CUSTODES: Automatic Spreadsheet Cell Clustering and Smell Detection using Strong and Weak Features

Shing-Chi Cheung[§], Wanjun Chen[§], Yepang Liu[§], and Chang Xu[‡]

[§]Dept. of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong, China

[‡]State Key Lab for Novel Software Technology and Dept. of Computer Sci. and Tech., Nanjing University, Nanjing, China

[§]{scc*, wchenah, andrewust}@cse.ust.hk, [‡]changxu@nju.edu.cn*

ABSTRACT

Various techniques have been proposed to detect smells in spreadsheets, which are susceptible to errors. These techniques typically detect spreadsheet smells through a mechanism based on a fixed set of patterns or metric thresholds. Unlike conventional programs, tabulation styles vary greatly across spreadsheets. Smell detection based on fixed patterns or metric thresholds, which are insensitive to the varying tabulation styles, can miss many smells in one spreadsheet while reporting many spurious smells in another. In this paper, we propose CUSTODES to effectively cluster spreadsheet cells and detect smells in these clusters. The clustering mechanism can automatically adapt to the tabulation styles of each spreadsheet using strong and weak features. These strong and weak features capture the invariant and variant parts of tabulation styles, respectively. As smelly cells in a spreadsheet normally occur in minority, they can be mechanically detected as clusters' outliers in feature spaces. We implemented and applied CUSTODES to 70 spreadsheets files randomly sampled from the EUSES corpus. These spreadsheets contain 1,610 formula cell clusters. Experimental results confirmed that CUSTODES is effective. It successfully detected harmful smells that can induce computation anomalies in spreadsheets with an F-measure of 0.72, outperforming state-of-the-art techniques.

CCS Concepts

• Software and its engineering→Software creation and management→Software verification and validation→Software defect analysis→Software testing and debugging • Applied computing→Computers in other domains→Personal computers and PC applications→Spreadsheets.

Keywords

Spreadsheets; cell clustering; smell detection; feature modeling; end-user programming.

1. INTRODUCTION

Spreadsheets are a popular computation paradigm used for data storage and analysis, decision support, financial reporting, and quality control [46]. Despite their popularity, spreadsheets are found to be error-prone [13, 25, 37, 42, 45]. Conventional quality assurance measures for software such as unit tests and assert statements are generally inapplicable to spreadsheets.

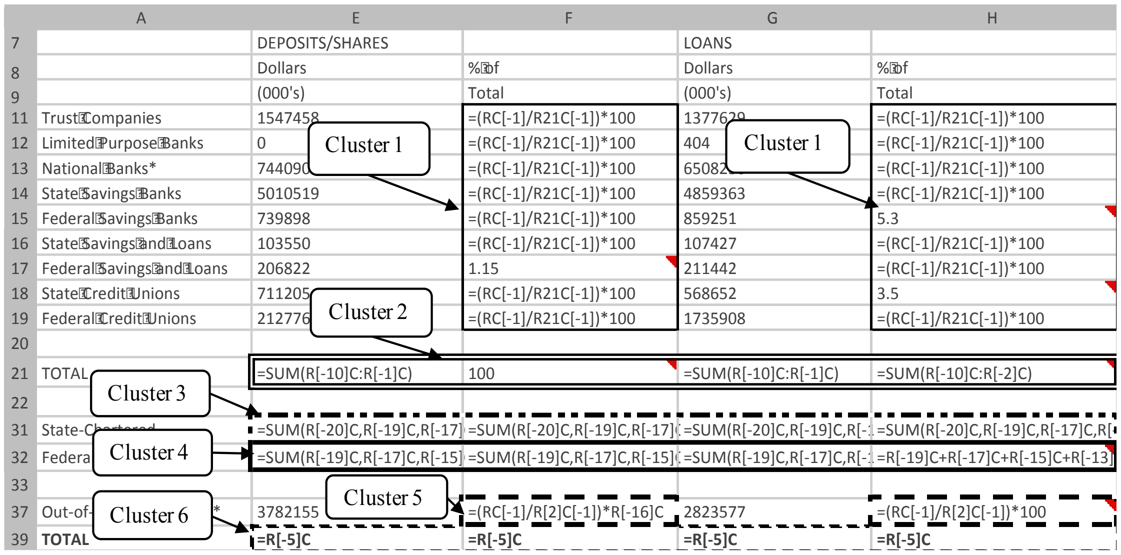
To improve spreadsheet quality, researchers have proposed various quality assurance techniques [34], among which smell detection draws much attention. The concept of spreadsheet smell is inspired by code smell, which was originally introduced by Fowler [23]. In conventional programs, code smells refer to those bad designs that are not necessarily errors themselves, but likely degenerate into faults in subsequent program development and maintenance. In the spreadsheet domain, researchers study smells mainly from two perspectives. One is to apply the conventional principles of code smells to spreadsheets [28]. Formula cells with complexity exceeding certain pre-defined metric thresholds are detected as smells in the sense that they hinder spreadsheet understanding. The other is to consider spreadsheet-specific characteristics such as statistical properties of data [16] and computation semantics of formulas [19, 32]. From the latter perspective, spreadsheet smells usually refer to the discrepancies between a group of similar cells that are usually neighbors. There is so far no systematic way to characterize smells in spreadsheets and address the problem uniformly. Users need to apply various techniques individually under different settings and validate their results one by one. Moreover, smells detected as violations of pre-defined patterns or metric thresholds fail to adapt to diverse tabulation styles, which reflect different user formula design practices and application domains. As a result, the smell detection can miss many smells in one spreadsheet while reporting many spurious smells in another.

We made two observations from the existing literature on spreadsheet smells. One is that the studied smells are mostly related to formulas. These smells occur at formula operations, cell references or missing formulas [16, 19, 28, 32]. Second, the identification (or clustering) of neighboring cells plays an important role in smell detection so that these cells can be checked against some pre-defined pattern or threshold as a group. For example, Hermans et al. [28] identified the smells that occur when one formula differs slightly from its "neighbors", and referred to them as "Rare Formulas". Cunha et al. [16] catalogued smells like "pattern finder" that identifies the deviation of one cell from the pattern in a row. Dou et al. [19] identified "ambiguous computational smells" in a cell array (a contiguous range of formula cells in a row or a column) if it contains cells evaluated using different computations.

While the first observation motivates us to study formula-related smell detection, the second observation suggests two existing problems in smell detection. First, clustering based on neighboring cells can be inaccurate because neighboring cells can sometimes be spatially unobvious. An alternative is to locate neighboring cells based on formula expressions [2, 15, 22, 47]. However, this alternative is

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
ICSE '16, May 14-22, 2016, Austin, TX, USA
© 2016 ACM. ISBN 978-1-4503-3900-1/16/05…$15.00
DOI: http://dx.doi.org/10.1145/2884781.2884796

*Shing-Chi Cheung and Chang Xu are the corresponding authors.



	A	E	F	G	H
7		DEPOSITS/SHARES		LOANS	
8		Dollars	% of	Dollars	% of
9		(000's)	Total	(000's)	Total
11	Trust Companies	1547458	=(RC[-1]/R21C[-1])*100	1377629	=(RC[-1]/R21C[-1])*100
12	Limited Purpose Banks	0	=(RC[-1]/R21C[-1])*100	404	=(RC[-1]/R21C[-1])*100
13	National Banks*	744090	=(RC[-1]/R21C[-1])*100	65082	=(RC[-1]/R21C[-1])*100
14	State Savings Banks	5010519	=(RC[-1]/R21C[-1])*100	4859363	=(RC[-1]/R21C[-1])*100
15	Federal Savings Banks	739898	=(RC[-1]/R21C[-1])*100	859251	5.3
16	State Savings and Loans	103550	=(RC[-1]/R21C[-1])*100	107427	=(RC[-1]/R21C[-1])*100
17	Federal Savings and Loans	206822	1.15	211442	=(RC[-1]/R21C[-1])*100
18	State Credit Unions	711205	=(RC[-1]/R21C[-1])*100	568652	3.5
19	Federal Credit Unions	212776	=(RC[-1]/R21C[-1])*100	1735908	=(RC[-1]/R21C[-1])*100
20					
21	TOTAL	=SUM(R[-10]C:R[-1]C)	100	=SUM(R[-10]C:R[-1]C)	=SUM(R[-10]C:R[-2]C)
22					
31	State-Chartered	=SUM(R[-20]C,R[-19]C,R[-17]	=SUM(R[-20]C,R[-19]C,R[-17]	=SUM(R[-20]C,R[-19]C,R[-1	=SUM(R[-20]C,R[-19]C,R[-17]C,R[
32	Federa	=SUM(R[-19]C,R[-17]C,R[-15]	=SUM(R[-19]C,R[-17]C,R[-15]	=SUM(R[-19]C,R[-17]C,R[-1	=R[-19]C+R[-17]C+R[-15]C+R[-13]
33					
37	Out-of- *	3782155	=(RC[-1]/R[2]C[-1])*R[-16]C	2823577	=(RC[-1]/R[2]C[-1])*100
39	**TOTAL**	**=R[-5]C**	**=R[-5]C**	**=R[-5]C**	**=R[-5]C**

Figure 1. A motivating example of an extracted EUSES [21] spreadsheet in Excel R1C1 format

not effective on smell detection because cells suffering from formula-related smells often have their formulas deviated from others or even completely missing. Second, they do not consider that users often have their own styles in tabulating spreadsheets. Such styles can also vary across application domains. The use of a pre-defined pattern or metric threshold to cluster neighboring cells for any spreadsheet can incorrectly identify cell clusters, and therefore miss the detection of true smells or detect spurious smells.

Motivated by these two observations, we propose in this paper a formula-related smell detection approach called CUSTODES (clustering using strong and weak features) that can infer and adapt to users' different tabulation styles. To achieve this, we study how to effectively cluster similar formula cells using features based on unsupervised learning. The effect of each feature on clustering can vary across tables and spreadsheets, depending on tabulation styles. We also study the characterization of smelly cells as outliers of their clusters and detect them in feature spaces.

CUSTODES addresses three major challenges. The first challenge is how to determine the set of features to construct a feature model for automatic cell clustering. Spreadsheets are often prepared and maintained by users who likely have their own styles in tabulating data and formulas. These styles may not even be consistent in all cells in a cluster when parts of them are copied or adapted from another spreadsheet. For example, some users put a "Total" header label to indicate the summation computation; hence summation formula cells related to this header should reside in the same cluster. While in other situations, users may put a summation formula cell at the end of a row or column without labeling it as "Total". Automatically clustering cells based on computation semantics is difficult without prior knowledge of users' design and styles. The feature model should not only reflect general spreadsheet system characteristics, but also capture varying spreadsheet tabulation styles. The second challenge is how to extract those features. Comparing with conventional software, spreadsheets involve simpler computation [34]. Spreadsheets provide limited coding information from which relationships among cells can be inferred. The third challenge is to determine and quantify the deviation among cells for effective smell detection as outliers. The deviation should be so quantified that a cell suffering from more types of smells would be a clearer outlier.

To address the three challenges, CUSTODES adopts a two-stage clustering technique, which is inspired by Yoshida et al. [49], based on a spreadsheet's strong and weak features. *Strong features* model the invariant parts of tabulation styles. They are properties that can be generically used for clustering. Examples are cell formulas and cell reference relations (e.g., those cells in [F11:F16] in Figure 1). These features can often be used to cluster cells together. However, relying on strong features alone for cell clustering is inadequate and can result in many non-clustered cells because not all cells in a cluster necessarily exhibit these strong features. For example, a smelly cell F17 in Figure 1 misses the strong feature described by the formula expression as in cells [F11:F19], and hence cannot be clustered using formulas alone. To solve this problem, we also consider *weak features*, which model the variant parts of tabulation styles. Examples are cell labels, layouts, spatial relations, fonts and the number of significant figures. They may sometimes be used to cluster cells together (e.g., the header "Total" in F9 for cells [F11:F19]). The idea is to use the clusters obtained at the first stage to extract weak features for the second stage of clustering. These weak features can be specific to tables and spreadsheets. For example, not all tables and spreadsheets use "Total" to label cells in a cluster. CUSTODES have two unique advantages. First, it does not have to assume cells in a cluster appearing in the same row, same column or even contiguously in a spreadsheet. Second, it can automatically adapt to varying tabulation styles.

Cells dissimilar to most of their peers can be identified as outliers of their clusters. Such dissimilarity is a good indicator of smells or errors. Specifically, CUSTODES identifies outlier cells as smells and ranks them based on their outlier scores. We further classify various kinds of outliers into different smell types by examining the feature spaces. Explanation of the smell types exhibited by each outlier cell in terms of the corresponding feature space can be provided for users' further actions.

We implemented CUSTODES as a tool and evaluated its performance using the EUSES corpus [21] from two perspectives: cell clustering and smell detection. Experimental results show that our two-stage clustering approach can improve existing clustering techniques [39] by over 12% in recall while retaining a high precision. As smells typically occur in minority, such an improvement in recall is critical to effective smell detection. As we will show in Section 6.3, our tool successfully clustered many smelly cells that could not be clustered by existing approaches. The successful clustering of these smelly cells allows them to be detected as outliers. Our tool implemented the outlier detection of four popular types of smells reported by recent studies: missing formulas, dissimilar references, dissimilar operations and formulas with hard-coded constants [45]. The evaluation results show that CUSTODES can detect smells effectively, and outperform the state-of-the-art smell detection techniques. Smells are detected with a precision of 0.65, recall of 0.80, F-measure of 0.72 as compared with 0.57, 0.62, 0.60 achieved by our previous work AmCheck [19].

CUSTODES differs from our previous work AmCheck [19] in both cell clustering and smell detection. Unlike the current work using a feature model for clustering, AmCheck aggregates a row or column of contiguous cells into a cluster only if: (1) none of the cells is empty or contains labels, and (2) the row and column indices of their formula operands must be within the range of these contiguous cells. As such, AmCheck does not cluster the cells in Clusters 1 and 5 in Figure 1. Unlike the current work identifying smells as outliers in a feature space, AmCheck detects smells by checking whether a set of constraints extracted from the formulas in a cluster is solvable. As such, the smell detection of AmCheck is mostly restricted to formulas that can be handled by linear constraint solvers. In contrast, the current work detects smells as outliers in feature spaces. It is able to detect smelly cells that involve complex formula expressions such as those containing conditions and strings. No prior technique has been proposed to cluster spreadsheet cells using strong and weak features, and formulate the problem of spreadsheet smell detection as outlier detection.

In summary, this paper makes the following four contributions:

- We proposed a feature model that is able to capture both the invariant and variant parts of tabulation styles. The former models those common practices for spreadsheet tabulation. The latter models domain knowledge and user styles specific to individual spreadsheets.
- We adapted a recent algorithm [43] proposed in the information retrieval community to automatically cluster cells in two stages. Each cell cluster is associated with a feature model. Smells are then automatically identified and ranked as outliers.
- We implemented CUSTODES as a tool that can automatically cluster cells and detect smells for spreadsheets.
- We evaluated CUSTODES on 70 spreadsheet files, which contain 1,610 clusters of formula cells, randomly sampled from the EUSES corpus.

The remainder of this paper is organized as follows. We present a motivating example in Section 2, followed by an overview of CUSTODES in Section 3. Section 4 introduces our two-stage clustering and outlier-based smell detection approach. Section 5 presents the implementation of CUSTODES. After reporting our experimental results in Section 6, we discuss our findings and related work in Sections 7 and 8, followed by a review of threats to validity in Section 9. The paper is concluded in Section 10.

2. MOTIVATING EXAMPLE

In this section, we motivate our problem using an illustrative example extracted from the EUSES corpus [21] in Figure 1. Due to page limit, the example gives an excerpt of the actual spreadsheet, which summarizes the assets, deposits/shares and loans of MAINE financial institutions. The example contains seven smelly cells, among which the values of four data cells are likely errors (the four numeric data cells marked with a red right-cornered triangle "◥"). For example, the value of cell F17 is inconsistent with that computed by formula (RC[-1]/R21C[-1])*100, and the values of cells [F11:F19] do not sum up to 100 as indicated in cell F21. Errors in some cells such as F17 can induce further errors in other cells such as F21. Note that none of the smells shown in Figure 1 can be detected by the error checking tool of Excel 2013. In the example, we observe four different types of harmful smells that can induce computation anomalies.

Missing formula smell: It is a type of smells studied by AmCheck [19]. The smell occurs when a cell is supposed to contain a formula, but it does not. AmCheck detects such smells by checking whether a row/column of contiguous cells, referred to as a *cell array*, contains both data and formula cells. If yes, AmCheck would consider those data cells smelly. However, AmCheck fails to detect three missing formula smells in the example because they cannot be clustered using cell arrays. For instance, the cells in [F11:F19] should have the same formula, but AmCheck fails to detect the missing formula smell in F17 because [F11:F19] is not considered a cell array as its operand R21C[-1] is not a cell between rows 11 and 19. This illustrates that correct identification of cell clusters is important to smell detection. It also illustrates the difficulties in characterizing a cluster using fixed patterns and spatial relations.

Dissimilar reference smell: The smell occurs when a formula references an incorrect cell or cell range. An example is cell H21. Unlike its peers in Cluster 2, the formula of cell H21 misses the reference of cell H20. This smell can degenerate into errors in subsequent modification and maintenance, as for example, if row 20 is no longer empty or the SUM in H21 is replaced by another function that is sensitive to empty cells.

Dissimilar operation smell: It occurs when the cells in a cluster contain formulas with dissimilar operations. An example is cell H32, which uses the "+" operator to evaluate summation while its peers in the same row use a "SUM" function. Refactoring the concerned formula using function SUM, which is also used in H32's peers such as cell F32, could repair the smell in H32. However, identifying the peers of H32 as well as making refactoring suggestions are non-trivial.

Hard-coded constant smell: This type of smells occurs when a cell replaces cell referencing with a number while other peer cells do not. For example, unlike cell F37, cell H37 replaces the reference R[-16]C with a constant 100 in formula. Although the current result is correct since R[-16]C in H37 references cell H21 whose value happens to be 100, this may induce errors when H21 is no longer evaluated to 100 in future after spreadsheet maintenance**.**

The above four smell types can be found in real-life spreadsheets [19, 41, 45]. A case study on how they could be induced is discussed in our earlier study [19]. As illustrated above, smells of different types are characterized in existing work using different criteria, which require the use of different mechanisms for their detection. This makes it a hurdle for users to adopt automatic smell detection for spreadsheets, especially when they have no idea which mechanism would work best for their spreadsheets. It is because users generally do not have prior knowledge on the types of smells by which their spreadsheets are mostly infected. Another issue is that users adopt different styles in tabulating spreadsheets. This motivates us to study whether smells can be uniformly characterized and detected.

3. METHODOLOGY OVERVIEW

Figure 2 gives an overview of our methodology, which consists of four steps: preprocessing, first-stage clustering, second-stage clustering by bootstrapping, and smell detection. The preprocessing

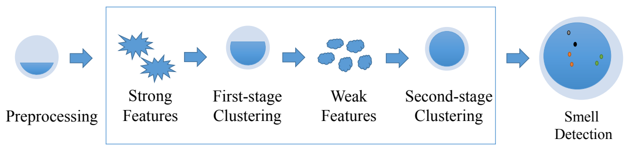


Figure 2. Methodology overview

step identifies formulas in given spreadsheets. From these formulas, we extract two strong features: abstract syntax trees and cell dependency trees. Cell dependency can be extracted using the "trace precedents" built-in function of Excel. These two strong features can be used to relate cells belonging to the same cluster in each spreadsheet. The first-stage clustering is an unsupervised learning process based on the similarities calculated using the two strong features. The clusters such constructed are referred to as *seed clusters*. The second-stage clustering extracts a set of weak features common to each seed cluster and its member cells. These weak features can be used to associate additional cells with the cluster. Note that each cell may only be associated with one cluster. The last step is smell detection. An intra-cluster outlier score is calculated using outlier detection technique for each cell within a cluster by exploring the feature space in the worksheet concerned (a worksheet is a page in a spreadsheet). The report of this step is a list of cells ranked by their outlier scores. Highly ranked cells are likely smells because they most likely miss features common to their clusters. These cells are more error-prone than others, prompting users' attention. Lowly ranked cells are unlikely smells.

Let us illustrate the process using Cluster 1 of the previous example in Figure 1. After the first-stage clustering, cells [F11:F16], [F18:F19], [H11:H14], [H16:H17] and H19 are aggregated to form one seed cluster since they share one R1C1 formula as well as the same structure of cell dependency. In the second stage, we are able to extract four weak features that are closely associated with the seed cluster: (1) they (cells) all have the same "Total" header labels; (2) they are all either in column F or column H; (3) they form two repeating cell blocks with horizontal one cell gap; and (4) they are in the same table. Using the extracted weak features, cells F17, H15 and H18 are further aggregated to the seed cluster since they share the same weak features. These three cells are also detected as outliers in the feature space of formulas. Unlike most of their peers in the seed cluster, these outliers miss all formula related features. This illustration demonstrates that cell clustering based on formula alone is inadequate. It misses the three smelly cells.

4. APPROACH

As mentioned in Section 1, CUSTODES needs to tackle three challenges. In this section, we introduce the features that can be leveraged for cell clustering to address the first challenge, and explain how they can be extracted to address the second challenge. After that, we present our two-stage clustering algorithm, followed by a smell detection mechanism to address the third challenge.

4.1 Preliminaries

The following introduces the terminologies used in this paper.

Cell classification: We follow Hermans et al.'s approach [29] to classify cells into: (1) *label cells*: string cells that explain the meaning of other cells, (2) *data cells*: value cells without embedding formulas, (3) *formula cells*: cells that embed a formula, and (4) *empty cells*. *Formula clusters* are those cell clusters that contain at least one formula cell. Since formula-related smells are common in spreadsheets [19, 30], we focus on deriving formula clusters and detecting smells in these clusters. Note that a formula cluster may contain data or empty cells. We do not cluster label cells, and treat labels as a candidate weak feature of formula clusters.

Cell address: A cell address is the coordinate (x, y) of a given cell, where x and y represent its row and column indices, respectively.

Cell reference: A cell reference is a reference in a formula to another cell. Cell references can be absolute or relative. An absolute reference in the form of $\$x\$y$ refers to a cell at the address (x, y). It does not change even if the reference is copied to another cell formula within the same worksheet. A relative reference encodes the cell address offset between the current cell and referenced cell. The offset of a relative reference remains unchanged when it is copied to another cell formula.

Cell formula: Let F be a set of cell formulas, A be a set of cell references, and V be a set of values. A formula $(f \in F)$ expresses a value $(v \in V)$, a cell reference $(a \in A)$, or an operation φ over one or more formulas: $f ::= v \mid a \mid \varphi(f, \ldots, f)$. Operations in a formula include basic operators, such as +, −, *, /, as well as Excel's built-in functions like SUM and AVERAGE. Formulas are given in R1C1 format, which is commonly adopted to represent the formula of a cell in Excel [42, 48]. Formula cells in the same cluster often have the same expression in R1C1 format. The formula in cells [F11:F19] is "=(RC[-1]/R21C[-1])*100", where "RC[-1]" refers to the cell in the same row but one column left to the current cell.

4.2 First-stage Clustering

In the first stage, we extract two strong clustering features from each formula cell: formula expression and cell dependency. These features are generic to spreadsheets.

4.2.1 Two Strong Clustering Features

Formula's abstract syntax tree: Formulas are a common feature to relate formula cells of the same cluster [2, 15, 22]. This feature is extracted by parsing each formula (including its cell references, values and operations) into an abstract syntax tree (AST), which represents the computation semantics of the associated formula.

Cell dependency: Formula cells in the same cluster are typically subject to the same pattern (or tree structure) of cell dependency. We adapt the spreadsheet program model proposed by Abrahm and Erwig [4] to model cell dependency. The cell dependency of a cell c is a tree rooted by c. The tree is so constructed that a formula cell is the parent node of those cells this formula cell references. Leaf nodes of the tree are data or empty cells.

4.2.2 Cell Similarity Function

Since both strong features are modeled as tree structures, we cluster formula cells with respect to strong features based on tree similarity. Tree similarity is measured by Tree Edit Distance using the RTED algorithm [44], which calculates an editing script of a tree structure with add, update and delete actions. Given two formula cells f_1 and f_2, their tree similarity is defined as follows:

$$sim(f_1, f_2) = 1.0 - \frac{RTED(T_1, T_2)}{|T_1| + |T_2|},$$

where $|T|$ is the node number in tree T.

4.2.3 Clustering Algorithm

For each pair of formula cells, we obtain two similarity scores *ASTsim* and *CRTsim* based on their tree similarity with respect to abstract syntax tree and cell dependency, respectively. The merged similarity score $sim(f_1, f_2)$ of two formula cells f_1 and f_2 is calculated as a multiplication of the two scores:

$$sim(f_1, f_2) = ASTsim(f_1, f_2) * CRTsim\,(f_1, f_2).$$

We use the standard hierarchical agglomerative clustering (HAC) [26] algorithm to cluster formula cells based on the merged similarity. Initially, each formula cell forms a cluster on its own. This cluster then iteratively merges clusters with the closest distance pairs. The average distance metric is defined as below in order to decide inter-cluster distances:

$$distance(Cl_i, Cl_j) = \frac{\sum_{f_1 \in Cl_i} \sum_{f_2 \in Cl_j} dist(f_1, f_2)}{|Cl_i||Cl_j|},$$

$$dist(f_1, f_2) = 1.0 - sim(f_1, f_2).$$

The parameter of HAC [26] is the similarity threshold value. In order to guarantee high accuracy in the first stage clustering, the threshold is set to strong similarity. Two cell clusters in a worksheet are considered separate when their distance exceeds 0.02. The output of this stage is a set of clusters that contain more than one cell. We refer to them as seed clusters, and the cells in these clusters as seed cells. The input of the algorithm can be a worksheet, a spreadsheet or even a family of spreadsheets with similar filenames.

4.3 Second-stage Clustering

This subsection explains weak features and our second-stage bootstrapping clustering algorithm. Weak features are collected from the existing literature on spreadsheets to relate similar cells. While these features (e.g., layout and labels) can relate similar cells, they vary across spreadsheets. We present in the following seven weak features currently implemented by our tool. The set of weak features can be readily extended and supported by CUSTODES for second-stage clustering.

4.3.1 Weak Clustering Features

Cell address: A cell is uniquely addressed by its row and column indices (x, y). It is common for users to add new cells in a cluster along the same row or column. As such, cells in the same row or column likely belong to the same cluster. Typically, a cluster groups cells either horizontally or vertically. Both extensions of clusters can also appear at the same time in a spreadsheet. Our clustering algorithm assumes no prior-knowledge about whether the cells in a spreadsheet are clustered along row or column.

Label: Labels in a spreadsheet can be horizontal or vertical. Horizontal and vertical labels describe cells in a row and a column, respectively. A label (e.g., "TOTAL") is the string content of a label cell. It is sometimes used to describe the computation performed by a cell cluster [1]. As such, labels can be a useful clustering feature if we can automatically relate each cell to a label. To do that, we follow an approach proposed by Abraham and Erwig [1] to relate each non-label cell to its nearest horizontal label cell (on the left in the same row) and its nearest vertical label cell (above in the same column). For example, in Figure 1, the label cell A21 "TOTAL" is assigned to cells [E21:H21] as their horizontal label, indicating a summation of values. Not all non-label cells have horizontal and vertical labels. We ignore meaningless label cells formed by strings serving as visual separators such as "=" and "–".

Layout: Spreadsheet users may deploy different layout styles to visually cluster related cells in the same format [33]. Examples of layout styles are font, font size, font color, borders and even background color. CUSTODES treats font color as well as background color as a candidate weak feature of a cell. This feature varies according to users' practices and preferences, and tends to be consistent within the same spreadsheet.

Alliance: A cell is an alliance of another cell if they are the parameters/operands of the same function/operation like SUM, MAX or AVG. For example, when a formula cell is evaluated by an operation "=SUM(R[-10]C:R[-2]C)", all cells in the range [R[-10]C:R[-2]C] referenced by the formula cell are alliances. Such alliance relation is referred to as *physical area*, which can be used to describe a conceptual cohesion among cells [9].

Table: A spreadsheet table serves as an area where data and formula cells are clustered [2]. An example is the rectangular area comprising cells [E11:H19] in Figure 1. However, users may not always separate unrelated data using tables or organize tables consistently. As such, the containment of a cell by a table is only a weak feature for clustering. We follow the approach adopted by Abraham and Erwig [2] to extract tables in a given spreadsheet.

Cell array membership: A cell array clusters a set of contiguous, non-empty cells in a column or row that are subject to the same computation semantics [19]. A row or column of contiguous cells are clustered into a cell array if: (1) none of the cells is empty or contains labels, and (2) the row and column indices of their formula operands must be within the range of these contiguous cells. A cell array must contain at least one formula cell. A cell may at most be clustered into one cell array. There are two kinds of cell arrays: row-based and column-based, depending on whether clustered cells in an array share the same row or column. For example, the cells in [E21:H21] in Figure 1 are members of the same cell array.

Gap template: Templates can be generalized from two or more cell blocks formed by consecutive rows or columns of cells that capture similar types of data and computation. They share the same visual layout, and some of their formula cells in the same relative position share the same formula. An example is the two cell blocks [F11:F19] and [H11:H19] in Figure 1. Cells F13 and H13 have the same relative position in their respective cell blocks, and they share the same formula. After a template based on similar cell blocks is derived, peer cells (e.g., F15 and H15) in the same relative position in their own blocks could be associated and clustered. The addresses of these peer cells differ by a horizontal or vertical offset. The derivation of templates as well as the corresponding offsets can be implemented by referencing the spreadsheet template inference mechanism proposed by Abraham and Erwig [2]. We refer to these offsets as *gap templates*. For example, the gap template based on [F11:F19] and [H11:H19] models a horizontal offset of two. Note that it is possible to derive multiple gap templates from a set of spreadsheet cells. A gap template can also be derived from similar cell blocks across worksheets and even spreadsheets.

4.3.2 Bootstrapping Algorithm

The bootstrapping algorithm (Figure 3) used in CUSTODES is adapted from an information retrieval technique, referred to as "Espresso", for computational linguistics [43]. The algorithm was later applied by Yoshida et al. [49] to reinforce a two-stage clustering that disambiguates person names in Web search results. Given a set of seed instances, Espresso learns their patterns and thereby extracts additional instances over the Web, iteratively. New instances and patterns are selected and added according to a reliability function defined over self-mutual-information values. We adapt Espresso by mapping instances and patterns to cells and weak features, respectively. Under such mappings, the algorithm is adapted to harvest weak features from the (seed) cells in each seed cluster derived in the first stage. Remaining (originally non-seed) cells are then successively added to seed clusters based on their extracted weak features. Algorithm 1 gives the bootstrapping algorithm. A matrix multiplication is used to propagate weak features from seed clusters to non-seed cells. Like the first-stage clustering, we apply the bootstrapping algorithm for each worksheet so that a cell can only be grouped (or harvested) to a seed cluster in the same worksheet.

Algorithm 1. Bootstrapping algorithm

```
Input: Feature cell matrix $\boldsymbol{P}$, cell cluster matrix $\boldsymbol{R}_{\boldsymbol{CE}}^{(\mathbf{0})}$, threshold
Output: Cell clusters $\boldsymbol{Clu}_{\boldsymbol{cell}}$
 1: $\boldsymbol{R}_{\boldsymbol{FT}}^{(\mathbf{0})} = \frac{1}{|\boldsymbol{CE}|}\boldsymbol{P}\boldsymbol{R}_{\boldsymbol{CE}}^{(\mathbf{0})}$   // Calculate feature-cluster matrix
 2: $\boldsymbol{R}_{\boldsymbol{FT}} = \begin{cases} \frac{1}{\max \boldsymbol{Ochiai}} \boldsymbol{log}\frac{p(ft,\ ce)}{p(ft)p(ce)} & \boldsymbol{if}\ \frac{p(ft,\ ce)}{p(ft)p(ce)} > 1 \\ 0 & \boldsymbol{Otherwise} \end{cases}$,
    $(\textbf{for each } \boldsymbol{ft} \in \boldsymbol{FT}, \boldsymbol{cell} \in \boldsymbol{CELL})$
 3: $\boldsymbol{R}_{\boldsymbol{CE}} = \frac{1}{|\boldsymbol{FT}|}\boldsymbol{P}^{\boldsymbol{T}}\boldsymbol{R}_{\boldsymbol{FT}}$   // Calculate cell-cluster matrix
 4:
 5: method Harvesting ($\boldsymbol{R}_{\boldsymbol{CE}}$)
 6:   for each $\boldsymbol{Clu} \in \boldsymbol{CLU}$ do
 7:     $\boldsymbol{Clu}_{\boldsymbol{NScell}} = \textbf{arg} \quad \boldsymbol{max}_{\boldsymbol{clu}}\boldsymbol{r}_{\boldsymbol{NScell},\boldsymbol{Clu'}}$, where
        $\{\boldsymbol{Clu'} \mid (\boldsymbol{Clu'} \in \boldsymbol{CLU} \wedge |\boldsymbol{Clu}| > 1) \vee \boldsymbol{CLU}_{(\boldsymbol{CELL})}^{(\mathbf{0})}\}$
 8:     if $\textbf{Comp}(\boldsymbol{Clu}_{\boldsymbol{NScell}}, \boldsymbol{Clu})$ > threshold
 9:       $\boldsymbol{Clu}_{\boldsymbol{cell}} = \boldsymbol{add}(\boldsymbol{NScell},\ \boldsymbol{Cu})$ //Add non-seed cell to cluster
10:     end if
11:   end for
12:
13: method Comp ($\boldsymbol{S}_{\boldsymbol{o}},\ \boldsymbol{Clu}$)
14:   for each $\boldsymbol{Scell} \in \boldsymbol{CE}$ do
15:     $\boldsymbol{S}_{\boldsymbol{min}} = \textbf{arg} \quad \boldsymbol{min}_{\boldsymbol{Scell}}\boldsymbol{r}_{\boldsymbol{Scell},\boldsymbol{Clu}}$
16:   end for
17: return $\frac{\boldsymbol{S}_{\boldsymbol{o}} - \boldsymbol{S}_{\boldsymbol{min}}}{\boldsymbol{S}_{\boldsymbol{min}}}$
```

Figure 3. Algorithm for the second-stage cell clustering

4.3.2.1 Input matrix construction

Let $\boldsymbol{FT}$ denote the weak feature vector, $\boldsymbol{CE}$ denote the cell vector (containing both seed and non-seed cells), $\boldsymbol{CL}$ denote the seed cluster vector. Feature-cell matrix $P \coloneqq |\boldsymbol{FT}| \times |\boldsymbol{CE}|$, where $p_{i,j} = 1$ when ft_i is contained in ce_j. Cell-Cluster matrix $\boldsymbol{R}_{\boldsymbol{CE}} \coloneqq |\boldsymbol{CE}| \times |\boldsymbol{CL}|$, is a matrix to capture the association strength among cells and clusters. For the initial matrix $\boldsymbol{R}_{\boldsymbol{CE}}^{(\mathbf{0})}$, $\boldsymbol{r}_{\boldsymbol{j},\boldsymbol{k}}^{(\mathbf{0})} = 1$ if the cell ce_j is in the seed cluster cl_k after the first stage clustering. Feature-cluster matrix $\boldsymbol{R}_{\boldsymbol{FT}} \coloneqq |\boldsymbol{FT}| \times |\boldsymbol{CL}|$, is a weighting matrix of weak features and clusters.

4.3.2.2 Algorithm details

This algorithm contains two major components: (1) *matrix multiplication* to refine cell-cluster association values (Lines 1–3), and (2) *harvesting* non-seed cells to be added to seed clusters (Lines 5–11). We elaborate on the details below.

Matrix multiplication (Lines 1–3): Before performing the matrix multiplication, we need to prepare two input items $\boldsymbol{R}_{\boldsymbol{CE}}^{(\mathbf{0})}$ and $\boldsymbol{P}$. First, cells in seed clusters of a size larger than one are identified, giving the cell-cluster matrix $\boldsymbol{R}_{\boldsymbol{CE}}^{(\mathbf{0})}$. These cells, referred to as *seed cells* earlier, are formula cells that exhibit the two strong features as presented in Section 4.2.1. Second, the weak features of each seed cluster are extracted from its seed cells to form a feature-cell matrix $\boldsymbol{P}$. Except for seed cells, label cells and empty cells, data and remaining unclustered formula cells are referred to as *non-seed cells*.

Line 1 is a matrix multiplication to get an initial feature-cluster association weights $\boldsymbol{R}_{\boldsymbol{FT}}^{(\mathbf{0})}$ by multiplying feature-cell matrix $\boldsymbol{P}$ with initial cell-cluster matrix $\boldsymbol{R}_{\boldsymbol{CE}}^{(\mathbf{0})}$. However, it is possible that a feature is common to many clusters, resulting in a high weight. For example, the same font style is used in many clusters. However, this contradicts to our goal of finding weak features specific to a cluster, i.e., features most uniquely characterize a cluster. As such, the Ochiai coefficient is calculated for resulting feature-cluster matrix $\boldsymbol{R}_{\boldsymbol{FT}}$ (Line 2). Ochiai coefficient is one of the best performing association measures [5]. This step could be regarded as a re-weighting process to put higher weights on features that are strongly related to a seed cluster. At Line 3, by multiplying $\boldsymbol{P}^{\boldsymbol{T}}$ with $\boldsymbol{R}_{\boldsymbol{FT}}$, cluster feature weights then propagate back to the cell-cluster matrix and new cell-cluster association values are obtained.

Harvesting non-seed cells (Lines 5–11): In order to harvest non-seed cells to seed clusters, the algorithm finds the cluster that maximizes the association value for each non-seed cell (Line 7). If the value is above the specified threshold, this non-seed cell could be harvested (Lines 8–9). Intuitively, we want to exclude those cells with insignificant associations when compared with the seed cells in the cluster. We use a relative difference (Line 15) to compare the minimum association value of the seed cells in the same cluster with the ready-to-harvest one, and this makes the difference normalized to range [0, 1] and relieves from the need for adopting a different threshold for each cluster.

4.4 Smell Detection

Let us explain smell detection based on our identified cell clusters.

4.4.1 Intra-cluster Smelly Suspiciousness

Our two-stage clustering algorithm identifies the set of weak features that uniquely characterize each seed cluster from seed cells, and leverages these features to group cells into seed clusters. This allows cells to be grouped into a seed cluster even if their strong features (i.e., formulas) deviate from those of the seed cells inside the same cluster. Since smells likely occur in minority, we utilize an outlier detection technique for smell detection. For example, an outlier detection in the feature space of the strong features can detect computation-related smells as a cell's computation is driven by its formula.

We focus on computation-related smells in spreadsheets because they are pervasive and can adversely affect spreadsheet correctness. To facilitate outlier detection of such smells, strong features are mapped onto vectors, and the Euclidian distance in the feature space is calculated to determine the degree of deviation.

Local Outlier Factor (LOF) [12] is a density-based outlier detection technique, which assumes that the density around a normal data object (inlier) should be similar to the density around its neighbors, while the density around an outlier is substantially lower than the density around its neighbors. An LOF score is assigned to each clustered cell to represent its outlierness. The score is calculated as the ratio of the local density of the cell and the average local density of its neighbor. The local density of a cell is inversely proportional to the average distance to its *k*-nearest neighbors. A cell can be considered an outlier if its LOF score is well above 1 [12]. In order to use LOF, a parameter *k* is needed to decide the neighborhood size. Here, we follow the rule-of-thumb in the *k*-nearest neighbor algorithm and assume *k* to be the square root of the cluster size [27, 35, 36]. The output of our algorithm is a list of cells ranked in a descending order by their LOF scores, which represent their smell suspiciousness.

4.4.2 *Smell Types Categorization*

In this subsection, we present how we categorize outliers into different types of computation smells.

Missing formula smell: It occurs at a numeric data cell where a formula is expected but missing. It is a smell detected by AmCheck. The smell is known as overwriting formula errors in another study [9]. Cells suffering from this smell do not exhibit any strong feature, and therefore they have high LOF scores. In the motivating example (Figure 1), there are four data cells such as cells F17 and F21 suffering from this smell. CUSTODES clusters these smelly cells together with formula cells. This facilitates users to validate the smells and their numeric values using the associated formulas. Since clustered data cells miss the two strong formula-related features completely, they are strong suspects of missing formula smells. An optimization of smell detection is to treat all these data cells as outliers and thereby save their LOF calculation. This optimization can sometimes improve the outlier detection of dissimilar formula smells. We deployed this optimization in our experiments.

Dissimilar formula smell: A cell's formula describes its computation. A formula typically consists of three types of components: cell references, constant values and operations. Depending on which type of component is smelly, smells can occur in any of the three following forms:

(1) *Dissimilar formula references*: It occurs when the R1C1 expression of a formula is dissimilar to its peer cells' formulas in the same cluster. The smell can be characterized by an outlier that deviates mainly in the cell dependency strong feature. This kind of smells (e.g., cell H21 in Figure 1) is often found to be error-prone [41, 45].

(2) *Dissimilar formula operations*: It occurs at a formula cell whose set of operations/operators differs from that of the majority cells in the same cluster. The smell can be characterized by an outlier that deviates mainly in the formula operations feature. Its effect is less serious if the different operations have the same computational semantics such as SUM and "+", an example cell falling into this subcategory is H32.

(3) *Formula with hard-coded constants*: It occurs at a formula cell where an operand of its formula is hard-coded with a constant instead of a cell reference, while it is not the case for most formula cells in the same cluster. It is referred to as a qualitative error in [45], and especially harmful when users forget to update the constant value in spreadsheets upon modification or reuse. An example of this is cell H37 in Figure 1.

Hermans et al. [28] proposed five other types of formula smells by adapting conventional code smells in programs to spreadsheet formulas. These smells occur to cells that subscribe a relatively complex formula. Although formula complexity can degrade readability and comprehensibility, *complexity smells* tend to be less problematic because formulas may need to be such implemented for their intended computations. Due to space, we do not discuss the detection and categorization of these five types of smells in this paper. Essentially, they can also be detected as outliers in a feature space that models formula complexity such as the number of references, number of operations and calculation chains.

5. IMPLEMENTATION

This section briefly explains some implementation details. In the preprocessing part, a formula cell's dependency relationship is tracked using Excel's built-in function. We implemented it using VBA in macros. The two-stage clustering and smell detection modules were implemented in Java, and Excel files' processing was realized on top of Apache POI [8]. Our tool automatically identifies cell clusters and highlights them visually to users. Each smelly cell is additionally marked by a red right-cornered triangle "◥" together with a comment, describing its smell type and ranking.

Table 1. Statistics of our experimental subjects

Category	# worksheets	# cells	# formula cells	# clusters	# smelly cells
cs101	1	106	40	8	3
database	60	42,688	6,973	547	1,206
financial	102	54,734	5,692	533	477
forms3	5	1,774	734	35	12
grades	30	23,998	2,571	73	124
homework	23	12,137	3,878	150	50
inventory	35	17,082	1,927	125	59
modeling	35	36,508	4,901	139	43
Total	291	189,027	26,716	1,610	1,974

6. EVALUATION

Our evaluation studies the following two research questions:

RQ1. How well can our two-stage approach cluster spreadsheet cells? We compare the clustering result of CUSTODES with those of two existing cell clustering techniques: (1) the cell array technique from AmCheck [19], and (2) the logical area technique [39] whose three cell similarity criteria are commonly used in spreadsheet visualization, auditing, testing and error categorization.

RQ2. How well can our outlier-based smell detection detect spreadsheet smells? We compare the smell detection result of CUSTODES with that of AmCheck [19]. We use the precision, recall and F-measure metrics in the comparison.

6.1 Experimental Subjects

We evaluated our research questions on 291 worksheets from 70 spreadsheet files, which were randomly selected from the EUSES corpus [21]. These spreadsheets form a representative subset of EUSES corpus according to the previous work AmCheck [19]. Before our experiments, we manually inspected each worksheet to label all cell clusters. In each cell cluster, we further marked the smelly cells and labeled their smell types. This ground truth was carefully established by many rounds of cross validations by all coauthors of this paper and an additional postgraduate student. Table 1 gives the statistics of the experimental subjects. As shown in the table, the subjects are diverse (covering eight different categories) and contain more than 20 thousand cells and formulas. Our manual inspection of these files found 1,610 cell clusters and 1,974 smelly cells. We made our experimental dataset available online for future research [17].

6.2 Evaluation Metrics

The performance of cell clustering is evaluated based on four outcomes of each cell pair's assignment: (1) a pair of similar cells assigned to the same cluster (true positive or TP), (2) a pair of dissimilar cells not assigned to the same cluster (true negative or TN), (3) a pair of dissimilar cells assigned to the same cluster (false positive or FP), and (4) a pair of similar cells not assigned to the same cluster (false negative or FN). Two cells are similar if they should belong to the same cell cluster.

Similarly, smell detection can also have the four outcomes: (1) a detected smelly cell is truly smelly (TP), (2) a normal cell (non-smelly) is detected as normal (TN), (3) a detected smelly cell is normal (FP), and (4) a truly smelly cell is detected as normal (FN).

Table 2. Precision, recall and F-measure of three clustering techniques

Clustering technique	# clusters	Precision	Recall	F-measure
AmCheck	2,486	0.98	0.17	0.29
Logical Area (CE)	1,725	0.99	0.74	0.85
Logical Area (LE)	1,840	0.93	0.73	0.82
Logical Area (SE)	823	0.58	0.79	0.67
CUSTODES	1,582	0.91	0.89	0.90

To study the effectiveness of our clustering approach and compare it with existing techniques, we use the following three evaluation metrics [18, 38, 49]:

$$\text{Precision} = \frac{TP}{TP + FP} \qquad \text{Recall} = \frac{TP}{TP + FN}$$

$$\text{F-measure} = \frac{2 \times Precision \times Recall}{Precison + Recall}$$

6.3 Cell Clustering Evaluation Results

We made no assumption on the optimal threshold value of our two-stage bootstrapping algorithm (Algorithm 1), and set the threshold to default 0.5 in all our experiments.

Table 2 gives our cell clustering results. In total, CUSTODES detected 1,582 cell clusters. Its average precision, recall and F-measure are 0.91, 0.89 and 0.90, respectively. For comparison, AmCheck [19] detected 2,486 cell arrays. Its precision is 0.98, which is slightly higher than CUSTODES. However, its recall is only 0.17, meaning that it failed to cluster many similar cells. This is because AmCheck imposes strong spatial constraints on cell array members as well as their referenced cells to achieve high clustering precision. As we discussed earlier, such a heuristic-based technique may not always perform well, although the heuristic can cluster quite a few smelly cells.

We also compared CUSTODES with the logical area technique [39]. The technique adopts three similarity criteria of formula cells to find conceptually-cohesive but not necessarily spatial-related regions in spreadsheets. The three similarity criteria are: (1) copy-equivalence (CE), meaning that a pair of cells share the same R1C1 formula, (2) logical-equivalence (LE), meaning that the two formulas of a pair of cells differ only in absolute cell references or constant values, and (3) structural-equivalence (SE), meaning that the formulas of a pair of cells contain the same operations and in the same order.

Table 2 gives the experimental results of the logical area technique. As shown in the table, the performance of the three criteria differs. CE has the highest precision (0.99), while SE has the highest recall (0.79) among the three. The precisions of CE (0.99) and LE (0.93) are slightly higher than CUSTODES (0.91) because we try to achieve a tradeoff between precision and recall. For example, by using weak features in our two-stage bootstrapping algorithm, more similar cells can be grouped into the same cluster, facilitating effective smell detection.

Figure 4 compares the effect of recall on the number of true smells successfully clustered by three logical area criteria and by our CUSTODES approach. The light (grey) bars show the recall for the logical area clustering technique with three different similarity criteria, respectively. The darker (orange) bars give the recall of our two-stage clustering approach. Our recall (0.89) outperforms those of CE (0.74) and LE (0.73). Although the differences seem not large, we found that the clusters detected by using the CE and LE criteria are not very useful in smell detection as the cells in these clusters are highly consistent. Our experimental results also confirmed this. For example, CE is a criterion requiring identical formulas, and only cells with the same formula can be aggregated to a cluster. As such, no smell can be detected from these clusters. The nine smelly cells clustered by the LE criterion can also be clustered by CUSTODES. More importantly, the improvement in recall from 0.73 (LE) to 0.89 (CUSTODES) brings significant benefit of clustering nearly 1,800 more true smelly cells that could not be clustered by LE. Finally, although SE has a better recall (0.79) than CE (0.74) and LE (0.73), its low precision (0.58) indicates that many cells are incorrectly clustered. Among the clusters identified by SE, 249 true smelly cells can be detected as outliers. In other words, there are many smelly cells failing to be clustered using SE. The clusters generated by CUSTODES can be more effectively used for smell detection as compared with those generated by the three logical area techniques. Note that CUSTODES clustered 1,804 smells but only detected 1,583 of them (see Section 6.4).

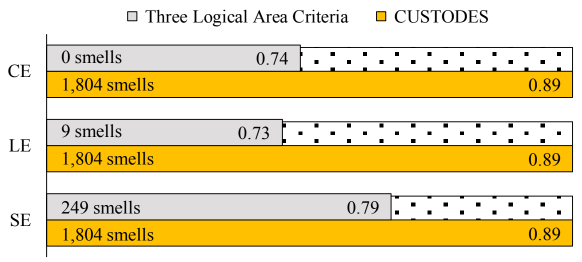


Figure 4. Comparison of recall and true smells clustered

Based on the above results and comparisons, we derive our answer to RQ1*: Our two-stage bootstrapping clustering approach is effective. It outperforms existing clustering techniques in recall while preserving a high precision.*

6.4 Smell Detection Evaluation Results

In order to evaluate the effectiveness of our outlier-based smell detection for two formula-related smells, i.e., missing formula and dissimilar formula (see Section 4.4.2), we made comparisons with AmCheck [19], UCheck [3], Dimension Inference [14], and the built-in error checking of Excel 2013. We enabled all checking rules in Excel 2013 except the two rules: (1) cells containing years are represented as two digits, and (2) numbers are formatted as text or preceded by an apostrophe. This is because the two rules are irrelevant to formula-related smell detection.

Our smell detection reports a score for each cell to represent its outlierness in its cluster. As explained in Section 4.4, cells with a score less than or equal to 1 are inliers. Hence, in experiments, we considered those cells with scores larger than 1 smelly or outliers.

In total, CUSTODES identified 1,582 cell clusters, of which 510 (32.23%) were considered smelly, i.e., suffering at least one type of smell. It further detected 2,443 smelly cells in these problematic cell clusters. Among these suspicious cells, 1,583 (64.80%) were confirmed as true positives. Table 3 reports more detailed results. As listed in Table 1, our 70 spreadsheet subjects in total contain 291 worksheets, out of which 60 (20.62%) come from the database category. These worksheets contribute the largest number of formula cells in our experiments. The result in Table 3 shows that

Table 3. Smell detection results compared with existing smell/error detection techniques

Category	Smell Detection Results of Different Techniques									
	CUSTODES		AmCheck		UCheck		Dimension Inference		Excel's Error Checking	
	Detected	True	Detected	True	Detected	True	Detected	True	Detected	True
cs101	3	3	6	1	0	0	0	0	1	0
database	1,116	1,066	823	790	158	0	863	4	563	18
financial	651	317	502	287	0	0	151	1	1,204	70
forms3	29	10	73	1	0	0	0	0	464	2
grades	316	94	79	78	0	0	11	0	322	8
homework	95	47	109	16	0	0	370	0	1,238	6
inventory	144	27	372	37	34	0	190	2	391	17
modeling	89	19	199	21	12	1	112	4	798	22
Total	**2,443**	**1,583 (64.80%)**	**2,163**	**1,231 (56.91%)**	**204**	**1 (0.49%)**	**1,697**	**11 (0.65%)**	**4,981**	**143 (2.87%)**

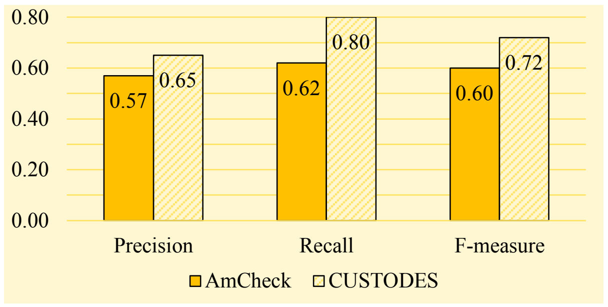


Figure 5. Performance comparison with AmCheck

these formula cells also account for the largest proportion of the detected smells. We do not observe a specific category whose formula cells are significantly more smelly than the others.

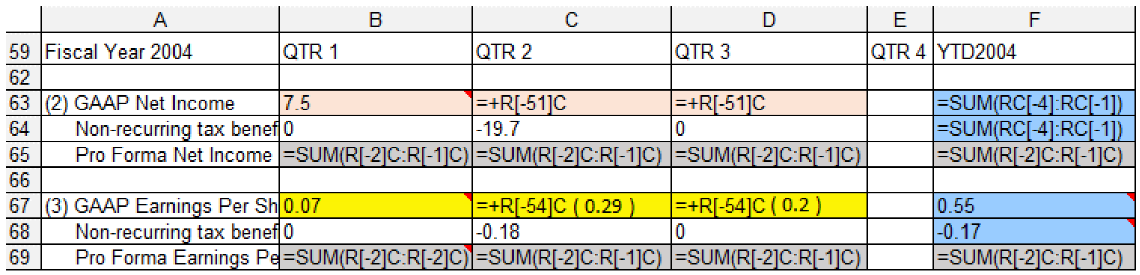
As a comparison, AmCheck detected 2,163 smelly cells in these spreadsheet files. 1,231 (56.91%) were confirmed to be true positives. CUSTODES is capable of detecting 352 more true smelly cells than AmCheck. UCheck [3] and Dimension Inference [14] were proposed to validate calculation by checking whether there are illegal combinations of incompatible units or dimensions as inferred by them. They marked 204 and 1,697 cells as erroneous, respectively. We manually checked these cells and found that only 1 (0.49%) of the 204 cells reported by UCheck and 11 (0.65%) of the 1,697 cells reported by Dimension Inference are erroneous or smelly. We conjecture that the low precision of UCheck is due to its imperfect unit analysis. For example, we observed in our experiments that an addition of a cell labelled as "Indian Point 2" and a cell labeled as "Indian Point 3" was marked as erroneous by UCheck. Although the two cells have slightly different labels, the addition of their values should not be considered an illegal operation. In fact, the two labels reflect a specific tabulation style by users. On the other hand, Dimension Inference is not applicable to those cells that are not associated with clear dimensions in spreadsheets. An earlier study [13] also showed that dimensions occur infrequently in spreadsheets (only occur in 25% of the formula spreadsheets). As a result, Dimension Inference can miss many erroneous cells. Moreover, both UCheck and Dimension Inference are only applicable to formula cells. As such, smelly cells without formulas cannot be detected. As for Excel 2013, it only detected 143 true smelly cells. Most of these smelly cells were detected by three error checking rules: (1) formulas that do not match nearby formulas (detected 23 true smelly cells), (2) formula omitting cells in a region (detected 37 true smelly cells), and (3) formulas that refer to empty cells (detected 81 true smelly cells).

Figure 5 further compares CUSTODES with AmCheck in terms of precision, recall and F-measure in smell detection. CUSTODES achieved a precision of 0.65, recall of 0.80, and F-measure of 0.72, all higher than those of AmCheck (precision of 0.57, recall of 0.62 and F-measure of 0.60). These experimental results show that our outlier-based smell detection outperforms AmCheck, UCheck, Dimension Inference and Excel's built-in error checking support.

Based on these results, we derive our answer to RQ2: *Our outlier-based smell detection can effectively locate smelly cells. It significantly outperforms existing smell detection techniques.*

7. DISCUSSION

Figure 6 gives a typical example from our experiments. It is an excerpt from a EUSES spreadsheet that reports quarterly financial data in each fiscal year. The four cell clusters identified by our two-stage approach are shaded in four different colors. We made two observations on the user's tabulation style: (1) The region of rows 63, 64 and 65 capture similar type of data and computation as the region of rows 67, 68 and 69, where each pair of peer cells (e.g., C65 and C69) in rows 65 and 69 share similar computation. This favors a weak clustering feature based on a gap template with a vertical offset of 4. (2) Users using header information, such as "Pro Forma Earning per share" as a summation calculation indicator. There are four missing formula smells and one dissimilar formula smell detected by our tool (marked by "◥"). Noticeable smell examples are cells F67 and F68. Although a spatial contiguous clustering method might cluster them together with F69, they prescribe essentially different computations. F69 is a summation of some vertical cells above, while F67 and F68 should follow the same calculation as F63 and F64 to sum up the cells on their left. This exposes the requirement and thus challenge for automatically deciding the extension of user table design. CUSTODES tackles this challenge by capturing the user's specific tabulation style in terms of weak features. Actually, the two missing formula smells in cells F67 and F68 likely reveal real data discrepancies. After applying the formula in cells F63 and F64, cell F67's value changes from 0.55 to 0.56, and F68's value changes from -0.17 to -0.18. No existing automatic smell detection approaches can detect the two smells/errors. The success of clustering F67 and F68 with formula cells (F63 and F64) using weak features contributes greatly to the detection of the two smells.



	A	B	C	D	E	F
59	Fiscal Year 2004	QTR 1	QTR 2	QTR 3	QTR 4	YTD2004
62						
63	(2) GAAP Net Income	7.5	=+R[-51]C	=+R[-51]C		=SUM(RC[-4]:RC[-1])
64	Non-recurring tax benef	0	-19.7	0		=SUM(RC[-4]:RC[-1])
65	Pro Forma Net Income	=SUM(R[-2]C:R[-1]C)	=SUM(R[-2]C:R[-1]C)	=SUM(R[-2]C:R[-1]C)		=SUM(R[-2]C:R[-1]C)
66						
67	(3) GAAP Earnings Per Sh	0.07	=+R[-54]C (0.29)	=+R[-54]C (0.2)		0.55
68	Non-recurring tax benef	0	-0.18	0		-0.17
69	Pro Forma Earnings Pe	=SUM(R[-2]C:R[-2]C)	=SUM(R[-2]C:R[-1]C)	=SUM(R[-2]C:R[-1]C)		=SUM(R[-2]C:R[-1]C)

Figure 6. An typical example output from CUSTODES

8. RELATED WORK

There are several approaches proposed to identify spatially, structurally or semantically related blocks or regions of cells in spreadsheet visualization. Mittermeir and Clermont [39] introduced three kinds of "logical areas" to cluster formula cells that satisfy three forms of equivalences: copy, logical and structural equivalences. Among the three equivalences, copy equivalence that requires identical formulas is the strongest, and structural equivalence that requires only the same set of operations is the weakest. Sajaniemi [46] also employs equivalent formulas to identify cell blocks. While existing work clusters cells based on formula similarity, our two-stage approach goes beyond that and takes advantage of weak features to identify cell clusters. The use of weak features enables us to cluster smelly cells even though their formulas are dissimilar to their peers.

Various techniques have been proposed to detect different kinds of spreadsheet smells or errors. Unit and type inference is quite a major research topic in the last decade. UCheck [3] and dimension inference [14] are proposed to derive unit or dimensional information to verify the correctness of calculation by checking whether there is any illegal combination of incompatible units. Hermans et al. implemented tools to detect and visualize inter-worksheet smells [29], data clones [31], formula smells [30], lookup smells [32] by adapting the concept of code smells to the spreadsheet domain. Cunha et al. [16] presented a catalog of spreadsheet smells by considering spreadsheet-specific data statistics features. AmCheck [19] catched potential errors caused by ambiguous computation in a cell array. AmCheck's smell detection works in two steps. It first clusters cells to cell arrays. AmCheck adopts a simple heuristic to group contiguous cells in a row or column into a cluster. A cell array is then considered smelly if it contains cells that prescribe inequivalent formulas. CheckCell [11] identifies those data entries that cause extraordinary impact on results as potential data errors. VEnron [20] publishes the first versioned spreadsheet corpus to facilitate evolution analysis.

Spreadsheet quality is also studied by the information systems community. Galletta et al. [24] in an empirical study reports that spreadsheet experts did not outperform novices in identifying formula errors, and that identifying simple spreadsheet errors is non-trivial even for practitioners. Another empirical experiment by Nixon and O'Hara [40] reports that spreadsheet auditing software can assist auditors to detect errors. Anderson [7] in a later study confirms the finding but reports that auditing software tools missed many spreadsheet errors. The study concludes that both tools and competent users are needed for spreadsheet error detection.

This paper differs from the existing techniques. It makes the first effort to derive clusters of spreadsheet cells using a two-stage technique based on strong and weak features. It proposes to detect multiple types of smells uniformly based on outlier detection in feature spaces. In addition to smell detection, there are other pieces of work related to smell refactoring [10] and fault localization using smells [6], which target at different scopes from our work.

9. THREATS TO VALIDITY

A threat to internal validity of our evaluation is that we are not able to confirm the ground truth that were manually marked by us, since EUSES corpus was collected in the wild without any labeling of true smells or errors. We alleviated this by making the data public for crosschecking and experiment repeating [17]. Another threat to the external validity is the representativeness of experimental subjects used in our evaluation. We chose a set of 70 spreadsheets from the EUSES corpus [21]. These spreadsheets are non-trivial, containing 26,716 formulas and 1,610 clusters. They span across major application domains categorized by EUSES, which is the most popular corpus that has been used for spreadsheet evaluation.

10. CONCLUSION

In this paper, we have proposed an automatic spreadsheet cell clustering approach by constructing a feature model that is capable of extracting invariant tabulation styles as strong features and variant tabulation styles as weak features. Two formula-based strong features and seven weak features are presented. A two-stage clustering technique is proposed to effectively aggregate spreadsheet cells into clusters. Our evaluation shows that our clustering technique is able to achieve a high precision (0.91) and recall (0.89) at the same time, and successfully group smelly cells into their own clusters. About one-third of clusters are found to be smelly, confirming previous findings in the literature that spreadsheets are error-prone. It is time-consuming to identify all smells manually. To address this issue, we have also proposed an automatic outlier-based smell detection technique. A wide class of smells can be characterized as outliers of various feature spaces. Our evaluation results show that our smell detection technique achieves large improvements in F-measure (0.72) as compared to state-of-the-art techniques.

In conclusion, our CUSTODES approach is effective in cell clustering and smell detection. In future, we plan to further improve CUSTODES by identifying more spreadsheet user behaviors and modeling them as features.

11. ACKNOWLEGMENTS

The research was supported by RGC/GRF Grant 611811 of Hong Kong SAR, and by National Basic Research 973 Program (Grant No. 2015CB352202) and National Natural Science Foundation (Grant Nos. 61472174, 91318301, 61321491) of China. We would like to thank Ruiqing Zhang for his great help in validating experimental results and the writing of this paper. We also want to greatly thank the ICSE 2016 reviewers for their valuable comments and suggestions, which helped improve our paper.

12. REFERENCES

[1] R. Abraham and M. Erwig. 2004. Header and unit inference for spreadsheets through spatial analyses. In *Proceedings of the 2004 IEEE Symposium on Visual Languages and Human Centric Computing*. VL/HCC '04. 165–172.

[2] R. Abraham and M. Erwig. 2006. Inferring templates from spreadsheets. In *Proceedings of the 28th International Conference on Software Engineering*. ICSE '06. 182–191.

[3] R. Abraham and M. Erwig. 2007. UCheck: A spreadsheet type checker for end users. In *Journal of Visual Languages & Computing*, 18(1):71–95.

[4] R. Abraham and M. Erwig. 2009. Mutation operators for spreadsheets. In *IEEE Transactions on Software Engineering*, 35(1):94–108.

[5] R. Abreu, J. Cunha, J. P. Fernandes, P. Martins, A. Perez, and J. Saraiva. 2014. Smelling faults in spreadsheets. In *Proceedings of the 30th International Conference on Software Maintenance and Evolution*. ICSME '14. 111–120.

[6] R. Abreu, P. Zoeteweij, and A. J. Van Gemund. 2006. An evaluation of similarity coefficients for software fault localization. In *Proceedings of the 12th Pacific Rim International Symposium on Dependable Computing*. PRDC '06. 39–46.

[7] W. Anderson. 2004. A comparison of automated and manual spreadsheet error detection. Master thesis, Massey University.

[8] Apache POI: the Java API for Microsoft Documents. URL: https://poi.apache.org/.

[9] Y. Ayalew, M. Clermont, and R. T. Mittermeir. 2000. Detecting errors in spreadsheets. In *Proceedings of the European Spreadsheet Risks Interest Group Annual Conference*. EuSpRIG '00.

[10] S. Badame and D. Dig. 2012. Refactoring meets spreadsheet formulas. In *Proceedings of the 28th International Conference on Software Maintenance.* ICSM '12. 399–409.

[11] D.W. Barowy, D. Gochev, and E.D. Berger. 2014. CheckCell: Data Debugging for Spreadsheets. In *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages & Applications*. OOPSLA '14. 507–523.

[12] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. 2000. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data.* SIGMOD '00. 93–104.

[13] J. P. Caulkins, E. L. Morrison, and T. Weidemann. 2007. Spreadsheet errors and decision making: evidence from field interviews. In *Journal of Organizational and End User Computing*, 19(3):1–23.

[14] C. Chambers and M. Erwig. 2009. Automatic detection of dimension errors in spreadsheets. In *Journal of Visual Languages & Computing*, 20(4):269–283.

[15] M. Clermont. A toolkit for scalable spreadsheet visualization. 2008. arXiv preprint arXiv:0802.3924.

[16] J. Cunha, J. P. Fernandes, J. Mendes, and J. Saraiva. 2015. Spreadsheet engineering. In *Lecture Notes in Computer Science*, volume 8606, 246–299.

[17] CUSTODES project. URL: http://sccpu2.cse.ust.hk/custodes/.

[18] I. S. Dhillon, S. Mallela, and D. S. Modha. 2003. Information-theoretic co-clustering. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '03. 89–98.

[19] W. Dou, S.C. Cheung, and J. Wei. 2014. Is spreadsheet ambiguity harmful? Detecting and repairing spreadsheet smells due to ambiguous computation. In *Proceedings of the 36th International Conference on Software Engineering*. ICSE '14. 848–858.

[20] W. Dou, L. Xu, S.C. Cheung, C. Gao, J. Wei, and T. Huang. 2016. VEnron: A Versioned Spreadsheet Corpus and Related Evolution Analsysis. In *Proceedings of the 38th International Conference on Software Engineering*. ICSE'16 - SEIP. To appear.

[21] M. Fisher and G. Rothermel. 2005. The Euses spreadsheet corpus: a shared resource for supporting experimentation with spreadsheet dependability mechanisms. In *ACM SIGSOFT Software Engineering Notes,* volume 30, 1–5.

[22] M. Fisher, G. Rothermel, T. Creelan, and M. Burnett. 2006. Scaling a dataflow testing methodology to the multiparadigm world of commercial spreadsheets. In *Proceedings of 17th IEEE International Symposium on Software Reliability Engineering*. ISSRE '06. 13–22.

[23] M. Fowler. 1999. Refactoring: improving the design of existing code. Addison-Wesley.

[24] D. Galletta, D. Abraham, and et al. 1993. An empirical study of spreadsheet error-finding performance. In *Accounting, Management, and Information Technologies*, 3(2), 79–95.

[25] S. Gandel. Damn Excel! How the 'most important software application of all time' is ruining the world. URL: http://fortune.com/2013/04/17/damn-excel-how-the-most-important-software-application-of-all-time-is-ruining-the-world/. Last accessed 13/2/2016.

[26] J. Han, M. Kamber, and J. Pei. 2006. Data mining, Southeast Asia edition: Concepts and techniques. Morgan Kaufmann.

[27] A. B. Hassanat, M. A. Abbadi, G. A. Altarawneh, and A. A. Alhasanat. 2014. Solving the Problem of the K Parameter in the KNN Classifier Using an Ensemble Learning Approach. arXiv preprint arXiv:1409.0919.

[28] F. Hermans, M. Pinzger, and A. van Deursen. 2010. Automatically extracting class diagrams from spreadsheets. In *Proceedings of the 24th European Conference on Object-Oriented Programming*. ECOOP '10. 52–75.

[29] F. Hermans, M. Pinzger, and A. van Deursen. 2012. Detecting and visualizing inter-worksheet smells in spreadsheets. In *Proceedings of the 34th International Conference on Software Engineering*. ICSE '12. 441–451.

[30] F. Hermans, M. Pinzger, and A. van Deursen. 2012. Detecting code smells in spreadsheet formulas. In *Proceedings of the 28th International Conference on Software Maintenance*. ICSM '12. 409–418.

[31] F. Hermans, B. Sedee, M. Pinzger, and A. v. Deursen. 2013. Data clone detection and visualization in spreadsheets. In *Proceedings of the 35th International Conference on Software Engineering*. ICSE '13. 292–301.

[32] F. Hermans, E. Aivaloglou, and B. Jansen. 2015. Detecting and Repairing Smelly Lookup Functions in Spreadsheets. In *Proceedings of the 2015 IEEE Symposium on Visual Languages and Human Centric Computing.* VL/HCC '15. 153–157.

[33] S. Hipfl. Using layout information for spreadsheet visualization. arXiv preprint arXiv:0802.3939, 2008.

[34] D. Jannach, T. Schmitz, B. Hofer, and F. Wotawa. 2014. Avoiding, finding and fixing spreadsheet errors–a survey of automated approaches for spreadsheet. In *Journal of Systems and Software*, 94:129–150.

[35] M. Jirina and M. Jirina. 2010. Using singularity exponent in distance based classifier. In *Proceedings of the 10th International Conference on Intelligent Systems Design and Applications*. ISDA '10. 220–224.

[36] M. Jirina and M. Jirina Jr. 2011. Classifiers Based on Inverted Distances. INTECH Open Access Publisher.

[37] A. J. Ko, R. Abraham, L. Beckwith, A. Blackwell, M. Burnett, M. Erwig, C. Scaffidi, J. Lawrance, H. Lieberman, B. Myers, et al. 2011. The state of the art in end-user software engineering. In *ACM Computing Surveys (CSUR)*, 43(3):21.

[38] C. D. Manning, P. Raghavan, and H. Schutze. 2008. Introduction to information retrieval. Cambridge University Press.

[39] R. Mittermeir and M. Clermont. 2012. Finding high-level structures in spreadsheet programs. In Proceedings of the Ninth Working Conference on Reverse Engineering. WCRE '12. 221–232.

[40] D. Nixon and M. O'Hara. 2001. Spreadsheet Auditing Software. In *Proceedings of the European Spreadsheet Risks Interest Group Annual Conference*. EuSpRIG '01.

[41] R. R. Panko. 1998. What we know about spreadsheet errors. In *Journal of Organizational and End User Computing*, 10(2):15–21.

[42] R. R. Panko and N. Ordway. 2008. Sarbanes-oxley: What about all the spreadsheets? arXiv preprint arXiv:0804.0797.

[43] P. Pantel and M. Pennacchiotti. 2006. Espresso: Leveraging generic patterns for automatically harvesting semantic relations. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*. COLING/ACL '06. 113–120.

[44] M. Pawlik and N. Augsten. 2011. Rted: a robust algorithm for the tree edit distance. In *Proceedings of the VLDB Endowment*, 5(4):334–345.

[45] S. G. Powell, K. R. Baker, and B. Lawson. 2008. A critical review of the literature on spreadsheet errors. In *Decision Support Systems*, 46:128–138.

[46] J. Sajaniemi. 2000. Modeling spreadsheet audit: A rigorous approach to automatic visualization. In *Journal of Visual Languages & Computing*, 11(1):49–82.

[47] C. Scaffidi, M. Shaw, and B. Myers. 2005. Estimating the numbers of end users and end user programmers. In *Proceedings of the 2005 IEEE Symposium on Visual Languages and Human Centric Computing*. VL/HCC '05. 207–214.

[48] J. Walkenbach et al. 1999. Microsoft Excel 2000 power programming with VBA. John Wiley & Sons, Inc..

[49] M. Yoshida, M. Ikeda, S. Ono, I. Sato, and H. Nakagawa. 2010. Person name disambiguation by bootstrapping. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '10. 10–17.