads an Excel file, reads features in each cell, and analyzes all expandable groups in it.

In order to calculate the similarity among words, we use pre-trained Word2Vec model [34] published by Google [35]. This pre-trained model contains 300-dimensional vectors for 3 million words and phrases, trained on Google News dataset (about 100 billion words). Since the words in the model are case-sensitive, we first retrieve a word from the model in a case-insensitive way, and then use the word to get the word's vector. For phases, each word in a phase are connected by "_". In this case, we first try to connect words in a header using "_", and check whether the combination word exists in the pre-trained model. If yes, we consider the header as a phase, otherwise, we consider it as a sentence. For example, we first transform "New York" into "New_York", and then check whether the model contain "New_York". Since we find this new combined word, we consider it as a phase.

To transform a sentence into words, we use the special characters (e.g., white space, -, comma, colon, semicolon, and so on) to separate a sentence. Note that, all numbers in the sentence are considered as a unique word, and do not distinct their concrete values. Since, headers are usually words and short phrases, there are very few stop words (e.g., a, the) in them. Thus, we do not handle stop words in our tool.

5 EVALUATION

Our evaluation studies the following two research questions:

RQ1: Can ExpCheck detect expandable groups precisely? Specifically, what are the precision, recall and F1-measure?

RQ2: How is ExpCheck compared with existing techniques?

To answer RQ1, we ran ExpCheck on 120 spreadsheets randomly sampled from the EUSES [16] and VEnron [13] corpora, and checked their performance. To answer RQ2, we compared ExpCheck with the expandable group analysis approach proposed by Abraham et al. [2] in the detection of expandable groups.

5.1 Experimental Subjects and Methodology

We used spreadsheets from the EUSES and VEnron corpora to conduct our experiments. First, EUSES was created in 2005, and has been widely used by spreadsheet research [8][12][22]. Second, VEnron was created from Enron [17], and contains many versioned spreadsheets used in the Enron company. We used VEnron other than Enron as our subject, since Enron contains too many spreadsheets with the same or similar structures. This may weaken the effectiveness of our evaluation. These two spreadsheets corpora have been the most widely used for spreadsheet research so far.

Manually building ground truth for all expandable groups in the EUSES and VEnron spreadsheets is extremely difficult, as we are not the authors of these spreadsheets and we cannot find their corresponding authors, either. Thus, we randomly sampled 70 spreadsheets from EUSES and 50 spreadsheets from VEnron. To make our experimental subjects accurate and representative for expandable group analysis, we built our ground truth by the following steps. (1) For each randomly sampled spreadsheet, we inspected each of its contained worksheets for expandable groups. (2) A spreadsheet may contain multiple worksheets that have the same or very similar structure. To guarantee the structure diversity of our experimental subjects, for the worksheets with the same or very similar structure, we only randomly kept one of them. (3) For each remaining worksheet that does not contain any expandable group or cannot be fully understood by any one of us (two authors in this paper), we removed it from consideration. If no worksheet in a spreadsheet was left after the previous steps, we removed it from consideration. (4) We repeated the above sampling process until a fixed number of spreadsheets were sampled.

Note that, sampling spreadsheets from VEnron is slightly different from that of EUSES. Since each evolution group in VEnron contains multiple versions of a spreadsheet, we only kept the first spreadsheet in each group before performing the above sampling steps.

In our sampling process, we manually inspected the sampled spreadsheets, and tried our best to understand the semantics of the spreadsheets, and finally labelled all expandable groups in them. The ground truth was built carefully by cross-validating all expandable groups by two authors of this paper. Finally, we obtained 313 expandable groups from 120 spreadsheets. Table 1 gives the statistics of our experimental subjects. As shown in the table, the experimental subjects are diverse: 70 EUSES spreadsheets cover 8 categories and 50 spreadsheets come from VEnron. Among these 313 expandable groups, 221 groups are vertically expandable ones (i.e., vex groups), and 92 groups are horizontal expandable ones (i.e., hex groups). We have made our experimental subjects available online for future research: http://www.tcse.cn/~wsdou/project/ExpCheck.

Table 1: Statistics of Our Experimental Subjects

Corpus	Category	Spreadsheet	Worksheet	Expandable groups		
				vex	hex	Total
EUSES	cs101	1	1	2	1	3
	database	11	12	19	8	27
	financial	10	10	12	5	17
	forms3	1	1	1	0	1
	grades	5	5	7	3	10
	homework	13	15	23	13	36
	inventory	17	17	36	5	41
	modeling	12	12	12	1	13
VEnron		50	66	109	56	165
Total		**120**	**139**	**221**	**92**	**313**

5.2 Evaluation Metrics

We ran ExpCheck on these spreadsheets and checked its performance. Expandable group detection can have three outcomes: (1) A detected expandable group is the same as one of expandable groups in the ground truth (TP). (2) A detected expandable group is not found in the ground truth (FP). (3) An expandable group in the ground truth is not detected (FN). To study the effectiveness of our expandable group detection approach and compare it with existing approaches, we use the following three metrics:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$F1 - measure = \frac{2 \times precision \times recall}{precision + recall}$$

5.3 Expandable Group Detection Results

We ran ExpCheck on 120 spreadsheets sampled from the EUSES and VEnron corpora. Table 2 lists our expandable group detection results. It gives the numbers of detected expandable group in each category in EUSES and VEnron. For each kind of expandable groups, Table 2 lists the numbers of detected expandable groups (Detected) and the numbers of true expandable groups (True).

In total, ExpCheck detects 349 expandable groups (ExpCheck/Total/Detected) and 242 (69.3%) expandable groups are true. We further evaluated the performance on vex groups (ExpCheck/vex) and hex groups (ExpCheck/hex), respectively. From Table 2, the precision for vex group detection is 80.1%, whereas the precision for hex group detection is 49.6%. We can see

Table 2: Expandable Detection Results of ExpCheck and ViSTL

Corpus	Category	Ground truth	ExpCheck						ViSTL	
			vex		hex		Total			
			Detected	True	Detected	True	Detected	True	Detected	True
EUSES	cs101	3	2	2	2	1	4	3	3	2
	database	27	21	17	8	6	29	23	6	1
	financial	17	10	3	8	3	18	6	6	1
	forms3	1	1	1	0	0	1	1	0	0
	grades	10	8	7	2	2	10	9	10	2
	homework	36	23	23	15	6	38	29	12	5
	inventory	41	37	33	7	2	44	35	4	2
	modeling	13	13	6	5	0	18	6	0	0
VEnron		165	111	89	76	41	187	130	68	15
Total		**313**	**226**	**181 (80.1%)**	**123**	**61 (49.6%)**	**349**	**242 (69.3%)**	**109**	**28 (25.7%)**

the precision for vex group detection is much better than that of hex group detection.

ExpCheck misses 71 (313-242) expandable groups, and thus the recall for expandable group detection is 77.3%. We further analyzed the recalls for vex group detection and hex group detection, and they are 81.9% and 66.3%, respectively. Similarly, ExpCheck's F1-measure for expandable group detection, vex group detection and hex group detection is 73.1%, 81.0%, and 56.7%, respectively. We can see that vex group detection performs better than hex group detection.

False positives of expandable group detection. ExpCheck wrongly detected 107 (349 − 242) expandable groups. We further investigated the causes for these 107 false positives. There are 3 main reasons. (1) We use Word2Vec to compute the semantic similarity between two headers. Although some headers with the similar semantics (e.g., buy and sell), they are not expandable. In some other cases, spreadsheet users use some domain-specific words. Word2Vec cannot work in this situation. 54 false positives belong to the above case. (2) In an expandable group, some rows / columns use different styles, e.g., a row in vex group use different colors with others. In this case, ExpCheck divides it into multiple vex groups. 20 false positives belong to this case. (3) Our header inference is rule-based, and it cannot be applied on some specific styles of spreadsheets. Thus, we can obtain wrong headers. 26 false positives belong to this case. The remaining false positives are caused by various cases, e.g., wrongly identified table range. This indicates further study can improve expandable group detection from the above aspects.

False negatives of expandable group detection. ExpCheck misses 71 (out of 313) expandable groups. The reasons for missed expandable group detection are the same with false positives of expandable group detection. 36 missed expandable groups belong to the semantic similarity issues among headers. 13 missed expandable groups belong to different formats in an expandable group. 12 missed expandable groups belong to wrong header inference. The remaining groups are caused by various cases.

Note that vex group detection performs much better than hex group detection. This is because vex groups usually do not have row headers (e.g., {*vex*, [A4:L4], 27} in Figure 1), whereas hex groups usually have column headers (e.g., {*hex*, [A2:D31], L} in Figure 1). The issues in semantic similarity of headers are the key cause to introduce false positives and false negatives.

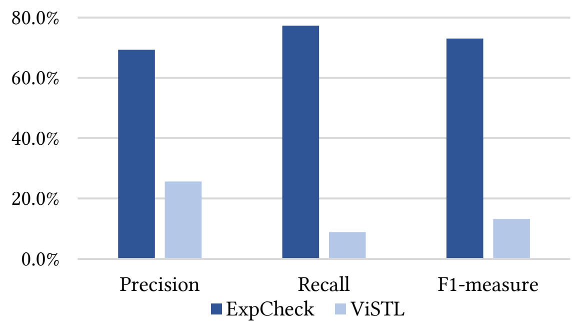


Figure 5: Performance comparison of ExpCheck and ViSTL.

Based on the above results and analysis, we draw the following conclusion to RQ1: *Our expandable group detection approach is effective. It can detect expandable group in a high precision and recall.*

5.4 Comparison with Existing Techniques

To better evaluate the effectiveness of our ExpCheck in expandable group detection, we compared ExpCheck with the template inference approach for ViSTL (ViSTL for short) [2].

ViSTL mainly focuses on formula cells and cells' types. First, ViSTL finds the repeating patterns of formula cells based on whether they have the same formula in the R1C1 style. Then, ViSTL checks if other corresponding cells follow the same types. However, ViSTL has a few drawbacks. (1) There must be formulas for expandable groups. However, this is not true. In our ground truth, many expandable groups do not have any formula. (2) All related cells should have formulas. Once some formulas are missing, the obtained expandable group will be incomplete. As discussed earlier, ViSTL will get a vex group {*vex*, [A4:L4], 24}. This group is incomplete. (3) ViSTL does not try to understand the real semantics, and thus can extract groups with unrelated semantics. (4) ViSTL does not check formats at all. Therefore, it may miss some expandable groups, and detects wrong expandable groups.

We ran ViSTL on 120 spreadsheets and evaluated its performance. The last columns in Table 2 shows the detected result for ViSTL. ViSTL detected 108 expandable groups, and only 28 are true. Thus, ViSTL achieved a precision of 25.7%, recall of 8.9%, and F1-measure of 13.3%. Figure 5 shows the performance comparison between ExpCheck and ViSTL. We can see that ExpCheck performs much better than ViSTL. We improve precision of 43.7%, recall of 68.4% and F1-measure of 59.8%.

Based on these results, we draw the following conclusion for RQ2: *ExpCheck significantly outperforms existing expandable group detection approaches.*

6 DISCUSSION

While our evaluation shows that ExpCheck is promising for detecting expandable groups, we discuss potential threats and limitations in our approach.

6.1 Threats to Validity

Representativeness of experimental subjects. In our experiments, we selected the EUSES and VEnron corpora as our study objects. First, the spreadsheets in these two corpora have been widely used for spreadsheet research [8][10][22]. Second, the spreadsheets in VEnron came from the Enron company. Thus, we believe our studied subjects represent real-world spreadsheets.

Expandable group validation. There is not any clue about expandable groups in the EUSES and VEnron corpora. It is also impossible to inspect expandable groups with the help of their original users. Thus, we manually inspected our studied subjects, and built ground truth for expandable groups. To alleviate possible mistakes, two authors of this paper cross-validated all ground truth.

Parameter setting in our approach. In our approach, several parameters are preset according to our experience, e.g., the threshold for semantic similarity of headers, the percentage of cells that should have consistent formats. This may affect our evaluation results. We do not know whether the current parameter setting can obtain optimal results. We will study how to obtain better parameter setting in the future, e.g., conducting empirical studies on large-scale spreadsheets to choose better parameters.

6.2 Limitations

Domain-specific vocabularies in spreadsheets. During our inspection of the spreadsheets in VEnron, we find that there are some domain-specific vocabularies in the Enron company, which cannot be handled properly by Word2Vec. For now, we cannot handle these domain-specific vocabularies yet, and thus introduce false positives and false negatives. A new approach that can analyze the semantic similarity among these domain-specific vocabularies is required.

Different language support. For different languages, we need different word segmentation models and different Word2Vec models. For now, ExpCheck only supports English. Other languages can be integrated into ExpCheck.

7 RELATED WORK

In the section, we discuss related work on spreadsheet research, e.g., modeling, structure analysis and fault detection.

Spreadsheet modeling. Spreadsheet systems provide high flexibility for users in building spreadsheets. However, this flexibility can easily induce faults into spreadsheets. Spreadsheet researchers proposed several rigorous models for spreadsheets, e.g., ViSTL [4][5] and ClassSheet [14], to help users reduce the chance of introducing faults. However, it is challenging to construct such models. Thus, some approaches [2][9][18] try to build models from existing spreadsheets. They mainly depend on the formulas in the existing spreadsheets, and do not try to understand the spreadsheet structure, e.g., expandable groups. Our expandable group detection can be used to improve spreadsheet model reversing engineering.

Spreadsheet structure analysis. Unlike relational data, spreadsheets have very flexible structures. Understanding spreadsheet structures is the key to other spreadsheet analysis approaches, e.g., spreadsheet data integration. For example, Chen et. al. adopted conditional random fields (CRF) to infer hierarchical relationship among spreadsheet cells [6]. They further proposed an active learning framework to detect the structure property in a spreadsheet. Koci et al. [25] proposed a classification approach to discover the layout of tables in spreadsheets on the cell level. However, these approaches cannot detect expandable groups.

Spreadsheet fault detection. Since spreadsheets are created and maintained by non-expert end users, faults can be easily induced into spreadsheets [23][24][27]. Many techniques have been proposed to detect faults in spreadsheets. UCheck [3] uses type system to check type inconsistency in formulas. Hermans et al. proposed to detect inter-worksheet smells [20], data clone and related inconsistencies [22], and formula smells [21]. TableCheck [10] detects inconsistency among table clones. Some fault detection tools utilize certain structure information to detect errors, e.g., AmCheck/CACheck/EmptyCheck [11][12][29] and CUSTODES [8] detect errors in a group of similar cells (i.e., cell array). However, they do not try to understand expandable groups in a table. While, ExpCheck focuses on understanding the key structure in a table.

8 CONCLUSIONS

In this paper, we study expandable groups in spreadsheets, in which their units share the similar formats and semantics. We have proposed an automated approach, ExpCheck, to extract expandable groups by inspecting related cells' format and semantic information. Our experimental study on the spreadsheets from the EUSES and VEnron corpora shows that our proposed approach is effective and precise, and significantly outperforms existing approaches.

We plan to pursue our future work in four ways. First, ExpCheck can be improved by more precise spreadsheet header analysis and header semantic analysis. Second, ExpCheck can also be extended to extract other spreadsheet structures and build the whole structure for a spreadsheet. Third, we plan to use the extracted expandable groups to detect issues in spreadsheets, e.g., formula errors and format errors. Fourth, we plan to use ExpCheck to understand spreadsheet structures, and transform spreadsheets data into relational data for easy data integration.

ACKNOWLEDGMENTS

This work was partially supported by National Natural Science Foundation of China (61702490), Microsoft Research Asia Star-Track Program, National Key Research and Development Program of China (2017YFA0700603), Frontier Science Project of Chinese Academy of Sciences (QYZDJ-SSW-JSC036), and Youth Innovation Promotion Association at Chinese Academy of Sciences.

REFERENCES

[1] Robin Abraham and Martin Erwig. 2004. Header and Unit Inference for Spreadsheets through Spatial Analyses. In *IEEE Symposium on Visual Languages and Human Centric Computing (VL/HCC)*, 165–172.

[2] Robin Abraham and Martin Erwig. 2006. Inferring Templates from Spreadsheets. In *International Conference on Software Engineering (ICSE)*, 182–191.

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

[4] Robin Abraham, Martin Erwig, Steve Kollmansberger, and Ethan Seifert. 2005. Visual Specifications of Correct Spreadsheets. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 189–196.

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

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

[7] Zhe Chen, Sasha Dadiomov, Richard Wesley, Gang Xiao, Daniel Cory, Michael Cafarella, and Jock Mackinlay. 2017. Spreadsheet Property Detection With Rule-assisted Active Learning. In *ACM on Conference on Information and Knowledge Management (CIKM)*, 999–1008.

[8] 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 *International Conference on Software Engineering (ICSE)*, 464–475.

[9] Jacome Cunha, Martin Erwig, and Joao Saraiva. 2010. Automatically Inferring ClassSheet Models from Spreadsheets. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 93–100.

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

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

[12] 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.

[13] 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 *International Conference on Software Engineering (ICSE SEIP)*, 162–171.

[14] Gregor Engels and Martin Erwig. 2005. ClassSheets: Automatic Generation of Spreadsheet Applications from Object-oriented Specifications. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 124–133.

[15] Martin Erwig, Robin Abraham, Steve Kollmansberger, and Irene Cooperstein. 2006. Gencel: A Program Generator for Correct Spreadsheets. *Journal of Functional Programming (JFP)* 16, 3 (2006), 293–325.

[16] Marc Fisher and Gregg Rothermel. 2005. The EUSES Spreadsheet Corpus: A Shared Resource for Supporting Experimentation with Spreadsheet Dependability Mechanisms. *ACM SIGSOFT Software Engineering Notes* 30, 4 (2005), 1–5.

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

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

[19] Felienne Hermans, Martin Pinzger, and Arie van Deursen. 2011. Supporting Professional Spreadsheet Users by Generating Leveled Dataflow Diagrams. In *International Conference on Software Engineering (ICSE)*, 451–460.

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

[21] Felienne Hermans, Martin Pinzger, and Arie van Deursen. 2012. Detecting Code Smells in Spreadsheet Formulas. In *International Conference on Software Maintenance (ICSM)*, 409–418.

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

[23] Dietmar Jannach, Thomas Schmitz, Birgit Hofer, and Franz Wotawa. 2014. Avoiding, Finding and Fixing Spreadsheet Errors – A Survey of Automated Approaches for Spreadsheet QA. *Journal of Systems and Software (JSS)* 94, (2014), 129–150.

[24] Andrew J. Ko, Robin Abraham, Laura Beckwith, Alan Blackwell, Margaret Burnett, Martin Erwig, Chris Scaffidi, Joseph Lawrance, Henry Lieberman, Brad Myers, Mary Beth Rosson, Gregg Rothermel, Mary Shaw, and Susan Wiedenbeck. 2011. The State of the Art in End-user Software Engineering. *ACM Computing Surveys* 43, 3 (2011), 21:1–21:44.

[25] Elvis Koci, Maik Thiele, Oscar Romero, and Wolfgang Lehner. 2016. A Machine Learning Approach for Layout Inference in Spreadsheets. In *nternational Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, 77–88.

[26] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. In *Workshop at International Conference on Learning Representations (ICLR).*

[27] 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.

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

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

[30] Rethinking Spreadsheets and Performance Management. https://www.cutimes.com/2013/07/31/rethinking-spreadsheets-and-performance-management/. Accessed: 2018-04-23.

[31] Power BI | Interactive Data Visualization BI Tools. https://powerbi.microsoft.com. Accessed: 2018-04-27.

[32] Insights in Excel. https://support.office.com/en-ie/article/insights-in-excel-3223aab8-f543-4fda-85ed-76bb0295ffc4. Accessed: 2018-04-27.

[33] Apache POI - the Java API for Microsoft Documents. http://poi.apache.org/. Accessed: 2016-02-13.

[34] GoogleNews-vectors-negative300.bin.gz. https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUT-TlSS21pQmM/edit?usp=embed_facebook. Accessed: 2018-04-27.

[35] word2vec project on google code. https://code.google.com/archive/p/word2vec/. Accessed: 2018-04-27.