Detecting Table Clones and Smells in Spreadsheets

Wensheng Dou1, Shing-Chi Cheung2, Chushu Gao1, Chang Xu3, Liang Xu1, Jun Wei1
1State Key Lab of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
2Dept. of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong, China
3State Key Lab for Novel Software Technology and Dept. of Computer Sci. and Tech., Nanjing University, Nanjing, China
{wsdou, gaochushu, xuliang12, wj}@otcaix.iscas.ac.cn, scc@cse.ust.hk, changxu@nju.edu.cn

ABSTRACT

Spreadsheets are widely used by end users for various business tasks, such as data analysis and financial reporting. End users may perform similar tasks by cloning a block of cells (table) in their spreadsheets. The corresponding cells in these cloned tables are supposed to keep the same or similar computational semantics. However, when spreadsheets evolve, thus cloned tables can become inconsistent due to ad-hoc modifications, and as a result suffer from smells.

In this paper, we propose TableCheck to detect table clones and related smells due to inconsistency among them. We observe that two tables with the same header information at their corresponding cells are likely to be table clones. Inspired by existing fingerprint-based clone detection techniques, we developed a detection algorithm to detect this kind of table clones. We further detected outliers among corresponding cells as smells in the detection algorithm to detect this kind of table clones. We implemented our idea into TableCheck, and applied it to real-world spreadsheets from the EUSES corpus. Experimental results show that table clones commonly exist (21.8%), and 25.6% of the spreadsheets with table clones suffer from smells due to inconsistency among these clones. TableCheck detected table clones and their smells with a precision of 92.2% and 85.5%, respectively, while existing techniques detected no more than 35.6% true smells that TableCheck could detect.

CCS Concepts

•Applied computing→Spreadsheets •Software and its engineering→Software testing and debugging •Social and professional topics→Software maintenance.

Keywords

Spreadsheet; table clone; copy and paste; smell detection; end-user programming

1. INTRODUCTION

Spreadsheets have been widely used by end users for various business tasks, including data analysis, decision support, financial reporting, and so on. Spreadsheets are more popularly used than other computational paradigms [44]. Since spreadsheets are created and maintained by non-expert end users, errors can be easily induced into spreadsheets [36][41]. In order to improve the quality of spreadsheets, researchers have applied various software engineering techniques to spreadsheets [11], such as auditing [17], testing [2][6][26], smell detection [14][19][30] and debugging [4][43].

A spreadsheet comprises blocks of cells arranged in rows and columns. Like code clones in conventional programs [34][39][45], cells in spreadsheets can be copy-and-pasted (reused) during spreadsheet development and maintenance. Copy-and-paste can shorten the preparation time for spreadsheets. For example, a user can quickly prepare a new financial report by updating a copy based on an existing one. Commercial spreadsheet systems (e.g., Microsoft Excel) make it easy to copy cells from existing spreadsheets. When a user copies a block of cells to a new location, such spreadsheet systems can help deduce new formulas (if any) for them based on the underlying patterns of the copied cells automatically [47]. Thus, the computational semantics among the original and new cells are kept consistent.

However, after copy-and-paste, there are rarely any records or meta-data indicating which cells in a spreadsheet were thus created, although they are supposed to be uniformly modified in future. Besides, little provision is offered to warn users against arbitrary modification of these cells [31]. The current practice assumes that users should own conscientiousness to maintain the consistency between copied (original) and pasted (new) cells. Users, however, may unintentionally break the consistency when maintaining the concerned cells. This would lead to undesirable consequences, which could further manifest into errors. Caulkins’ survey [12] reported that 49% investigated users found errors in spreadsheets that were prepared by reusing existing ones. This motivates the need for effective techniques of detecting table clones (groups of copied and pasted cells) and their related inconsistency problems (if any).

In this paper, we focus on detecting table clones resulted from copy-and-paste operations and smells induced by the inconsistency among table clones. We refer to two rectangular blocks of cells as table clones when their corresponding cells share the same or similar computational semantics. For example, the corresponding cells in [C3:D7] of Figure 1(a) and [B3:C7] of Figure 1(b) share the same computation, although their contents can look different: (1) some corresponding cells (e.g., C3 in Figure 1(a) and B3 in Figure 1(b)) are both input cells, and some other corresponding cells (e.g., C7 in Figure 1(a) and B7 in Figure 1(b)) prescribe the same formula pattern although their concrete formulas take seemingly different inputs (these inputs come from corresponding cells in the two blocks); (2) all corresponding cells have the same headers (e.g., D3 in Figure 1(a) and C3 in Figure 1(b) both have the row header “Weekly” and column header “% Responses”). Our empirical study found that such table clones commonly exist, e.g., 1,214 table clone groups exist in 352 spreadsheets from the widely-used EUSES corpus [25]. Besides, 25.6% of these spreadsheets with table clones suffer from smells due to inconsistency among these table clone groups. This indicates the necessity of

This is the author’s version of the work. It is posted here for your personal use. Not for redistribution. The definitive version was published in the following publication:
FSE’16, November 13–18, 2016, Seattle, WA, USA
© 2016 ACM. 978-1-4503-4218-6/16/11...
http://dx.doi.org/10.1145/2950290.2950559

787
detecting table clones in real-life spreadsheets and their contained smells.

We propose TableCheck in this paper to detect table clones and their related smells. The key challenge is how to determine which cells can form table clones and which cells are subject to smells when they are involved in inconsistency among table clone groups. TableCheck works based on the observation that two tables (blocks of cells), if having the same row and column headers at their corresponding cells, are likely to share the same computational semantics and become table clones. However, simply examining each possible pair of cell blocks for detecting table clones can be extremely time-consuming. As inspired by existing fingerprint-based code clone detection techniques [32], we exploit a cell’s derived header information as its fingerprint for facilitating table clone detection. Furthermore, to detect smells among grouped table clones, we analyze possible inconsistency among corresponding cells in these table clones (e.g., dissimilar formulas from D3 in Figure 1(a) and C3 in Figure 1(b-c)), and mark outliers from them as smells (e.g., C3 in Figure 1(c)).

We implemented TableCheck as a prototype tool and evaluated its performance using the EUSES corpus [25] from two perspectives: table clone detection and smell detection. Experimental results show that: (1) table clones are common, and 21.8% of spreadsheets with formulas contain table clones; (2) smells associated with table clones are also common, and TableCheck detected 2,892 smelly cells in 177 table clone groups, which cover 14.6% validated-as-true table clone groups; (3) smells among table clones are harmful, and 971 smelly cells indeed contain wrong values; (4) TableCheck detected table clones and their smells with a high precision of 92.2% and 85.5%, respectively; (5) other existing smell detection techniques (e.g., AmCheck/CACheck [21][22], CUSTODES [16], Excel and UCheck/Dimension [5][13]) detected no more than 35.6% true smells TableCheck detected.

TableCheck differs from Hermans’ data clone detection work [30] in the types of detected clones. Their work considers two blocks of cells with (almost) the same values as clones. Thus it cannot detect table clones (with different values) and smells in Figure 1. On the other hand, TableCheck can detect clones with the same computational semantics but different values. TableCheck also differs from our previous work AmCheck/CACheck [21][22] and CUSTODES [16] in both cell clustering and smell detection. TableCheck leverages the header information to detect table clones in the same worksheets, different worksheets and different spreadsheets. Unlike TableCheck, AmCheck/CACheck and CUSTODES aggregate cells into clusters by their formula similarity (e.g., referenced cells and continuous cell locations) in a single worksheet. As such, AmCheck/CACheck and CUSTODES cluster cells [D3:D6] in Figure 1(a) and [C3:C6] in Figure 1(c) into different clusters. Unlike TableCheck, which detects smells by cross-checking inconsistency among table clones (across clusters), AmCheck/CACheck and CUSTODES detect smells inside single clusters. As such, the smell detection of AmCheck/CACheck and CUSTODES are mostly restricted to those cells that can be clustered by their formula similarity (e.g., they treat [C3:C6] in Figure 1(c) as a cluster, but there is no smell in it). Therefore, TableCheck and AmCheck/CACheck/CUSTODES are working at two different levels. Unlike TableCheck, which detects smells among cells, UCheck [5] and dimension inference [13] exploit the header information to detect type inconsistency in single formulas. As a summary, no prior technique has been dedicated for detecting table clones and their contained smells in spreadsheets.

We summarize our main contributions in this paper as follows:

- We proposed the notion of a new and commonly existing structure in spreadsheets, table clone, in which corresponding cells share the same or similar computational semantics.
- We proposed a fingerprint-based algorithm to detect table clones in spreadsheets by exploiting the header information derived from cells, and an outlier-based technique to detect smells in grouped table clones. Both are automated.
- We implemented our work into a TableCheck tool and evaluated it with real-life spreadsheets from the EUSES corpus. The experimental results show that TableCheck could detect table clones and smells effectively and precisely.

The remainder of this paper is organized as follows. Section 2 presents a motivating example. Section 3 explains the concept of table clone. Section 4 presents our table clone and smell detection approach. Section 5 presents our TableCheck implementation, and Section 6 evaluates it experimentally. Sections 7 and 8 discuss limitations and related work, and Section 9 concludes this paper.

2. MOTIVATION

In this section, we illustrate table clones and related smells using an illustrative example extracted from the EUSES corpus [25], and then explain how to detect them.
2.1 Example

Figure 1 shows four worksheet excerpts from the EUSES corpus. These excerpts perform quantitative analysis on four questions, and form a table clone group (marked by four rectangles).

Smells among table clones. A cell suffers from a smell when there are inconsistent computations in the corresponding cells among table clones. The above example contains 9 smelly cells (marked with red right-cornered triangles), among which the value of cell C3 in Figure 1(d) is likely to be wrong. Note that smelly cells do not necessarily contain wrong values. For instance, smelly cells [C3:C6] in Figure 1(c) likely contain correct values. Improper modifications of cell formulas in table clones are a major cause for these smells. For example, users may prepare the excerpt in Figure 1(c) based on the one in Figure 1(b). Cells [C3:C6] in Figure 1(c) could be a consequence of improper cell modifications that replace SBS7 with its computed value 30. Even though the modifications do not alter the computed values, they impose an assumption that the computed value of SBS7 is always 30 in view of spreadsheet updates. The four smelly cells can become erroneous if the assumption is violated by later updates.

In the example, we observe two different types of smells that can induce computation anomalies in cells, which are also studied by our previous work AmCheck/CACheck [21][22] and CUSTODES [16]. (1) Missing formula smell: The smell occurs when a cell is supposed to contain a formula, but it does not. For example, cells [C3:C6] in Figure 1(d) exhibit such smells. However, due to no formula in Figure 1(d), formula-based smell detection (e.g., AmCheck/CACheck and CUSTODES) cannot detect these smelly cells [C3:C6]. (2) Inconsistent formula smells: The smell occurs when a cell contains an incorrect formula. For example, cells [C3:C6] in Figure 1(e) exhibit such smells (e.g., the formula in C3 should be B3/SBS7). However, since all formulas in cells [C3:C6] in Figure 1(e) are the same, formula-based smell detection (e.g., AmCheck/CACheck and CUSTODES) treats these cells [C3:C6] as correct ones. Note that none of the smells shown in Figure 1 can be detected by other existing tools, such as Excel 2013 and UCheck/Dimension [5][13].

2.2 TableCheck Overview

Detecting table clones and their associated smells needs to address three technical challenges. Let us explain them using the example in Figure 1. First, how can one judge whether a cell (e.g., D3 in Figure 1(a)) belongs to a table clone as well as the boundary of the concerned table clone? Second, in which situation do the occurrences of dissimilar formulas in table clones suggest smells? Note that the occurrence of two dissimilar formulas does not necessarily suggest that the corresponding cells are smelly. For example, cells D3 in Figure 1(a) and C3 in Figure 1(b) do not share the same formula (in the R1C1 format; explained later in Section 3.1), but they are not smelly. Third, suppose that a group of cells fall into the situation where smells must have occurred. How can one locate the smelly cells in the group? For example, the group formed by D3 in Figure 1(a) and C3 in Figure 1(b–d) falls into a smelly situation, but only C3 in Figure 1(c–d) are smelly.

For the first challenge, we observe that table clones usually have the same header information. Inspired by existing fingerprint-based clone detection techniques [32], TableCheck uses cells’ header information as fingerprints, and groups cells with the same header information as table clones. For the second challenge, we observe that the formulas in table clones should reference their input cells similarly using relative indices. Thus, we convert all formulas into their relative R1C1 formats (e.g., the formulas in D3 in Figure 1(a) and C3 in Figure 1(b) would be converted into the same formula RC[-1]/R[4]C[-1] in Figure 2). For the third challenge, cells dissimilar to most of their corresponding ones can be identified as outliers. Such outliers are a good indicator of smells.

3. PRELIMINARIES

In this section, we introduce the terminologies used in this paper.

3.1 Spreadsheet Programming Model

A spreadsheet can be modeled as a set of cells with expressions, which are indexed by two-dimensional cell addresses (e.g., C3) [6]. A cell may contain a formula, which describes the computation of this cell, or contains a plain value.

A formula references another cell by a cell reference that denotes the referenced cell’s address. Most spreadsheet systems have two built-in formats to represent a cell reference: A1 and R1C1 formats [46]. These two formats can be either absolute or relative. An absolute reference keeps pointing to the same cell even if it is copied to another cell. A relative reference encodes the cell address offset between the current cell and the referenced cell, and the offset keeps unchanged when it is copied to another cell. In the A1 format, a cell at the x-th column and y-th row is notated as xy in relative reference (e.g., C3), or $x$y in absolute reference (e.g., SC3). In the R1C1 format, a cell at m rows below and n columns right to the current cell is notated as R[m]C[n] (in relative reference), and a cell at the m-th row and n-th column is notated as R[mn] (in absolute reference).

Let R be a set of cell references, EXP be a set of expressions, and V be a set of plain values. A cell’s expression exp is either a plain value ($v \in V$), a cell reference ($r \in R$), or a function $f$ over one or more expressions: $exp = v \mid r \mid f(exp_1, ..., exp_n)$. Functions in a formula include basic operators, such as,”+”, “−”, “∗”, “/”, and other built-in functions like SUM and MAX. Note that formulas in the corresponding cells among table clones often have the same expressions in the R1C1 format.

3.2 Table Clone

We focus on numerical cells in this paper. A table is a rectangular block of numerical cells (e.g., cells [C3:D7] in Figure 1(a) and [B3:C7] in Figure 1(b)). Note that we also consider empty cells (e.g., D7 in Figure 1(a)) and cells with some special strings (e.g., “NA” and “-“; discussed more in Section 4.1.1) as parts of a table, because they are often used as default values in spreadsheets.
s1. Classify cells.  
s2. Infer cell headers.  
s3. Create lookup table.  
s4. Find clones.  
s5. Detect smells.

Figure 3. Overview of TableCheck’s workflow. For ease of presentation, we give only the excerpts in Figure 1(a) and Figure 1(b).

Users often use headers to denote the meanings of cells. For example, C3 in Figure 1(a) has a row header “Weekly” and a column header “Responses”, and these headers show that C3’s value represents the number of weekly responses. We therefore look for tables of cells labelled by the same set of row/column headers. Note that headers are not treated as parts of a table in this paper.

**Definition 1:** A table clone (pair) is a tuple \((t_1, t_2)\) where \(t_1\) and \(t_2\) are two tables, in which corresponding cells are labelled by the same headers.

We made three observations on table clones in EUSES. (1) It can, of course, happen by chance that two irrelevant cells are labelled by the same headers. However, two blocks of cells in the spreadsheet are more likely to be a table clone than not if their cells at the same relative row/column have the same headers. (2) Formula cells that have the same headers in a table clone usually follow the same/similar computation. They often contain the same or similar formulas in the R1C1 format. (3) If the tables in a clone group contain only one row/column, it may not expose interesting computational semantics. Thus, we require that tables in a clone group have at least two rows and columns.

**Definition 2:** A table clone group is a group of tables in which every two tables are a clone pair.

We assume that the corresponding cells in a table clone group share the same or similar computational semantics. If no cell in a table clone group has a formula, these cells are all data ones, and do not contain any clear computational semantics. Therefore, we require that table clone groups should contain at least one formula cell.

4. APPROACH

Given a spreadsheet, TableCheck analyzes it and reports all detected table clones, as well as related smells, if any, with explanations. TableCheck works in five steps as outlined in Figure 3. First, it determines each cell’s type (data, formula, label or empty) in the spreadsheet (Section 4.1.1). Second, it extracts headers for the cells with types of data, formula and empty (Section 4.1.2). Third, it builds a lookup table for all cells with complete header information. Fourth, it detects and extracts table clones from these cells (Section 4.1.3). Finally, it analyzes corresponding cells in detected table clones, and detects smells, if any, inside them (Section 4.2).

4.1 Table Clone Detection

TableCheck extracts headers for all (data, formula and empty) cells in a given spreadsheet, and then detects table clones based on the fingerprints that are built on the cells’ headers.

4.1.1 Cell Classification

To distinguish different cell types, we follow the approaches described by Hermans [28] and Abraham and Erwig [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 have strings and explain the meanings of other cells, and (4) empty cells.

Our cell classification algorithm works as follows. First, all numerical cells without formulas are marked as data cells, and all numerical cells with formulas are marked as formula cells. Note that, although strings could be results of formulas, they are typically used as labels. Therefore, we do not treat their concerned cells as formula cells. Second, we observe that some common strings in spreadsheets are usually used as data, not labels. (1) Some special strings, e.g., “na”, “n/a” and “n.a.”, represent “not applicable”, usually implying data cells in spreadsheets. (2) Some special strings are usually used as default values (e.g., 0), such as “,” “ ” and “.”. Treating such strings as labels would definitely affect the precision of the header inference (Section 4.1.2). Therefore, we consider the cells with such special strings as data cells. All cells, which do not get a type from the above steps, are classified as label cells when not empty, and empty cells, otherwise.

Note that the above classification algorithm differs slightly from previous approaches [1][28][30]. First, all numerical cells without formulas are considered as data cells. Thus, data cells are not necessarily referenced by formulas (e.g., cells [B3:C7] in Figure 1(d)), which is mandatory in work [28]. Second, special strings (e.g., “-” and “NA”) could mislead us to derive wrong headers if not treated specially (i.e., we treat them as data rather than labels).

Applying the above algorithm to the examples in Figure 1(a) and Figure 1(b) results in the colored excerpts in Figure 3(s1). For the excerpt Q1 in Figure 3(s1), cells [C3:C6] are marked as data cells (in yellow), and cells [D3:D6] and C7 are marked as formula cells (in blue). The remaining gray cells are marked as label cells.

4.1.2 Cell Header Inference

According to Definition 1 in Section 3.2, tables can contain data, formula and empty cells. Thus, we infer the headers for all non-label (i.e., data, formula or empty) cells in a spreadsheet.

Each non-label cell is subject to two headers: a row header and a column header. For a non-label cell \(c\), we identify the nearest (to the left in the same row as cell \(c\) lies) label cell as its row header. Similarly, we consider the nearest (above in the same column as cell \(c\) lies) label cell as its column header.
Our algorithm for inferring the row header for a given cell \( c \) works as follows. It starts from cell \( c \)'s neighboring cell to the left in the same row as cell \( c \) lies. If the examined cell is a label cell, its value is set as the row header of cell \( c \). If the cell is a formula, data or empty cell, the algorithm skips it and continues to examine its left cell. The algorithm repeats until either a label cell is found or the boundary of the current worksheet is reached. For the latter case, the algorithm ends without assigning a row header to cell \( c \). The algorithm for inferring the column header for cell \( c \) is similar. Once the row and column headers are found, they together form the headers for cell \( c \).

Note that the two headers of a cell generally denote the intended computation of this cell. The row headers of cells in a column usually differ. Similarly do the column headers of cells in a row. The spreadsheet excerpt in Figure 4 shows an interesting case. According to our previous algorithm, the column headers of cells C11, D11 and E11 should be “£000”. However, the string “£000” does not denote the intended computation of these cells. We should identify “Rec”, “Non-rec” and “Total” as these cells’ column headers. Thus, we assume that if a string (e.g., “£000” in Figure 4) occurs frequently in a row/column (more than \( x\% \) of all label cells in the row/column; we set \( x \) to be 50 in TableCheck), the concerned label cells in the row/column should not be considered as column/row headers for other cells.

Referring to the example in Figure 4, the inference of the column header for C12 starts from C11, which is a data cell. Next, C10 and C9 are inspected one after another, which are empty cells. Next, C8 is inspected. According to our rule above, the label cells in Row 8 should not be used as column headers. Thus, C7 is next inspected, where the right column header “Rec” is found. Similarly, we identify “Other funding” in B12 as C12’s row header.

Our inferred headers are comparable to the first-level headers in UCheck [1]. For table clones, their first-level headers are often reused, but their higher-level headers are usually changeable. For example, in Figure 1(a), the label cell C1 (“Q1”) is treated as a higher-level header for other data and formula cells. Similarly, in Figure 1(b), the label cell B1 (“Q2”) is treated as a higher-level header for other data and formula cells, too. These two higher-level headers are not the same. Thus, treating them as headers for other cells would make us miss table clones in Figure 1(a-b).

We note that the tabulation styles of cells in a spreadsheet often vary. Thus, the above header inference is heuristic, and it may not always find right headers for cells. Nevertheless, our experimental evaluation in Section 6.2 reported that our inference algorithm could already extract cell headers precisely, and generated only few false positives in detecting table clones in the EUSES corpus.

### 4.1.3 Table Clone Detection Algorithm

We then explain how TableCheck detects table clones, in which each pair of corresponding cells should have the same header information. Our TableCheck’s clone detection approach was inspired by Johnson’s fingerprint-based code clone detection technique [32]. Still, we need to significantly extend it to handle two-dimensional spreadsheet programs.

TableCheck creates a lookup table for all cells with both row and column headers. In the lookup table, each cell’s row and column headers are used as its key, and a list of cell addresses as the corresponding value (one-to-many mapping). For example, in Figure 3(s3), the key “Weekly:Responses” occurs in Q1!C3 and Q2!B3. Note that TableCheck creates fingerprints only for cells with complete (both row and column) header information. Users may create table clones with no header or only incomplete headers (e.g., only row or column header). However, given two cells without any header or with only incomplete headers, it is hard to tell whether they are clones. Therefore, TableCheck focus only on cells with complete header information.

TableCheck’s clone detection algorithm (Algorithm 1) takes the lookup table and all cells (allCells) with complete header information as inputs, and returns table clone groups. The algorithm works as follows. (1) It finds a seed cell that is contained in allCells, and treats this seed cell as the initial table curTable (Lines 3-4). (2) It inspects all neighboring cells of curTable in each direction (left, right, up and down). After curTable includes all cells in one of the four directions, it may find some tables (getClones method; Lines 22-38) in which all corresponding cells share the same header information. Then the algorithm expands curTable to include its neighboring cells in that direction (Lines 8-10). (3) It repeats Step (2) until it cannot extend curTable in all directions (Lines 5-13). (4) Now, curTable and its clones form a new table clone group if they contain at least one formula cell, two rows and columns. Then all cells in this group are removed from allCells (Lines 14-18). (5) The algorithm repeats the above

---

**Algorithm 1. Table clone detection algorithm.**

```plaintext
Input: lookup (lookup table), allCells (all cells with complete header information).
Output: groups (all detected table clone groups).
1. groups = EMPTY;
2. while allCells not EMPTY do
3.   seedCell = allCells.removeOne(); // Retrieve one cell.
4.   curTable = (seedCell);
5.   foreach direction in (LEFT, RIGHT, UP, DOWN) do
6.     if newTable contains all cells in curTable and
7.     // curTable’s neighboring cells in the direction.
8.     newTable = generateTable(curTable, direction);
9.     if (getClones(newTable) not EMPTY) then
10.    curTable = newTable;
11.   goto Line 5;
12.   end if
13. end for
14. group = (curTable, getClones(curTable));
15. if (group contains formulas, >1 row and >1 column)
16.   groups.add(group);
17. end if
18. allCells.removeCells(group); // Delete cells in group.
19. end while
20. return groups;
23. method getClones(curTable);
24. clones = EMPTY;
25. firstCell = table.firstCell(); // Upper-left cell.
26. cloneSeeds = lookup.get(firstCell.header);
27. foreach cloneSeed in cloneSeeds do
28.   if cloneTable has the same size as table, and
29.   // cloneSeed as cloneLabel’s upper-left cell.
30.   cloneTable = buildClone(cloneSeed, table);
31.   if (isClone(table, cloneTable) then
32.     clones.addCloneTable();
33.   end if
34. end for
35. return clones;
36. end method
```

---

Figure 4. Cell header inference example, extracted from the EUSES corpus. The column headers of C11, D11 and E11 should be “Rec”, “Non-rec” and “Total”, respectively.
four steps until allCells becomes empty (Lines 2-19). The get
Clones method (Lines 22–38) computes clones according to a
given table. It uses the upper-left cell of table to find potential
cloned upper-left cells by inspecting the lookup table (Lines 25-
26). Starting from each potential clone’s upper-left cell, it builds a
clonetable that has the same size as table, and then checks
whether clonetable has the same header information with table. If
yes, clonetable is treated as a clone of table (Lines 28-35).

Note that for each table clone group thus detected, there does not
exist another table clone group that contains all cells in this table
cloned group. Consider our example in Figure 3. First, Algorithm 1
could select cell Q1!C3 as the initial table curTable = Q1!!C3:C3).
Subsequently, curTable’s right neighbor Q1!D3 is inspected. The
algorithm now finds a table Q2!B3:C3 that has the same header
information as table Q1!!C3:D3). Thus, curTable is extended as
Q1!!C3:D3). Now, curTable cannot be extended to left or right,
because there is no other cell to the left or right of curTable. Then
the algorithm inspects the neighboring cells (Q1!C4 and Q1!D4)
below curTable. It can still find a table Q2!B3:C4 that has the
same header information as table Q1!!C3:D4). Thus, curTable is
extended as Q1!!C3:D4). Repeating the above process, the algo-

**4.2 Smell Detection among Table Clones**

We then explain how to detect smells in extracted table clones.

**4.2.1 Detection Strategy**

Since table clones are used to express similar business tasks, the
corresponding cells among them usually share the same or similar
computational semantics. Inconsistent computational semantics in
corresponding cells usually indicate that some of them are smelly.

As smelly cells in a spreadsheet normally occur in minority, they

**Missing Formula Smell.** If some (not all) cells in CellGroup have
formulas, it indicates that the cells in CellGroup should be com-
puted by formulas. Thus, all data cells in CellGroup are consid-
ered smelly (i.e., suffering from missing formula smells). For
example, in CellGroup2 = {Q1!D3, Q2!C3, Q3!C3, Q4!C3}, the
first three cells are formula cells, and the last cell Q4!C3 is a data
cell. Thus, TableCheck considers Q4!C3 smelly. Similar smells
occur to cells [C4:C6] and B7 in Figure 1(d). For this type of
smell, TableCheck provides users with formula cells in CellGroup
for reference.

**Inconsistent Formula Smell.** If there is more than one different
formula in the R1C1 format in CellGroup, it indicates that some
cells in CellGroup are not computed in the same way. Thus, some
formulas in CellGroup are considered smelly (i.e., suffering
from inconsistent formula smells). However, it is challenging to
determine which formula cells in CellGroup are smelly. Ta-
bleCheck detects this type of smell by the following strategy. (1)
It counts the number of cells that use the same formula (in the
R1C1 format) in CellGroup. (2) The cells with the formula that
occurs most frequently are considered correct, and the remaining
ones are considered smelly. (3) If there is more than one formula
identified in the previous step, TableCheck conservatively consid-
ers all formula cells smelly. Take CellGroup2 = {Q1!D3, Q2!C3,
Q3!C3, Q4!C3} for example. The first two cells share the same
formula RC[-1]/RC[-1] in (the R1C1 format, discussed later as
Case 2 in Section 4.2.2), whereas the third cell Q3!C3 has a for-

**4.2.2 Suppressing False Positives**

Since cloning tables is usually adopted for new requirements by
users, inconsistency may not necessarily indicate smell. Thus, our
basic smell detection strategy could introduce false positives. We
observed three common cases that need to be taken care of before
applying the above detection strategy.

**Case 1.** A table contains a formula cell whose computation refer-
ces some cells outside the table. Due to the lack of header
information for thus referenced cells, one has no idea about their
meanings, whether they represent the same cells as those in other
table clones, and whether the formula can be reused by other table
clones. TableCheck chooses to remove from CellGroup those

**Case 2.** Some formula cells in CellGroup may use absolute refer-
ces, and these absolute references can make them have different
formulas in the R1C1 format. For example, in CellGroup2 =
{Q1!D3, Q2!C3, Q3!C3, Q4!C3}, the first two cells Q1!D3 and
Q2!C3 have different formulas in the R1C1 format (RC[-1]/RC[-1]
and RC[-1]/RC[-1], respectively). However, the referenced cells
e.g., SC7 in Q1!D3 and SB7 in Q2!C3) have the same relative
positions to the formula cells (e.g., Q1!D3 and Q2!C3, respecti-

**Case 3.** Some formula cells in CellGroup use constants (e.g., 30
in cell C3 of Figure 1(c)). Formula cells that use different con-

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Responses</th>
<th>% Responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 5.** Cells [B3:B6] compute total responses from Q1-Q4
in Figure 1(a-d), and they reference cells out of table [B3:C7].
Table 1. Statistics of spreadsheets in the EUSES corpus (n.a.: not applicable).

<table>
<thead>
<tr>
<th>Category</th>
<th>Spreadsheets</th>
<th>Spreadsheets with table clones</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SS Formula</td>
<td>Clone True True/Formula</td>
</tr>
<tr>
<td>cs101</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>database</td>
<td>678</td>
<td>200</td>
</tr>
<tr>
<td>filthy</td>
<td>45</td>
<td>4</td>
</tr>
<tr>
<td>financial</td>
<td>720</td>
<td>358</td>
</tr>
<tr>
<td>forms3</td>
<td>26</td>
<td>18</td>
</tr>
<tr>
<td>grades</td>
<td>588</td>
<td>282</td>
</tr>
<tr>
<td>homework</td>
<td>576</td>
<td>277</td>
</tr>
<tr>
<td>inventory</td>
<td>699</td>
<td>278</td>
</tr>
<tr>
<td>jackson</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>modeling</td>
<td>679</td>
<td>190</td>
</tr>
<tr>
<td>personal</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>4,037</strong></td>
<td><strong>1,617</strong></td>
</tr>
</tbody>
</table>

TableCheck can detect table clones and their related smells among spreadsheets. Since almost all spreadsheets in the EUSES corpus are irrelevant with each other, table clones detected among spreadsheets could be prone to false positives. Thus, we focused only on table clones within each spreadsheet.

We manually validated all detected table clones and smells by checking: 1) whether all corresponding cells in detected table clones share the same headers, 2) whether the corresponding cells in detected table clones share the same or similar computations, and 3) whether detected smelly cells can be repaired by their referred cells as reported by TableCheck. A smell is considered real if it can be repaired by inspecting its referred cells.

6.2 Table Clone Detection Results

We first investigate how common table clones occur in the EUSES corpus (Section 6.2.1), and then investigate the precision of our table clone detection (Section 6.2.2).

6.2.1 Statistics of Spreadsheets with Table Clones

Table 1 lists our statistics of spreadsheets that contain table clones (Spreadsheets with table clones). Table 1 gives the number of spreadsheets in which TableCheck detected table clones (Clone) and number of spreadsheets in which table clones were confirmed (True). As shown in Table 1 (True/Formula), 21.8% of our studied spreadsheets contain confirmed table clones. As reported by Hermans’ data clone detection [30], they detected data clones only in 86 spreadsheets in the same corpus. Thus, the spreadsheets containing table clones are much more than those containing data clones (352 vs. 86).
We also investigate what kinds of table clones a spreadsheet can contain. Table 1 lists the numbers of spreadsheets containing confirmed intra-worksheet table clones (Intra) and inter-worksheet table clones (Inter). Note that a spreadsheet can contain both intra- and inter-worksheet table clones simultaneously. As shown in Table 1, intra-worksheet table clones occur more often than inter-worksheet ones (271 vs. 150).

Table 1 also lists the number of spreadsheets in which table clones contain smells (Smelly). We observe that 25.6% of spreadsheets with confirmed table clones contain smells (Smelly/True). This proportion (25.6%) is non-negligible, and discloses that smells related to table clones are also common in real-life spreadsheets.

Therefore, we draw the following conclusion:

**Table clones commonly occur in real-life spreadsheets with formulas, with intra-worksheet clones occurring more often than inter-worksheet ones. Smells related to table clones are also common, and thus deserve detection.**

### 6.2.2 Results and Analyses

**TableCheck** detects table clones based on cells’ header inference that may not be fully precise. We are thus interested in the precision of our table clone detection. Since we observed that in all detected table clone groups, none of them has both true and false clone pairs, we thus measured the precision of table clone detection based on table clone groups.

Table 2 gives our table clone detection results on the EUSES corpus. In total, TableCheck detected 1,317 table clone groups (Table clone groups/Detected). We confirmed that 1,214 table clone groups are true (Table clone groups/True). The precision for table clone detection is thus 92.2%. This also discloses that our header inference works well for our table clone detection algorithm.

We also investigate what kinds of table clones a table clone group can contain. If two tables in a table clone group are located in the same worksheet, this table clone group is considered as an **intra-worksheet group**. Otherwise, they are located in two different worksheets, and the clone group is considered an **inter-worksheet group**. Note that a table clone group can be both intra- and inter-worksheet if some of its contained tables satisfy both conditions. Table 2 lists the number of confirmed intra- and inter-worksheet clone groups (Intra and Inter, respectively). We can observe that a high proportion (42.6% = 517/1,214) of table clone groups contain clones among worksheets. This discloses that detecting smells only within worksheets (e.g., AmCheck/CACheck) would miss smells that need to consider cells from different worksheets.

Table 2 also lists the number of table clone groups that contain smells (Smelly table clone groups/ Detected). We confirmed that 92.2% of these smelly table clone groups are true. The precision is also promising. In total, 14.6% (177/1,214) of confirmed table clone groups contain smells. This also indicates that smells associated with table clones are common.

**False positives of table clones.** The differences in value for the second (Detected) and third (True) columns in Table 2 give the number of false positives, which arise because extracted table clones do not have the same or similar computational semantics. Out of these 103 (1,317 – 1,214) false positives, 100 are caused by wrongly inferred headers (in Section 4.1.2). For the remaining three cases, although they share the same header information, their contained cells do not have the same or similar computational semantics. That said, in 99.8% (1,214 / (1,214 + 3)) of detected table clone groups that share the same header information, their corresponding cells have the same or similar computational semantics. This validates our observation in Section 3.2. A more precise header inference process could help improve TableCheck’s precision. We leave it to our future work.

Therefore, we draw the following conclusion:

**TableCheck detects table clones precisely. The precision for table clone and smelly table clone detection happens to be both 92.2%.**

### 6.3 Smell Detection Results

Table 3 shows TableCheck’s smell detection results. In total, TableCheck detected 3,382 (Smelly cells/Detected) from the 177 smelly table clone groups. Among these detected smelly cells, 2,892 (85.5%) were confirmed as true positives. Note that each smelly table clone may contain a different number of smelly cells.

We further investigate missing formula smells and inconsistent formula smells out of these 3,382 smells. Table 3 shows the numbers of missing formula smells (Missing formula smells) and inconsistent formula smells (Inconsistent formula smells) that were detected (Detected) and confirmed (True). Out of 3,382 detected smells, 2,383 (70.5%) are missing formula smells, and the remaining 999 (29.5%) are inconsistent formula smells. We observe that the smell detection precision of these two types of smells are comparable (83.6% vs. 90.2%), and do not differ significantly.

**Harmfulness of smells.** We manually inspected the confirmed 2,892 smelly cells, and used their referred corresponding cells to try to recover their formulas. Then, we validated whether these smelly cells’ values can be computed by the recovered formulas. We observed that 971 (33.6%) smelly cells contain wrong values. Thus, TableCheck’s detected smells are indeed harmful.

**False positives of smells.** The differences in value for the second (Detected) and third (True) columns in Table 3 give the number of
false positives of smells. We observed three reasons why these 490 (3,382 – 2,892) false positives occurred: (1) Some cells in a table clone indeed have different computations, e.g., in a financial report, an earning computation may differ in different years. It is difficult to distinguish such different computations without understanding the meanings of all related cells in spreadsheets. 404 (82.4%) false positives belong to this case. (2) Our outlier-based smell detection technique relies on heuristics, and it may wrongly treat correct cells as smelly. For example, if only two corresponding cells have different formulas, TableCheck has no idea about which one should be smelly. Thus, TableCheck conservatively considers both of them smelly. 47 (9.6%) false positives belong to this case. For the remaining 39 (8.0%) false positives, they are caused by wrongly detected table clones.

Therefore, we draw the following conclusion:

TableCheck can precisely (85.5%) detects smells associated with table clones, which are harmful. Missing formula smells occur more often than inconsistent formula smells.

### 6.4 Comparison with Existing Techniques

To better evaluate the effectiveness of our TableCheck in smell detection, we compared TableCheck with AmCheck/CACheck [21][22], CUSTODES [16], Excel 2013’s built-in error checking mechanism, and UCheck/Dimension [5][13]. For these techniques/tools, we investigate whether TableCheck’s 2,892 validated-as-smelly cells could also be detected by them.

#### 6.4.1 Comparison with AmCheck/CACheck and CUSTODES

AmCheck/CACheck [21][22] aggregates a row or column of contiguous cells into a cell array if: (1) none of these cells is empty or contains any label, (2) these cells share the same/similar input dependence and (3) at least one cell contains a formula. For example, AmCheck/CACheck considers [C3:C6] in Figure 1(c) as a well-formed cell array, and no smell can be detected in it. CUSTODES [16] uses strong features (e.g., same or similar R1C1 formulas) and weak features (e.g., same labels and font colors) to classify cells into different clusters. It then identifies smelly cells in each cluster. CUSTODES detects a cell cluster mainly by the equivalence of formulas contained by its cells. For example, CUSTODES considers cells [C3:C6] in Figure 1(c) as a correct cell cluster, and cannot detect any cell cluster in Figure 1(d). Thus, CUSTODES cannot detect smells in them.

We ran AmCheck/CACheck and CUSTODES on the EUSES corpus, and checked how many TableCheck’s validated-as-smelly cells could be detected by them. In Table 3, the AmCheck, CACheck and CUSTODES columns give the comparison results. We observe that AmCheck, CACheck and CUSTODES detected only 15.4% (444/2,892), 20.7% (599/2,892) and 35.6% (1,029/2,892) validated-as-smelly cells, respectively. Thus, AmCheck/CACheck and CUSTODES missed a lot that could be detected by TableCheck. Note that AmCheck/CACheck and CUSTODES could also detect smelly cells that are not related to table clones. Thus, TableCheck is orthogonal to AmCheck/CACheck and CUSTODES in spreadsheet smell detection.

#### 6.4.2 Comparison with Excel

Microsoft Excel has its built-in support for detecting inconsistent formula smells among cells. However, this smell detection is applicable when: (1) a cell’s formula is syntactically inconsistent with those of its two adjacent cells, and (2) the formulas of the two adjacent cells are syntactically consistent. Besides, Excel also supports detecting some well-known smells like division by zero. We used Excel 2013 for the comparison, and enabled all its checking rules. In Table 3, the Excel column shows that Excel 2013 issued warnings for only 0.4% (12/2,892) validated-as-smelly cells detected by TableCheck.

#### 6.4.3 Comparison with UCheck/Dimension

UCheck [5] and Dimension[13] also exploit the header information from spreadsheets to check type inconsistency in formulas. UCheck uses unit to represent the type of a cell, and defines some rules to detect type inconsistency. Dimension detects smells by inspecting whether units of measurement are used incorrectly. UCheck and Dimension are only applicable to formula cells. As such, smelly cells without formulas cannot be detected. We ran UCheck/Dimension tool (obtained from its authors) on the EUSES corpus. In Table 3, the UCheck column shows that UCheck/Dimension detected only 0.3% (9/2,892) validated-as-smelly cells by checking type inconsistency in formulas. Note that if a type inconsistency does not relate to any table clone, our TableCheck will not detect it. Therefore, TableCheck is orthogonal to UCheck/Dimension in smell detection.

Therefore, we draw the following conclusion:

TableCheck can detect smells that existing techniques fail to detect. Existing techniques detected at most 35.6% smells that TableCheck detected.

### 7. DISCUSSION

While our evaluation shows that TableCheck is promising for detecting table clones and their related smells in EUSES spreadsheets, we discuss potential threats and issues below.
Table clone and smelly cell validation. Since the EUSES corpus was collected in the wild without any explanation for table clones and smells, we were unable to validate our results by their original users. Thus, we manually validated our detected table clones and smells. To alleviate possible mistakes, two authors of this paper cross-checked all detected results.

Recall study of TableCheck. Since we do not have the ground truth for all table clones in the EUSES corpus, we did not measure the recall of TableCheck. It is challenging and almost infeasible to find all table clones and smells by manually inspecting and understanding each cell’s meaning of all EUSES spreadsheets. Besides, this could also cause extra speculation and bias. Similar cases also happened to the recall evaluation for code clone detection in conventional programs [9][32][38]. In the future, we plan to conduct a recall study in a company where we can find the concerned users for their spreadsheets to fairly evaluate TableCheck’s recall.

7.2 Limitations

Our TableCheck assumes the existence of meaningful row/column headers in tables. If such headers are missing, incomplete or mixed with other label cells in spreadsheets, TableCheck may not detect table clones and smells effectively. This limitation is common to existing header-based techniques [5][14][28]. Further, after cloning a table, users may modify the header information or add/remove rows/columns according to new requirements. Then TableCheck could fail to detect such table clones. However, we argue that TableCheck is a valuable step towards table clone detection for spreadsheets, which is a hard problem in general, since table styles can vary significantly among spreadsheets.

Table cloning can help save editing efforts when repeating similar tasks. However, we still do not know how and why smells among table clones occurred. A real-life spreadsheet corpus with detailed version information will definitely be helpful for understanding such causes. In the future, we plan to conduct a user study on this.

8. RELATED WORK

In the section, we discuss the pieces of work that concern the code clone detection in conventional programs and the maintenance of spreadsheet quality (e.g., error, detection, testing and evolution).

Clone detection. Code clone detection in conventional programs is a well-studied problem. Bruntink et al. [10] classified these techniques into four categories. (1) Text-based techniques [24][32]: little or no transformation to the source code is performed before attempting to detect identical or similar lines of code. (2) Token-based techniques [7][33][38]: they apply a lexical analysis (tokenization) to the source code and then use the tokens as a basis for clone detection. (3) AST-based techniques [8]: they use parsers to obtain a syntactical representation (e.g., an abstract syntax tree (AST)) of the source code, and then search for similar subtrees in the AST. (4) PDG-based techniques [37]: they use program dependence graphs (PDGs), and then look for similar subgraphs in PDGs. These code clone detection techniques cannot easily be applied to spreadsheets, which adopt a different programming model with variables indexed by rows and columns.

Our table clone detection algorithm was inspired by a text-based clone detection technique [32]. The most related work is data clone detection [30], which detects clones with the same values in spreadsheets. To the best of our knowledge, no prior technique for detecting table clones in spreadsheets has ever been proposed.

Spreadsheet error and detection. Spreadsheet errors are common [40][41][42]. Various techniques have been proposed to detect errors in spreadsheets. UCheck/Dimension[5][13] use a type system to check type inconsistency in formulas. Smellsheet Detective [19][20] detects statistical smells, functional dependence smells and so on. Hermans et al. proposed visualizing spreadsheets by dataflow graphs [28], and detected inter-worksheet smells in them [29]. These techniques mostly focus on syntactic errors, while our TableCheck focuses on inconsistency among table clones that can concern semantic errors, which are more challenging to detect. Our previous work AmCheck/CACheck [21][22] and CUSTODES [16] detect smells in a group of similar cells, while TableCheck detects smells from a group of table clones. Thus, our TableCheck is orthogonal to these pieces of existing work.

Spreadsheet modeling and testing. Building rigorous models for spreadsheets [3][18][27] can help users reduce the chance of introducing errors. However, it is challenging to construct such models. Testing spreadsheets [6][26][35] can be an alternative, but its major challenge comes from the lack of test oracles. TableCheck extracts table clones and detects inconsistency among them. TableCheck does not require explicit test oracles.

Spreadsheet evolution. Table clones are often created during spreadsheet evolution when cells are copied for different purposes. SheetDiff [15] identifies changes between two spreadsheets as a whole and does not detect table clones and smells directly. VEnron [23] publishes the first versioned spreadsheet corpus to facilitate spreadsheet evolution analysis. TableCheck can be used on VEnron to detect smells across multiple versions of spreadsheets.

9. CONCLUSION

In this paper, we study table clones in spreadsheets. These clones occur when copying spreadsheet cells during spreadsheet development. We have also proposed an automated approach, TableCheck, to extracting table clones by grouping cells with the same header information. Further, we propose an outlier-based technique to detect smells among table clones by cross-checking the corresponding cells among table clones. Our empirical study shows that table clones and their related smells are common in real-life spreadsheets. Our experimental evaluation shows that our table clone and smell detection approach is both effective and precise, while existing techniques can only detect at most 35.6% of validated-as-true smells detected by our TableCheck.

We plan to pursue our future work in three ways. First, TableCheck can be improved by more precise spreadsheet header inference. Second, TableCheck can also be improved by handling table clones with missing or incomplete header information. Third, more case studies on real-life spreadsheets would be conducted for evaluating TableCheck’s practical effectiveness in companies.

10. ACKNOWLEDGMENTS

This work was supported in part by Beijing Natural Science Foundation (4164104), National Key Research and Development Plan (2016YFB1000803), the Hong Kong SAR RGC/GRF grant (611811), National Natural Science Foundation (Grant Nos. 61472174, 91318301, 61321491) of China, and the Collaborative Innovation Center of Novel Software Technology and Industrialization, Jiangsu, China.
11. REFERENCES


[49] TableCheck project (including experimental subjects and results). http://www.tcse.cn/~wsdou/project/clone/ .