Autom Softw Eng (2015) 22:47–74
DOI 10.1007/s10515-014-0145-3

On the empirical evaluation of similarity coefficients for spreadsheets fault localization

Birgit Hofer · Alexandre Perez · Rui Abreu · Franz Wotawa

Received: 7 October 2013 / Accepted: 17 February 2014 / Published online: 22 March 2014

Abstract Spreadsheets are by far the most prominent example of end-user programs of ample size and substantial structural complexity. They are usually not thoroughly tested so they often contain faults. Debugging spreadsheets is a hard task due to the size and structure, which is usually not directly visible to the user, i.e., the functions are hidden and only the computed values are presented. A way to locate faulty cells in spreadsheets is by adapting software debugging approaches for traditional procedural or object-oriented programming languages. One of such approaches is spectrum-based fault localization (SFL). In this paper, we study the impact of different similarity coefficients on the accuracy of SFL applied to the spreadsheet domain. Our empirical evaluation shows that three of the 42 studied coefficients (Ochiai, Jaccard and Sorensen-Dice) require less effort by the user while inspecting the diagnostic report, and can also be used interchangeably without a loss of accuracy. In addition, we illustrate the influence of the number of correct and incorrect output cells on the diagnostic report.

Keywords End-user debugging · Spreadsheets · Similarity coefficients · Spectrum-based fault localization

B. Hofer (✉) · F. Wotawa
Institute for Software Technology, Graz University of Technology, Graz, Austria
e-mail: bhofer@ist.tugraz.at

F. Wotawa
e-mail: wotawa@ist.tugraz.at

A. Perez · R. Abreu
Department of Informatics Engineering, Faculty of Engineering, University of Porto, Porto, Portugal
e-mail: alexandre.perez@fe.up.pt

R. Abreu
e-mail: rui@computer.org

1 Introduction

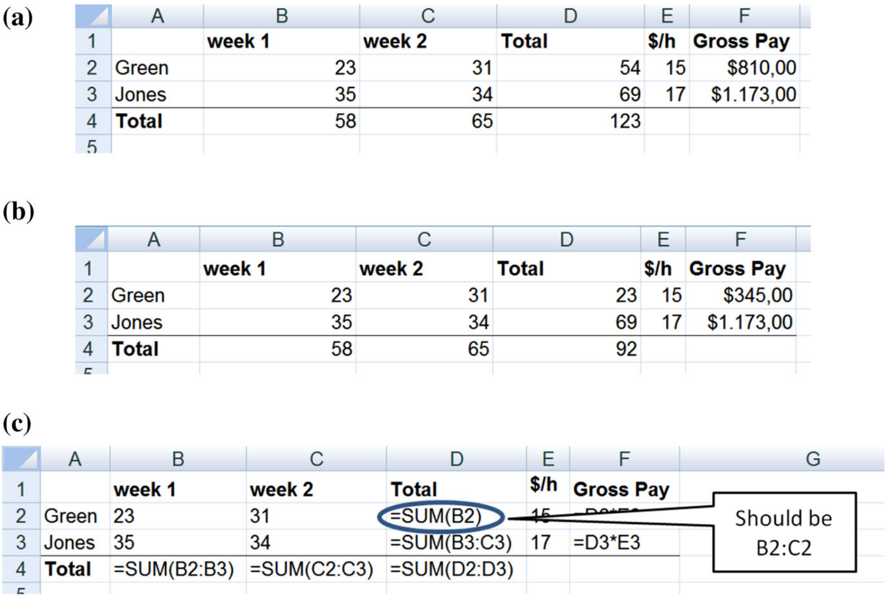
Spreadsheet tools, such as Microsoft Excel, iWork's Numbers, and OpenOffice's Calc, can be viewed as programming environments for non-professional programmers (Ko et al. 2011). In fact, these so-called "end-user" programmers vastly outnumber professional ones: the US Bureau of Labor and Statistics estimates that more than 55 million people use spreadsheets and databases at work on a daily basis (Ko et al. 2011). Despite this trend, as a programming language, spreadsheets lack support for abstraction, testing, encapsulation, or structured programming. As a consequence, spreadsheets are error-prone. Numerous studies have shown that existing spreadsheets contain redundancy and errors at an alarmingly high rate (Chadwick et al. 2001).

Furthermore, spreadsheets are applications created by single end-users without planning ahead of time for maintainability or scalability. After their initial creation, many spreadsheets turn out to be used for storing and processing increasing amounts of data as well as supporting increasing numbers of users over long periods of time. Therefore, debugging (i.e., locating the cell(s) that are responsible for the wrong output in a given cell) can be a rather cumbersome task, requiring substantial time and effort.

There is potential in the application of software engineering techniques to the spreadsheet world, so that such techniques will improve the overall quality of spreadsheets. So far, only a few attempts have been made in this field (Abreu et al. 2012; Hofer et al. 2013). We build on top of previous work that described how to modify traditional fault localization techniques in order to render them applicable to the spreadsheet world (Hofer et al. 2013), namely with the use of spectrum-based fault localization (SFL). The accuracy of SFL is directly related to the similarity coefficient used to compare component traces to execution outcomes. In previous work (Hofer et al. 2013), several approaches to spreadsheet debugging were compared, and SFL was deemed amongst the best performing techniques in terms of both computation time and diagnostic accuracy. However, the paper only used one similarity coefficient for ranking faulty candidates.

Studies have been made to assess the accuracy of several coefficients in software debugging (Lucia et al. 2013), but so far none focused on the spreadsheet domain. In this paper, we present and describe 42 similarity coefficients used in different fields of research: from software fault localization (Abreu et al. 2007; Jones and Harrold 2005) and clustering analysis (Anderberg 1973) to biology (Ochiai 1957) and medicine (Ohsaki et al. 2004). We evaluate the impact that each similarity coefficient has on the quality of the diagnoses using real spreadsheets taken from the EUSES Spreadsheet Corpus (Fisher and Rothermel 2005) and a spreadsheet version of the ISCAS85 circuits (Brglez and Fujiwara 1985).

In order to explain the basic concepts and the application of SFL to spreadsheets, we make use of a running example. Figure 1 shows this example spreadsheet which is borrowed from the EUSES spreadsheet corpus (Fisher and Rothermel 2005). For the sake of clarity, we have reduced the number of columns and rows of this example spreadsheet. Figure 1a illustrates the correct version of this spreadsheet and Fig. 1b a faulty variant of the same spreadsheet. This spreadsheet is used to calculate the wages



(a)

	A	B	C	D	E	F
1		week 1	week 2	Total	$/h	Gross Pay
2	Green	23	31	54	15	$810,00
3	Jones	35	34	69	17	$1.173,00
4	Total	58	65	123		
5						

(b)

	A	B	C	D	E	F
1		week 1	week 2	Total	$/h	Gross Pay
2	Green	23	31	23	15	$345,00
3	Jones	35	34	69	17	$1.173,00
4	Total	58	65	92		
5						

(c)

	A	B	C	D	E	F	G
1		week 1	week 2	Total	$/h	Gross Pay	
2	Green	23	31	=SUM(B2)	15	=D2*E2	
3	Jones	35	34	=SUM(B3:C3)	17	=D3*E3	
4	Total	=SUM(B2:B3)	=SUM(C2:C3)	=SUM(D2:D3)			
5							

Fig. 1 'Workers' example borrowed from the EUSES spreadsheet corpus (Fisher and Rothermel 2005) **a** Correct spreadsheet, **b** faulty spreadsheet, **c** formula view of the spreadsheet from Fig. 1b

of the workers (cells F2:F3) and the total working hours (cell D4). Figure 1c shows the formula view of the faulty spreadsheet from Fig. 1b. In this faulty spreadsheet, the computation of the total hours for the worker "Green" (cell D2) is faulty because the programmer of the spreadsheet unintentionally set a wrong area for the SUM formula. This happens for example when a programmer adds a new week but forgets to adapt some calculations. Because of this fault, the wage of the worker "Green" (cell F2) and the total hours (cell D4) are erroneous. In the following sections, we show how to use SFL to pinpoint the faulty cell D2.

The remainder of the paper is organized as follows: Section 2 deals with the related work. In addition, existing spreadsheet debugging and testing techniques are discussed. Section 3 deals with the syntax and semantics of spreadsheets. Furthermore, the spreadsheet debugging problem is defined. Section 4 explains the changes that have to be made in order to use SFL for the debugging of spreadsheets. In addition, the similarity coefficients are discussed. Section 5 deals with the setup and the results of the empirical evaluation. Finally, Section 6 concludes this paper and presents ideas for future empirical evaluations.

2 Related work

Basically, our paper is based on the work of Lo et al. (2010), Lucia et al. (2013) which compares the fault localization capabilities of 42 similarity coefficients for programs written in C. In contrast to them, we focus on spreadsheets. To the best of our knowledge, there has not been published any paper that compares similarity coefficients for spreadsheets. However, there exists work about debugging spreadsheets. Ruthruff et al. (2003) propose three techniques for visualizing potentially faulty cells. The first of those three techniques is based the ideas of Tarantula, but the approach is incremental. The second technique is a blocking technique that can be compared to program dicing. The third technique is a nearest consumer approach. Jannach et al. (2013) have created the EXQUISTE framework for declarative debugging of spreadsheets. This framework is based on previous work of Jannach and Engler (2010) about model-based debugging in spreadsheets. The framework is enhanced by a graphical user interface which carefully considers the needs of end-users. The underlying model-based approach relies on an extended hitting-set algorithm and user-specified test cases. Ayalew and Mittermeir (2003) address the spreadsheet debugging problem by presenting a trace-based fault localization approach. Their approach is data-flow driven and uses the concept of slices in the spreadsheet domain. The authors propose an approach that prioritizes cells based on the number of incorrect successor cells and predecessor cells.

GoalDebug (Abraham and Erwig 2005, 2007) is a spreadsheet debugger offering repair suggestions. Whenever the computed output of a cell is incorrect, the user can supply an expected value for a cell, which is employed by the system to generate a list of change suggestions for formulas that, when applied, would result in the user-specified output. In Abraham and Erwig (2007), a thorough evaluation of the tool is given. As no implementation of the approach is available for experimental comparison (Abraham and Erwig 2013), we refrain from empirically comparing our approach to GoalDebug. Other mentionable work of Abraham and Erwig includes the definition of mutation operators for spreadsheets (Abraham and Erwig 2009) and the UCheck system (Abraham and Erwig 2007). UCheck detects errors that are caused by unit faults by analyzing the header information of spreadsheets and reasoning about the formulas. Coblenz et al. (2005) also reasoned about errors using the header information. Coblenz introduced the SLATE system, short for "A Spreadsheet Language for Accentuating Type Errors". This spreadsheet language separates the unit from the object of measurement. This technique helps to detect spreadsheet formula errors.

Spreadsheet debugging is closely related to testing. There exist manual testing approaches, e.g., WYSIWYT (Rothermel et al. 2000), and approaches that automatically generate test cases for spreadsheets, e.g., (Fisher et al. 2002) and AUTOTEST (Abraham and Erwig 2006). In the WYSIWYT system, users can indicate incorrect output values by placing a faulty token in the cell. Similarly, they can indicate that the value in a cell is correct by placing a correct token (Rothermel et al. 2000). When a user indicates one or more program failures during this testing process, fault localization techniques direct the user's attention to the possible faulty cells. However, WYSIWYT does not provide any suggestions for how to change faulty formulas.

Since spreadsheet *developers* are typically end-users without significant background in computer science, there has been considerable effort to adapt software engineering principles to form a spreadsheet engineering discipline: Burnett et al. (2003) suggest to use assertions in the spreadsheet domain. Cunha et al. (2012) specialized on model-driven spreadsheet engineering. Mittermeir and Clermont (2002) focus on identifying high-level structures in spreadsheets. Bregar (2008) developed metrics for determining the complexity of spreadsheet models. Hermans et al. (2012a, c) address the issue of code smells in spreadsheets. In particular, they transform code smells defined for object-oriented programs (e.g. coupling and cohesion of classes) to the spreadsheet domain. Spreadsheet code smells are an important tool for improving spreadsheet quality with respect to usability, maintainability and error frequency. Other work of Hermans et al. includes the visualization of the dataflow in spreadsheets (Hermans et al. 2011), class diagram extraction (Hermans et al. 2010), data clone detection (Hermans et al. 2013), and metrics measuring the understandability of spreadsheet formulas (Hermans et al. 2012b).

3 Basic definitions

A spreadsheet is a matrix comprising cells. Each cell is unique and can be accessed using its corresponding column and row number. For simplicity, we assume a function φ that maps the cell names from a set CELLS to their corresponding position (x, y) in the matrix where x represents the column and y the row number. The functions φ_x and φ_y return the column and row number of a cell respectively.

Aside from a position, each cell $c \in$ CELLS has a value $\nu(c)$ and an expression $\ell(c)$. The value of a cell can be either undefined ϵ, an error $\perp$, or any number, boolean or string value. The expression of a cell $\ell(c)$ can either be empty or an expression written in the language $\mathcal{L}$. The value of a cell c is determined by its expression. If no expression is explicitly declared for a cell, the function ℓ returns the value ϵ. An area $\bar{\mathcal{C}}$ is a set of cells ($\bar{\mathcal{C}} \subseteq$ CELLS). Areas might represent a part of a spreadsheet that is within a bound specified by two cells.

After defining the basic elements of spreadsheets, we introduce the language $\mathcal{L}$ for representing expressions that are used to compute values for cells. The introduced language takes the values of cells and constants together with operators and conditionals to compute values for other cells. The language is a functional language, i.e., only one value is computed for a specific cell. Moreover, we do not allow recursive functions. First, we define the syntax of $\mathcal{L}$.

Definition 1 (*Syntax of* $\mathcal{L}$) We define the syntax of $\mathcal{L}$ recursively as follows:

- Constants k representing ϵ, number, boolean, or string values are elements of $\mathcal{L}$ (i.e., $k \in \mathcal{L}$).
- All cell names are elements of $\mathcal{L}$ (i.e., CELLS $\subset \mathcal{L}$).
- All areas $\bar{\mathcal{C}} \subseteq CELLS$ are element of $\mathcal{L}$.
- If e_i for $i = 0 \ldots n$ is an element of $\mathcal{L}$, then the following expressions are also elements of $\mathcal{L}$:
  - (e_0) is an element of $\mathcal{L}$.
  - A function call $f(e_0, \ldots, e_n)$ is an element of $\mathcal{L}$ where f denotes functions like **+**, **-**, *, **/**, <, =, >, **IF**, **SUM**, **AVG**, etc.

Second, we define the semantics of $\mathcal{L}$ by introducing an interpretation function $[\![\cdot]\!]$ that maps an expression $e \in \mathcal{L}$ to a value. The value is ϵ if no value can be determined or $\perp$ if a type error occurs. Otherwise it is either a number, a boolean, a string, or an area.

Definition 2 (*Semantics of* $\mathcal{L}$) Let e be an expression from $\mathcal{L}$ and ν a function mapping cell names to values. We define the semantic of $\mathcal{L}$ recursively as follows:

- If e is a constant k, then the constant is given back as result, i.e., $[\![e]\!] = k$.
- If e denotes a cell name c, then its value is returned, i.e., $[\![e]\!] = \nu(c)$.
- If e denotes an area $\bar{\mathcal{C}}$, then itself is returned, i.e., $[\![\bar{\mathcal{C}}]\!] = \bar{\mathcal{C}}$.
- If e is of the form (e_0), then $[\![e]\!] = [\![e_0]\!]$.
- If e is of the form $f(e_0,\ldots,e_n)$, then the value returned by the implementation of the function f is returned. Let f_I be the implementation of the function f. The semantics of the call to a function is defined as follows:
  $[\![f(e_0,\ldots,e_n)]\!] = f_I([\![e_0]\!], \ldots, [\![e_n]\!])$
  Note that f_I represents the implementations of operators like **+** or functions like **IF**. The return value might also be $\perp$ in case of type errors or mismatches of arguments given.

Frequently, we require information about cells that are used as input in an expression. We call such cells *referenced cells*.

Definition 3 (*Referenced cell*) A cell c is said to be referenced by an expression $e \in \mathcal{L}$, if and only if c is used in e.

We furthermore introduce a function $\rho : \mathcal{L} \mapsto 2^{\text{CELLS}}$ that returns the set of referenced cells. Formally, we define ρ as follows:

Definition 4 (*The function* ρ) Let $e \in \mathcal{L}$ be an expression. We define the referenced cells function ρ recursively as follows:

- If e is a constant, then $\rho(e) = \emptyset$.
- If e is a cell c, then $\rho(e) = \{c\}$.
- If e is an area $\bar{\mathcal{C}}$, then $\rho(e) = \bar{\mathcal{C}}$.
- If $e = (e_0)$, then $\rho(e) = \rho(e_0)$.
- If $e = f(e_0,\ldots,e_n)$, then $\rho(e) = \bigcup_{i=0}^{n} \rho(e_i)$.

A spreadsheet is a matrix of cells comprising values and expressions written in a language $\mathcal{L}$. In addition, we know that the values of cells are determined by their expressions. Hence, we can state that $\forall c \in \text{CELLS} : \nu(c) = [\![\ell(c)]\!]$ must hold. Unfortunately, we face two challenges: (1) In all of the previous definitions, the set of cells need not be of finite size. (2) There might be a loop in the computation of values, e.g. a cell c with $\ell(c) = c{+}1$. In this case, we are not able to determine a value for cell c. In order to solve the first challenge, we formally restrict spreadsheets to comprise only a finite number of cells.

Definition 5 (*Spreadsheet*) A countable set of cells $\Pi \subseteq \text{CELLS}$ is a spreadsheet if all cells in Π have a non empty corresponding expression or are referenced in an expression, i.e., $\forall c \in \Pi : (\ell(c) \neq \epsilon) \vee (\exists c' \in \Pi : c \in \rho(\ell(c')))$.

In order to solve the second challenge, we have to limit spreadsheets to loop-free spreadsheets. Although spreadsheet engines allow loops[1], exploring the underlying

[1] Known as iterative calculations; see http://office.microsoft.com/en-us/excel-help/remove-or-allow-a-circular-reference-HP010066243.aspx

stop criteria makes it possible to convert a spreadsheet into a loop-free version. As an example, Excel stops calculating after 100 iterations or after all values in the circular reference change by less than 0.001 between iterations, whichever comes first. However, handling of spreadsheets with loops is not in the focus of this paper. Standard techniques like unrolling loops that has been used in other debugging techniques might also be used in the spreadsheet domain.

In order to define loop-free spreadsheets, which we call feasible spreadsheets, we first introduce the notation of data dependence between cells, and furthermore the data dependence graph (DDG), which represents all dependencies occurring in a spreadsheet.

Definition 6 $(Direct\ dependence)$ Let c_1, c_2 be cells of a spreadsheet Π. The cell c_2 depends directly on cell c_1 if and only if c_1 is used in c_2's corresponding expression, i.e., $dd(c_1, c_2) \leftrightarrow (c_1 \in \rho(\ell(c_2)))$.

The direct dependence definition states the data dependence between two cells. This definition can be extended to the general case in order to specify indirect dependence. In addition, this dependence definition immediately leads to the definition of a graph that can be extracted from a spreadsheet.

Definition 7 $(Data\ dependence\ graph(DDG))$ Let Π be a spreadsheet. The DDG of Π is a tuple (V, A) with:

- V as a set of vertices comprising exactly one vertex n_c for each cell $c \in \Pi$
- A as a set comprising arcs (n_{c_1}, n_{c_2}) if and only if there is a direct dependence between the corresponding cells c_1 and c_2 respectively, i.e. $A = \bigcup (n_{c_1}, n_{c_2})$ where $n_{c_1}, n_{c_2} \in V \wedge dd(c_1, c_2)$.

From this definition, we are able to define general dependence between cells. Two cells of a spreadsheet are dependent if and only if there exists a path between the corresponding vertices in the DDG. In addition, we are able to further restrict spreadsheets to face the second challenge.

Definition 8 $(Feasible\ spreadsheet)$ A spreadsheet Π is feasible if and only if its DDG is acyclic.

From here on, we assume that all spreadsheets of interest are feasible. Hence, we use the terms spreadsheet and feasible spreadsheet synonymously.

In this paper, we focus on testing and debugging of spreadsheets. In ordinary sequential programs, a test case comprises input values and expected output values. If we want to rely on similar definitions, we have to clarify the terms input, output and test case. Defining the input and output of feasible spreadsheets is straightforward by means of the DDG.

Definition 9 $(Input, output)$ Given a feasible spreadsheet Π and its DDG (V, A), then the input cells of Π (or short: inputs) comprise all cells that have no incoming edges in the corresponding vertex of Π's DDG. The output cells of Π (or short: outputs) comprise all cells where the corresponding vertex of the DDG has no outgoing vertex.

$$inputs(\Pi) = \{c | \nexists (n_{c'}, n_c) \in A\}$$
$$outputs(\Pi) = \{c | \nexists (n_c, n_{c'}) \in A\}$$

All cells of a spreadsheet that serve neither as input nor as output are called intermediate cells. With this definition of input and output cells we are able to define a test case for a spreadsheet and its evaluation.

Definition 10 (*Test case*) Given a spreadsheet Π, then a tuple (I, O) is a test case for Π if and only if:

- I is a set of tuples (c, e) specifying input cells and their values. For each $c \in inputs(\Pi)$ there must be a tuple (c, e) in I where $e \in \mathcal{L}$ is a constant.
- O is a set of tuples (c, e) specifying expected output values. The expected output values must be constants of $\mathcal{L}$.

In our setting, test case evaluation works as follows: First, the functions $\ell(c)$ of the input cells are set to the constant values specified in the test case. Subsequently, the spreadsheet is evaluated. Afterwards, the computed output values are compared with the expected values stated in the test case. If at least one computed output value is not equivalent to the expected value, the spreadsheet fails the test case. Otherwise, the spreadsheet passes the test case.

Example 1 A test case for our running example from Fig. 1 is $I = \{B2 = 23, B3 = 35, C2 = 31, C3 = 34, E2 = 15, E3 = 17\}$ and $O = \{B4 = 58, C4 = 65, D4 = 123, F2 = 810, F3 = 1173\}$. The spreadsheet from Fig. 1a passes this test case. The spreadsheet from Fig. 1b fails this test case because it computes different values for the cells D4 and F2.

In traditional programming languages, test cases are separated from the source code. Usually, there are several test cases for one function under test. Each of the test cases calls the function with different parameters and checks the correctness of the returned values. However, test cases are only implicitly encoded into spreadsheets. This means, that test cases are not explicitly separated from the formulas under test. If the user wants to add an additional test case, he or she has to duplicate the spreadsheet. A duplication of a spreadsheet for testing purposes is unpractical since the duplicates have to be updated when the spreadsheet is modified or extended. Therefore, usually only one failing test case exists. Hence, we reduce the debugging problem for spreadsheets to handle only one test case.

Definition 11 (*Spreadsheet debugging problem*) Given a spreadsheet Π and a failing test case (I, O), then the debugging problem is to find a root cause for the mismatch between the expected output values and the computed ones.

We define the spreadsheet debugging problem as a fault localization problem. This definition implies that the following debugging approach pinpoints certain cells of a spreadsheet as possible root causes of faults. However, the approach does not make any suggestions how to change these parts. Alternatively, the debugging problem can be defined as a fault correction problem.

4 Spectrum-based fault localization for spreadsheets

In traditional programming paradigms, SFL (Abreu et al. 2007) uses code coverage data and the pass/fail result of each test execution of a given system under test (SUT) as input. The code coverage data (Tikir and Hollingsworth 2002) is collected from test cases by means of an instrumentation approach. This data is collected at runtime and is used to build a so-called hit-spectra matrix. A hit-spectra matrix is a binary matrix where each row i represents a system component and each column j represents a test case. The content of the matrix element a_{ij} represents whether component i was used (true) or not (false) during test execution j. The results of the test executions (pass/fail) are stored in an error vector. The error vector is a binary array where each position i represents a test execution. The value of the error vector at position i is true if the test case i failed, otherwise false.

In the spreadsheet paradigm, the concept of code coverage does not exist since there are no explicit lines of code like in traditional programming paradigms. Moreover, there is no concept of test execution. Therefore, in order to use SFL on spreadsheets, we have to perform some modifications: the lines of code in a traditional programming paradigm are mapped to the cells of a spreadsheet. There are cells designed to receive user input, cells to process data (using spreadsheet formulas), and cells intended to display the results. As an alternative to the code coverage of traditional programming paradigms, we compute so-called *cones* (data dependencies of each cell).

Definition 12 (*The function CONE*) Given a spreadsheet Π and a cell $c \in \Pi$, then we define the function CONE recursively as follows:

$$\text{CONE}(c) = c \cup \bigcup_{c' \in \rho(c)} \text{CONE}(c')$$

Example 2 The cones for the output cells the spreadsheet from Fig. 1 are

$$\begin{aligned}
\text{CONE}(F2) &= \{B2, D2, E2, F2\}\\
\text{CONE}(D4) &= \{B2, D2, B3, C3, D3, D4\}\\
\text{CONE}(B4) &= \{B2, B3, B4\}\\
\text{CONE}(C4) &= \{C2, C3, C4\}\\
\text{CONE}(F3) &= \{B3, C3, D3, E3, F3\}.
\end{aligned}$$

From the cones, the hit-spectra matrix can be generated (each column of the matrix has the dependencies of one output cell). The error vector represents the correctness of the output cells. The correctness of the output cells is determined either by the user (i.e. by guessing what values are wrong), by comparing the results of the current spreadsheet Π with another spreadsheet considered correct, or by applying techniques to automatically detect "bad smells" (Hermans et al. 2012a). For every cell, a dichotomy matrix is then created (see Table 1). One dimension of this matrix is related to the amount of cones the cell is involved in, and the other is the outcome of the output cells (correct / incorrect).

Example 3 If we take the faulty spreadsheet from Fig. 1b and the test case from Example 1, we have two erroneous output cells (F2 and D4) and three correct output cells (B4, C4 and F3). With this information and the previously computed cones,

Table 1 Dichotomy matrix of a cell

	Not involved	Involved
Correct outcome	n_{00}	n_{10}
Incorrect outcome	n_{01}	n_{11}

Table 2 The hit-spectra matrix and the values of the dichotomy matrices for the running example

Cell	F2	D4	B4	C4	F3	n_{11}	n_{10}	n_{00}	n_{01}
B2	•	•	•			2	1	2	0
B3		•	•		•	1	2	1	1
B4			•			0	1	2	2
C2				•		0	1	2	2
C3		•		•	•	1	2	1	1
C4				•		0	1	2	2
D2	•	•				2	0	3	0
D3		•			•	1	1	2	1
D4		•				1	0	3	1
E2	•					1	0	3	1
E3					•	0	1	2	2
F2	•					1	0	3	1
F3					•	0	1	2	2
Error	•	•							

we are able to build the hit-spectra matrix and the dichotomy matrices illustrated in Table 2.

SFL uses similarity coefficients (or association measures) to estimate the likelihood of a given software component being faulty. Similarity coefficients compute the relationship between each column of the matrix (representing a system component) and the error vector. Similarity coefficients and the failure probability of the corresponding system component are directly related (Abreu et al. 2006). The coefficients are used to create rankings of system components (Janssen et al. 2009) or to create interactive visualizations of the SUT, revealing the most suspicious parts of the application's source code (Campos et al. 2012).

In order to assess what are the best coefficients for ranking suspicious spreadsheet cells, we have considered the 42 similarity coefficients studied in the work of Lucia et al. (2013). This work focuses on evaluating several coefficients in the context of fault localization in software programs. The considered coefficients are: Accuracy (Geng and Hamilton 2006; Ohsaki et al. 2004), added value (Tan et al. 2002; Anderberg 1973), Certainty Factor (Shortliffe and Buchanan 1975), Collective Strength (Aggarwal et al. 1998), Confidence (Aggarwal et al. 1994), Conviction (Brin et al. 1997), Coverage (Geng and Hamilton 2006; Ohsaki et al. 2004), Example and Counterexample Rate (Geng and Hamilton 2006), Gini Index (Lucia et al. 2013), Goodman and Kruskal (Geng and Hamilton 2006; Goodman and Kruskal 1954; Ohsaki et al. 2004; Tan et al. 2002), Information Gain (Cheng et al. 2009), Interest (Brin et al. 1997), Interestingness Weighting Dependency (Geng and Hamilton 2006; Ohsaki et al. 2004), J-Measure (Smyth and Goodman 1992), Jaccard (Hand et al. 2001), Kappa (Cohen 1960), Klosgen (Klösgen 1996), Clark and Boswell's Laplace Accuracy (Boswell 1991), Least Contradiction (Geng and Hamilton 2006), Leverage (Geng and Hamilton 2006; Ohsaki et al. 2004), Loevinger (Geng and Hamilton 2006), Normalized Mutual Information (Tan et al. 2002; Geng and Hamilton 2006; Ohsaki et al. 2004), Ochiai (or Cosine) (Abreu et al. 2007), Ochiai II (Ochiai 1957), Odd Multiplier (Geng and Hamilton 2006), Odds Ratio (Agresti 1996), One-Way Support (Geng and Hamilton 2006; Ohsaki et al. 2004), Piatetsky-Shapiro's Leverage (Piatetsky-Shapiro 1991), relative risk (Geng and Hamilton 2006; Ohsaki et al. 2004), Rogers and Tanimoto (Rogers and Tanimoto 1960), Sebag-Schoenauer (Geng and Hamilton 2006), Simple-Matching (Sokal and Michener 1958), Sorensen-Dice (Dice 1945; Lucia et al. 2013), Support (Aggarwal et al. 1994), Tarantula (Jones and Harrold 2005), Two-Way Support (Geng and Hamilton 2006; Ohsaki et al. 2004), Two-Way Support Variation (Geng and Hamilton 2006; Ohsaki et al. 2004), Yule's Q (Lucia et al. 2013), Yule's Y (Lucia et al. 2013), Zhang (Geng and Hamilton 2006), and ϕ-coefficient (Healey 1993). Apart from Tarantula, the similarity coefficients have not been derived in the context of program's fault localization. Instead, they come from the statistics and data mining communities. It is possible to filter some of the coefficients out because their poor performance a-priory. However, for the sake of completeness we considered all of them in our experiments.

The mathematical formulae for computing each coefficient are shown in Table 3. In this context, the variables A and B are the two dimensions of our dichotomy matrix, corresponding to involvement and outcome, respectively. The table also uses standard notation from probability and statistics, namely: $P(A)$ is the probability of A, $P(\overline{A})$ is

Table 3 Mathematical formulae for the similarity coefficients used in this work

#	Coefficient	Formula
C_1	Accuracy (Geng and Hamilton 2006; Ohsaki et al. 2004)	$P(AB) + P(\overline{AB})$
C_2	Added Value (Tan et al. 2002)	$\max(P(B\|A) - P(B), P(A\|B) - P(A))$
C_3	Anderberg (Anderberg 1973)	$\frac{P(AB)}{P(AB)+2(P(AB)+P(A\overline{B}))}$
C_4	Certainty Factor (Shortliffe and Buchanan 1975)	$\max(\frac{P(B\|A)-P(B)}{1-P(B)}, \frac{P(A\|B)-P(A)}{1-P(A)})$
C_5	Collective Strength (Aggarwal et al. 1998)	$\frac{P(AB)+P(\overline{AB})}{P(A)P(B)+P(\overline{A})P(\overline{B})} \times \frac{1-P(A)P(B)-P(\overline{A})P(\overline{B})}{1-P(AB)-P(\overline{AB})}$
C_6	Confidence (Aggarwal et al. 1994)	$\max(P(B\|A), P(A\|B))$
C_7	Conviction (Brin et al. 1997)	$\max(\frac{P(A)P(\overline{B})}{P(A\overline{B})}, \frac{P(B)P(\overline{A})}{P(B\overline{A})})$
C_8	Coverage (Geng and Hamilton 2006; Ohsaki et al. 2004)	$P(A)$
C_9	Example and Counterexample (Geng and Hamilton 2006)	$1 - \frac{P(A\overline{B})}{P(AB)}$

Table 3 continued

#	Coefficient	Formula
C_{10}	Gini Index (Lucia et al. 2013)	$\max(P(A)[P(B\mid A)^2 + P(\overline{B}\mid A)^2] + P(\overline{A})[P(B\mid \overline{A})^2 + P(\overline{B}\mid \overline{A})^2] - P(B)^2 - P(\overline{B})^2,$ $P(B)[P(A\mid B)^2 + P(\overline{A}\mid B)^2] + P(\overline{B})[P(A\mid \overline{B})^2 + P(\overline{A}\mid \overline{B})^2] - P(A)^2 - P(\overline{A})^2)$
C_{11}	Goodman and Kruskal (Geng and Hamilton 2006; Goodman and Kruskal 1954; Ohsaki et al. 2004; Tan et al. 2002)	$\frac{\sum_i \max_j P(A_i B_j) + \sum_j \max_i P(A_i B_j) - \max_i P(A_i) - \max_j P(B_j)}{2 - \max_i P(A_i) - \max_j P(B_j)}$
C_{12}	Information Gain (Cheng et al. 2009)	$(-P(B)\log P(B) - P(\overline{B})\log P(\overline{B}))-$ $(P(A) \times (-P(B\mid A)\log P(B\mid A)) - P(\overline{B}\mid A)\log P(\overline{B}\mid A)-$ $P(\overline{A}) \times (-P(B\mid \overline{A})\log P(B\mid \overline{A})) - P(\overline{B}\mid \overline{A})\log P(\overline{B}\mid \overline{A})))$
C_{13}	Interest (Brin et al. 1997)	$\frac{P(AB)}{P(A)P(B)}$
C_{14}	Interestingness Weighting Dependency (Geng and Hamilton 2006; Ohsaki et al. 2004)	$\left((\frac{P(AB)}{P(A)P(B)})^k - 1\right) P(AB)^m$ where k,m are coefficients of dependency and generality respectively weighting the relative importance of the two factors.
C_{15}	J-Measure (Smyth and Goodman 1992)	$\max(P(AB)\log(\frac{P(B\mid A)}{P(B)}) + P(A\overline{B})\log(\frac{P(\overline{B}\mid A)}{P(\overline{B})}),$ $P(AB)\log(\frac{P(A\mid B)}{P(A)}) + P(\overline{A}B)\log(\frac{P(\overline{A}\mid B)}{P(\overline{A})}))$
C_{16}	Jaccard (Hand et al. 2001)	$\frac{P(AB)}{P(A)-P(B)-P(AB)}$
C_{17}	Kappa (Cohen 1960)	$\frac{P(AB)+P(\overline{AB})-P(A)P(B)-P(\overline{A})P(\overline{B})}{1-P(A)P(B)-P(\overline{A})P(\overline{B})}$
C_{18}	Klosgen (Klösgen 1996)	$\sqrt{P(AB)}\max(P(B\mid A) - (B), P(A\mid B) - P(A))$
C_{19}	Laplace (Boswell 1991)	$\max(\frac{P(AB)+1}{P(A)+2}, \frac{P(AB)+1}{P(B)+2})$
C_{20}	Least Contradiction (Geng and Hamilton 2006)	$\frac{P(AB)-P(A\overline{B})}{P(B)}$
C_{21}	Leverage (Geng and Hamilton 2006; Ohsaki et al. 2004)	$P(B\mid A) - P(A)P(B)$
C_{22}	Loevinger (Geng and Hamilton 2006)	$1 - \frac{P(A)P(\overline{B})}{P(A\overline{B})}$
C_{23}	Normalized Mutual Information (Tan et al. 2002)	$\frac{\sum_i \sum_j P(A_i B_j)\log_2 \frac{P(A_i B_j)}{P(A_i)P(B_j)}}{-\sum_i P(A_i)\log_2 P(A_i)}$
C_{24}	Ochiai (Abreu et al. 2007)	$\frac{P(AB)}{\sqrt{P(A)P(B)}}$
C_{25}	Ochiai II (Ochiai 1957)	$\frac{P(AB)+P(\overline{AB})}{\sqrt{P(AB)+P(\overline{A}B)(P(AB)+P(A\overline{B}))(P(\overline{AB})+P(\overline{A}B))(P(\overline{AB})+P(A\overline{B}))}}$
C_{26}	Odd Multiplier (Geng and Hamilton 2006)	$\frac{P(AB)P(\overline{B})}{P(B)P(A\overline{B})}$
C_{27}	Odds Ratio (Agresti 1996)	$\frac{P(AB)P(\overline{AB})}{P(A\overline{B})P(\overline{A}B)}$

Table 3 continued

#	Coefficient	Formula
C_{28}	One-Way Support (Geng and Hamilton 2006; Ohsaki et al. 2004)	$P(B\|A)\log_2 \frac{P(AB)}{P(A)P(B)}$
C_{29}	Piatetsky-Shapiro (Piatetsky-Shapiro 1991)	$P(AB) - P(A)P(B)$
C_{30}	Relative Risk (Geng and Hamilton 2006; Ohsaki et al. 2004)	$\frac{P(B\|A)}{P(B\|\overline{A})}$
C_{31}	Rogers and Tanimoto (Rogers and Tanimoto 1960)	$\frac{P(AB)+P(\overline{AB})}{P(AB)+P(\overline{AB})+2(P(\overline{A}B)+P(A\overline{B}))}$
C_{32}	Sebag-Schoenauer (Geng and Hamilton 2006)	$\frac{P(AB)}{P(A\overline{B})}$
C_{33}	Simple-Matching (Sokal and Michener 1958)	$P(AB) + P(\overline{AB})$
C_{34}	Sorensen-Dice (Dice 1945; Lucia et al. 2013)	$\frac{2P(AB)}{2P(AB)+P(\overline{A}B)+P(A\overline{B})}$
C_{35}	Support (Aggarwal et al. 1994)	$P(AB)$
C_{36}	Tarantula (Jones and Harrold 2005)	$\frac{\frac{P(AB)}{P(B)}}{\frac{P(A\overline{B})}{P(\overline{B})}+\frac{P(AB)}{P(B)}}$
C_{37}	Two-Way Support (Geng and Hamilton 2006; Ohsaki et al. 2004)	$P(AB)\log_2 \frac{P(AB)}{P(A)P(B)}$
C_{38}	Two-Way Support Variation (Geng and Hamilton 2006; Ohsaki et al. 2004)	$P(AB)\log_2 \frac{P(AB)}{P(A)P(B)} + P(\overline{A}B)\log_2 \frac{P(\overline{A}B)}{P(\overline{A})P(B)} +$ $P(A\overline{B})\log_2 \frac{P(A\overline{B})}{P(A)P(\overline{B})} + P(\overline{AB})\log_2 \frac{P(\overline{AB})}{P(\overline{A})P(\overline{B})}$
C_{39}	Yule's Q (Lucia et al. 2013)	$\frac{P(AB)P(\overline{AB})-P(A\overline{B}P(\overline{A}B))}{P(AB)P(\overline{AB})+P(A\overline{B}P(\overline{A}B)}$
C_{40}	Yule's Y (Lucia et al. 2013)	$\frac{\sqrt{P(AB)P(\overline{AB})}-\sqrt{P(A\overline{B}P(\overline{A}B))}}{\sqrt{P(AB)P(\overline{AB})}+\sqrt{P(A\overline{B}P(\overline{A}B)}}$
C_{41}	Zhang (Geng and Hamilton 2006)	$\frac{P(AB)-P(A)P(B)}{\max(P(AB)P(\overline{B}),P(B)P(A\overline{B}))}$
C_{42}	ϕ-coefficient (Healey 1993)	$\frac{P(AB)-P(A)P(B)}{\sqrt{P(A)P(B)(1-P(A))(1-P(B))}}$

the probability of *not* A, $P(AB)$ is the joint probability of A and B, and $P(A|B)$ is the conditional probability of A given B. In our dichotomy matrix, we gather frequencies instead of probabilities, but these can be easily substituted during actual computation (see Table 4).

Example 4 Let's continue with the debugging of our running example: We use the information of the dichotomy matrices from Table 2 to compute the similarity coefficients. In this example, we choose the Ochiai similarity coefficient (C_{24}). For each cell, we compute the similarity coefficients by inserting the information of the dichotomy matrix into the formula, e.g., for cell B2:

Table 4 Definitions of the probabilities used by the coefficients in the context of this paper

$P(A) = \frac{n_{11}+n_{10}}{n_{11}+n_{10}+n_{01}+n_{00}}$	$P(\overline{A}) = \frac{n_{01}+n_{00}}{n_{11}+n_{10}+n_{01}+n_{00}}$
$P(B) = \frac{n_{11}+n_{01}}{n_{11}+n_{10}+n_{01}+n_{00}}$	$P(\overline{B}) = \frac{n_{10}+n_{00}}{n_{11}+n_{10}+n_{01}+n_{00}}$
$P(AB) = \frac{n_{11}}{n_{11}+n_{10}+n_{01}+n_{00}}$	$P(\overline{A}B) = \frac{n_{01}}{n_{11}+n_{10}+n_{01}+n_{00}}$
$P(A\overline{B}) = \frac{n_{10}}{n_{11}+n_{10}+n_{01}+n_{00}}$	$P(\overline{AB}) = \frac{n_{00}}{n_{11}+n_{10}+n_{01}+n_{00}}$
$P(A\|B) = \frac{P(AB)}{P(B)}$	$P(B\|A) = \frac{P(AB)}{P(A)}$

Table 5 The Ochiai similarity coefficients and the subsequent ranking for the running example

Cell	Coefficient	Ranking
B2	0.82	2
B3	0.41	7
B4	0.00	-
C2	0.00	-
C3	0.41	7
C4	0.00	-
D2	1.00	1
D3	0.50	6
D4	0.71	3
E2	0.71	3
E3	0.00	-
F2	0.71	3
F3	0.00	-

$$C_{24}(B2) = \frac{P(AB)}{\sqrt{P(A)P(B)}} = \frac{n_{11}}{\sqrt{(n_{11}+n_{10})(n_{11}+n_{01})}} = \frac{2}{\sqrt{(2+1)(2+0)}} = 0.82$$

Similar, we continue with the other cells and obtain the coefficients illustrated in Table 5. Afterwards, we rank the cells in descending order of the coefficient. The faulty cell D2 is the highest ranked.

5 Empirical evaluation

To assess the effectiveness of the various similarity coefficients in the context of SFL in spreadsheets, we are evaluating the ranking of faulty cells computed using SFL with each similarity coefficient presented in the previous section. First of all, we are going the explain the experimental setup. We compare the previously described similarity coefficients by means of two corpora: (1) a modified version of the EUSES spreadsheet corpus (Fisher and Rothermel 2005; Hofer et al. 2013), and (2) a spreadsheet version of the ISCAS85 circuits (Brglez and Fujiwara 1985; Nica et al. 2013). All spreadsheets contained in these corpora a feasible, i.e they do not contain any circular references.

Table 6 gives an overview of the characteristic of these corpora. The modified EUSES spreadsheet corpus comes with 703 spreadsheets. Each of these spreadsheets contains a single fault that was randomly created by using some of the spreadsheet

Table 6 Characteristics of the used corpora

Corpus	Euses	Iscas85
Number of spreadsheets	703	100
Avg. number formulas	404.0 (6 to 10,316)	2,952.8 (396 to 7,345)
Avg. number incorrect output cells	3.1 (1 to 72)	1.6 (1 to 6)
Avg. number correct output cells	80.8 (0 to 2,962)	45.4 (1 to 122)
Avg. number coincidental correct output cells	0.6 (0 to 43)	5.7 (0 to 39)

mutation operators of Abraham and Erwig (2009). We refer the interested reader to Hofer et al. (2013) for more details about this spreadsheet corpus. ISCAS85 refers to a collection of different circuits (Brglez and Fujiwara 1985). Nica et al. (2013) created faulty versions of these circuits by randomly choosing a gate and changing the gate's Boolean function. In total, they created 300 faulty versions containing single-, double-, and triple-faults. In addition, they created a test case for each version, revealing the fault. For the empirical evaluation, we used a spreadsheet version of their faulty circuits (single-faults only) and test cases. In Table 6, we additionally indicate the average number of incorrect and correct output cells. Furthermore, we indicate the average number of output cells that are influenced by the faulty cell but coincidentally compute the correct value. Coincidental correctness (Wang et al. 2009) happens more often for the ISCAS85 spreadsheets than for the EUSES spreadsheets, because of the cell domains: The ISCAS85 spreadsheets compute the outcome of circuits, i.e. Boolean values, while the EUSES spreadsheets mainly compute Real numbers.

We run our fault localization approach to yield a list of likely faulty cells, sorted by their suspiciousness (*i.e.*, the value of the similarity coefficient that is currently being used). Since cells with a higher suspiciousness are more likely to be faulty, we assume that users start inspecting the highest ranked cells first, and go further down the list until the fault is reached. In this experiment, for each mutated subject of our corpora, we know the location of the injected fault. Therefore, we can compute a metric that measures the effort required by the user to pinpoint the fault ($f*$):

$$E_{best} = \frac{|\{i|C(i) > C(f*)\}| + 1}{|M|}, i \in M$$

where $C(i)$ is the similarity coefficient value for cell i and M is the set of all cells in the spreadsheet. This metric tells us the percentage of cells that need to be inspected until the bug is found. You may notice that we labeled this metric as E_{best}. In fact, it portrays the best case scenario concerning cell suspiciousness ties. A tie (Xu et al. 2011) is a set of statements (or cells in our case) with the same suspiciousness. All statements of the tie have consequently the same ranking position. The tie that contains a faulty statement is called critical. In the best case scenario, the faulty cell is always inspected first among the cells that share the same suspiciousness value. In order to avoid such premise, we consider two other metrics. One that takes into account all elements whose suspiciousness is the same as the defect's (*i.e.*, the worst case scenario):

$$E_{worst} = \frac{|\{i|C(i) \geq C(f*)\}|}{|M|}, i \in M$$

And other that assumes that, on average, the user will have to inspect half of the components in the critical tie to reach the fault:

$$E_{average} = \frac{|\{i|C(i) > C(f*)\}| + 1}{|M|} + \frac{|\{i|C(i) = C(f*)\}|}{2|M|}, i \in M$$

Considering the best case, worst case and average case scenarios for each spreadsheet will enable us to have a more accurate understanding of the impact that each similarity coefficient has in the debugging accuracy of SFL.

The overall mean best case, average case and worst case effort for each similarity coefficient, along with the standard deviation, are shown in Table 7. The smallest effort measure for the worst case scenario was achieved by Ochiai (C_{24}), with a value of 4.96 % for the EUSES corpus and 4.57 % for the ISCAS85 corpus. This means that, in the worst case, users have to inspect on average less than 5 % of the spreadsheet's cells to locate the bug using Ochiai as the coefficient for SFL. Ochiai also shows the lowest standard deviation (σ) in the worst case scenario, meaning that the results for this coefficient are closer to the mean. Other coefficients with similar effort when compared with Ochiai are Jaccard (C_{16}) and Sorensen-Dice (C_{34}). Several coefficients have a smaller best case effort than Ochiai, but a higher worst case effort. This means that these coefficients have a larger critical tie, e.g., Certainty Factor (C_4), Confidence (C_6), Laplace (C_{19}), Support (C_{35}), Yule's Q (C_{39}) and Yule's Y (C_{40}).

As there are coefficients that behave very similar, we have decided to perform a clustering analysis. k-means clustering was used to partition the coefficients into k different clusters (MacQueen 1967). Since we do not know *a priori* how many clusters there are in our dataset, we employed an incremental approach. We started by computing memberships for two clusters ($k = 2$ in the k-means algorithm). Afterwards, we performed statistical tests to ensure that every element from the same cluster is not statistically significantly different and that elements from different clusters were significantly different. The statistical test used is the Wilcoxon signed-rank (Wilcoxon 1945) with 97 % confidence. The reason we use Wilcoxon instead of, e.g., Student's t-test is because it does not assume that our data is normally distributed. If the statistical criteria was not met, we increase the number of clusters in k-means (by increasing the value of k). This way, coefficients within the same cluster are statistically significantly different from every coefficient that does not belong to that cluster.

The resulting clusters for the best, average and worst case scenarios are shown in Figs. 2a, b and c, respectively. Each node corresponds to a cluster, and edges indicate relationships between clusters such that A $\longrightarrow$ B means "cluster A requires less effort to diagnose than cluster B". Note that the number of clusters (and their elements) is not the same in every scenario—there are ten clusters for best case, nine clusters for average case and six clusters for worst case. This happens because differences in best case (and average case) behavior are more diverse than in the worst case. Furthermore, some clusters require similar effort, but they are statistically different (i.e. they generate different diagnostic rankings). These clusters appear side by side in the figure.

Table 7 Mean effort for each similarity coefficient

#	EUSES			ISCAS		
	E_{best} (σ)	$E_{average}$ (σ)	E_{worst} (σ)	E_{best} (σ)	$E_{average}$ (σ)	E_{worst} (σ)
C_1	0.0078 (0.0307)	0.0302 (0.0573)	0.0526 (0.0949)	0.0154 (0.0370)	0.0330 (0.0564)	0.0504 (0.0817)
C_2	0.0076 (0.0267)	0.0332 (0.0523)	0.0587 (0.0918)	0.0171 (0.0396)	0.0342 (0.0591)	0.0512 (0.0842)
C_3	0.0083 (0.0305)	0.0332 (0.0542)	0.0580 (0.0920)	0.0174 (0.0397)	0.0352 (0.0613)	0.0529 (0.0880)
C_4	0.0041 (0.0151)	0.0334 (0.0492)	0.0626 (0.0933)	0.0074 (0.0320)	0.0581 (0.0661)	0.1087 (0.1168)
C_5	0.0186 (0.0495)	0.0404 (0.0638)	0.0621 (0.0948)	0.0808 (0.1092)	0.0978 (0.1093)	0.1146 (0.1167)
C_6	0.0036 (0.0103)	0.0332 (0.0485)	0.0627 (0.0934)	0.0003 (0.0003)	0.0812 (0.0625)	0.1619 (0.1248)
C_7	0.0163 (0.0470)	0.0425 (0.0673)	0.0685 (0.1027)	0.0805 (0.1079)	0.0976 (0.1085)	0.1145 (0.1161)
C_8	0.0111 (0.0335)	0.0344 (0.0553)	0.0575 (0.0913)	0.1151 (0.1186)	0.1338 (0.1167)	0.1524 (0.1222)
C_9	0.0099 (0.0375)	0.0368 (0.0597)	0.0636 (0.0976)	0.0206 (0.0444)	0.0386 (0.0652)	0.0565 (0.0913)
C_{10}	0.0068 (0.0251)	0.0287 (0.0515)	0.0505 (0.0893)	0.0195 (0.0549)	0.0365 (0.0705)	0.0534 (0.0927)
C_{11}	0.0068 (0.0262)	0.0294 (0.0546)	0.0520 (0.0934)	0.0214 (0.0472)	0.0418 (0.0659)	0.0621 (0.0933)
C_{12}	0.0175 (0.0533)	0.0394 (0.0708)	0.0612 (0.1024)	0.0369 (0.0714)	0.0539 (0.0837)	0.0708 (0.1029)
C_{13}	0.0089 (0.0291)	0.0338 (0.0539)	0.0585 (0.0921)	0.0174 (0.0397)	0.0351 (0.0614)	0.0528 (0.0881)
C_{14}	0.0072 (0.0263)	0.0291 (0.0521)	0.0509 (0.0897)	0.0167 (0.0397)	0.0346 (0.0599)	0.0523 (0.0863)
C_{15}	0.0096 (0.0369)	0.0315 (0.0578)	0.0533 (0.0930)	0.0557 (0.0895)	0.0727 (0.0987)	0.0896 (0.1146)
C_{16}	0.0061 (0.0211)	0.0280 (0.0498)	0.0497 (0.0884)	0.0121 (0.0257)	0.0290 (0.0485)	0.0458 (0.0751)
C_{17}	0.0072 (0.0262)	0.0290 (0.0521)	0.0508 (0.0897)	0.0153 (0.0372)	0.0323 (0.0564)	0.0492 (0.0815)
C_{18}	0.0070 (0.0261)	0.0289 (0.0520)	0.0507 (0.0896)	0.0158 (0.0397)	0.0336 (0.0601)	0.0514 (0.0866)
C_{19}	0.0044 (0.0146)	0.0292 (0.0489)	0.0539 (0.0910)	0.0003 (0.0003)	0.0779 (0.0612)	0.1553 (0.1223)
C_{20}	0.0080 (0.0316)	0.0302 (0.0562)	0.0522 (0.0931)	0.0158 (0.0371)	0.0331 (0.0564)	0.0503 (0.0817)

Table 7 continued

#	EUSES			ISCAS		
	E_{best} (σ)	$E_{average}$ (σ)	E_{worst} (σ)	E_{best} (σ)	$E_{average}$ (σ)	E_{worst} (σ)
C_{21}	0.0160 (0.0486)	0.0378 (0.0632)	0.0595 (0.0944)	0.0211 (0.0554)	0.0381 (0.0721)	0.0550 (0.0948)
C_{22}	0.0181 (0.0560)	0.0445 (0.0743)	0.0707 (0.1078)	0.0758 (0.1260)	0.0937 (0.1323)	0.1115 (0.1446)
C_{23}	0.0068 (0.0231)	0.0287 (0.0501)	0.0505 (0.0882)	0.0207 (0.0548)	0.0377 (0.0702)	0.0546 (0.0922)
C_{24}	0.0060 (0.0210)	0.0279 (0.0498)	0.0496 (0.0884)	0.0119 (0.0255)	0.0289 (0.0484)	0.0457 (0.0750)
C_{25}	0.0225 (0.0590)	0.0445 (0.0737)	0.0663 (0.1035)	0.0286 (0.0622)	0.0462 (0.0773)	0.0636 (0.0990)
C_{26}	0.0134 (0.0410)	0.0402 (0.0640)	0.0669 (0.1014)	0.0766 (0.1069)	0.0946 (0.1073)	0.1125 (0.1156)
C_{27}	0.0186 (0.0554)	0.0502 (0.0799)	0.0816 (0.1192)	0.0192 (0.0547)	0.1036 (0.0870)	0.1879 (0.1428)
C_{28}	0.0079 (0.0273)	0.0333 (0.0534)	0.0586 (0.0923)	0.0174 (0.0397)	0.0362 (0.0624)	0.0548 (0.0909)
C_{29}	0.0071 (0.0262)	0.0290 (0.0520)	0.0507 (0.0897)	0.0141 (0.0370)	0.0312 (0.0567)	0.0481 (0.0819)
C_{30}	0.0239 (0.0638)	0.0506 (0.0803)	0.0772 (0.1135)	0.0293 (0.0719)	0.1087 (0.0949)	0.1879 (0.1428)
C_{31}	0.0078 (0.0307)	0.0300 (0.0558)	0.0521 (0.0930)	0.0154 (0.0370)	0.0330 (0.0564)	0.0504 (0.0817)
C_{32}	0.0131 (0.0401)	0.0400 (0.0636)	0.0669 (0.1014)	0.0766 (0.1069)	0.0946 (0.1073)	0.1125 (0.1156)
C_{33}	0.0078 (0.0307)	0.0302 (0.0573)	0.0526 (0.0949)	0.0154 (0.0370)	0.0330 (0.0564)	0.0504 (0.0817)
C_{34}	0.0061 (0.0211)	0.0280 (0.0498)	0.0497 (0.0884)	0.0121 (0.0257)	0.0290 (0.0485)	0.0458 (0.0751)
C_{35}	0.0042 (0.0137)	0.0292 (0.0489)	0.0540 (0.0914)	0.0003 (0.0003)	0.0779 (0.0612)	0.1553 (0.1223)
C_{36}	0.0077 (0.0269)	0.0335 (0.0532)	0.0592 (0.0928)	0.0174 (0.0397)	0.0352 (0.0613)	0.0529 (0.0880)
C_{37}	0.0071 (0.0264)	0.0290 (0.0522)	0.0508 (0.0898)	0.0142 (0.0370)	0.0320 (0.0574)	0.0498 (0.0840)
C_{38}	0.0068 (0.0250)	0.0287 (0.0515)	0.0505 (0.0893)	0.0195 (0.0548)	0.0365 (0.0704)	0.0534 (0.0926)
C_{39}	0.0041 (0.0151)	0.0334 (0.0491)	0.0626 (0.0933)	0.0074 (0.0320)	0.0581 (0.0661)	0.1087 (0.1168)
C_{40}	0.0041 (0.0151)	0.0334 (0.0492)	0.0626 (0.0933)	0.0074 (0.0320)	0.0581 (0.0661)	0.1087 (0.1168)
C_{41}	0.0094 (0.0329)	0.0340 (0.0550)	0.0586 (0.0921)	0.0174 (0.0397)	0.0353 (0.0619)	0.0531 (0.0891)
C_{42}	0.0070 (0.0250)	0.0289 (0.0514)	0.0507 (0.0893)	0.0151 (0.0371)	0.0321 (0.0565)	0.0490 (0.0817)

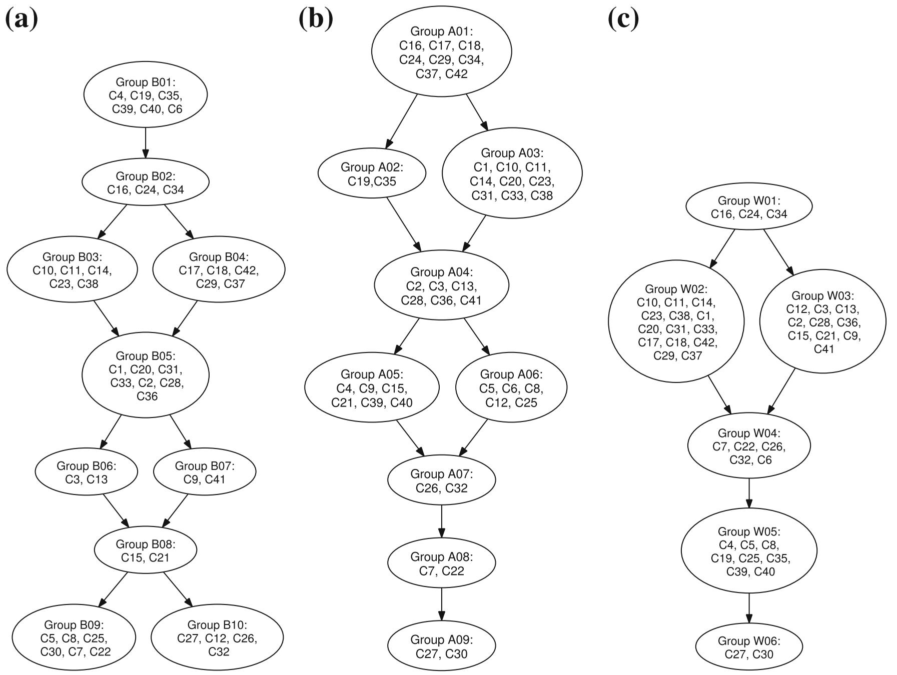

(a)
Group B01: C4, C19, C35, C39, C40, C6
Group B02: C16, C24, C34
Group B03: C10, C11, C14, C23, C38
Group B04: C17, C18, C42, C29, C37
Group B05: C1, C20, C31, C33, C2, C28, C36
Group B06: C3, C13
Group B07: C9, C41
Group B08: C15, C21
Group B09: C5, C8, C25, C30, C7, C22
Group B10: C27, C12, C26, C32
(b)
Group A01: C16, C17, C18, C24, C29, C34, C37, C42
Group A02: C19,C35
Group A03: C1, C10, C11, C14, C20, C23, C31, C33, C38
Group A04: C2, C3, C13, C28, C36, C41
Group A05: C4, C9, C15, C21, C39, C40
Group A06: C5, C6, C8, C12, C25
Group A07: C26, C32
Group A08: C7, C22
Group A09: C27, C30
(c)
Group W01: C16, C24, C34
Group W02: C10, C11, C14, C23, C38, C1, C20, C31, C33, C17, C18, C42, C29, C37
Group W03: C12, C3, C13, C2, C28, C36, C15, C21, C9, C41
Group W04: C7, C22, C26, C32, C6
Group W05: C4, C5, C8, C19, C25, C35, C39, C40
Group W06: C27, C30

Fig. 2 Similarity coefficients clustered by their required effort. Edges are directed, such that A ⟶ B should read "cluster A requires less effort to diagnose than cluster B". **a** Best case, **b** average case, **c** worst case

In order to better visualize the accuracy of each cluster, we also provide Fig. 3, 4 and 5, that graphically compare the fault localization capabilities of each scenario for both studied corpora. The x-axis represents the percentage of formula cells that is investigated. The y-axis represents the percentage of faults that are localized within that amount of cells.

For the worst case, the cluster W01 shows best efficiency and comprises Jaccard (C_{16}), Ochiai (C_{24}) and Sorensen-Dice (C_{34}). This not only agrees with our analysis of Table 7, but also evidences the fact that these three similarity coefficients are statistically similar. For the average case, the cluster A01 is comprised of the similarity coefficient with the lowest effort: Jaccard (C_{16}), Kappa (C_{17}), Klosgen (C_{18}), Ochiai (C_{24}), Piatetsky-Shapiro (C_{29}), Sorensen-Dice (C_{34}), Two-Way Support (C_{37}) and ϕ-coefficient (C_{42}).

For the best case, the cluster B01 has a lowest effort and comprises Certainty Factor (C_4), Confidence (C_6), Laplace (C_{19}), Support (C_{35}), Yule's Q (C_{39}) and Yule's Y (C_{40}). However, if we look at these coefficient's memberships in the worst case cluster, we see that they perform worse than many others. The reason for this phenomenon is that these coefficients generate a large critical tie, and therefore their real accuracy is much lower. In conclusion, if one is to diagnose spreadsheets, either Jaccard (C_{16}), Ochiai (C_{24}) or Sorensen-Dice (C_{34}) should be used, since these coefficients show the best accuracy results, and they are statistically similar among themselves.

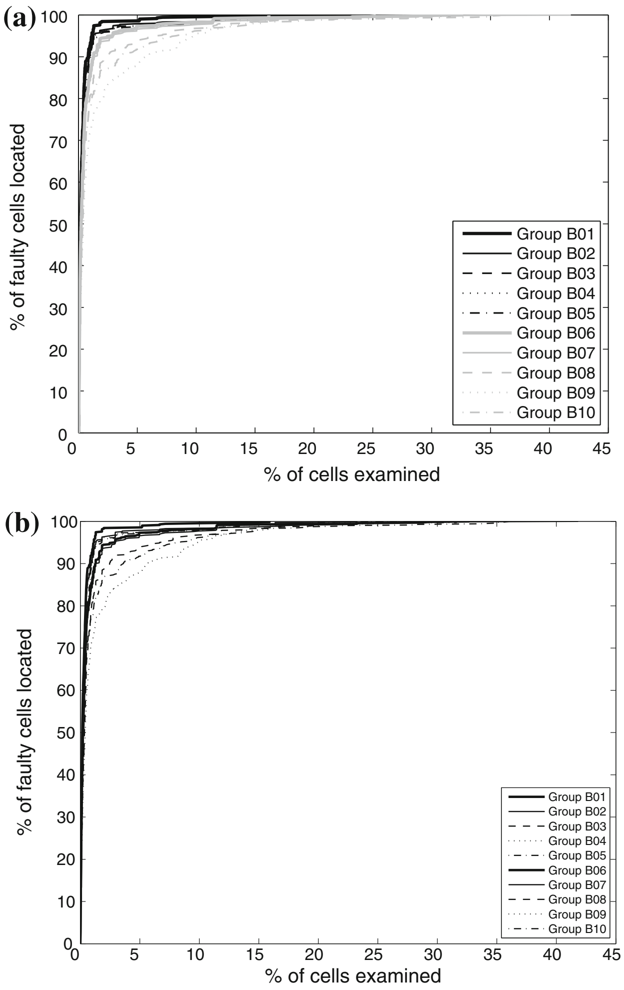


Fig. 3 Comparison of the best case scenario clusters in terms of effort required to diagnose. **a** Iscas85 Corpus, **b** Euses Corpus

In the above given results, we have assumed that the user has full knowledge about the correctness/incorrectness of the output cells. In practice, the user does not have that information or the user does not want to indicate so much data. Is it possible to localize faulty cells if the user only indicates a few correct and incorrect output cells? More general, what is the impact of the amount of observations the user indicates on the diagnostic report? Abreu et al. (2009) dealt with this research question in the context imperative programs and ascertain the following:

- The more incorrect output identified, the better the diagnostic report;
- Correct outputs have two effects in the final results: (i) negative impact if they cover the faulty part (i.e. if they execute a faulty statement in case of imperative programs or if their cone includes any faulty cell in case of spreadsheets); (ii) positive impact on the diagnostic report if they to not cover any faulty part.

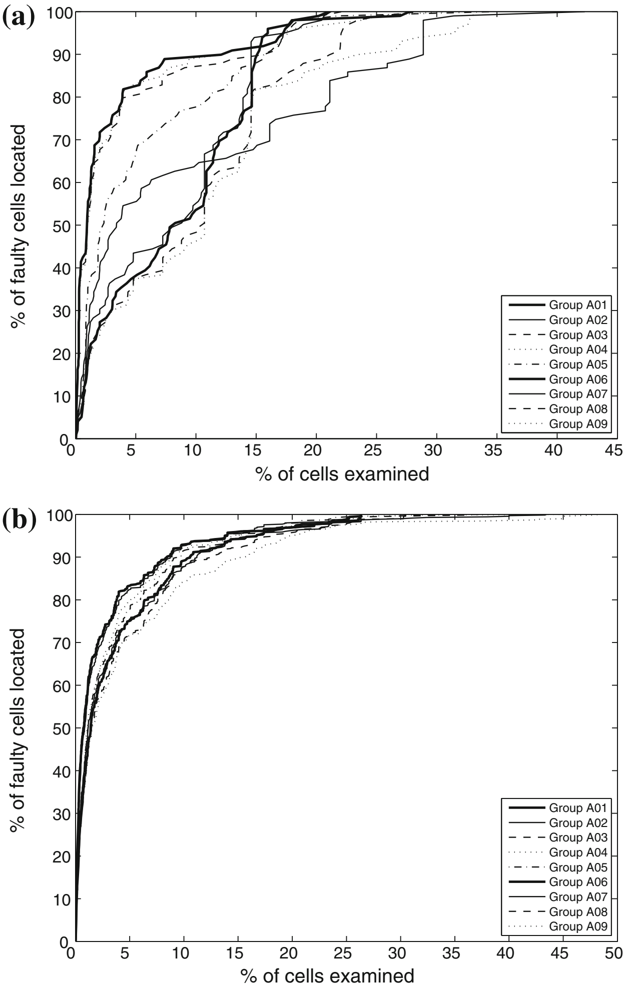


Fig. 4 Comparison of the average case scenario clusters in terms of effort required to diagnose. **a** ISCAS85 Corpus, **b** EUSES Corpus

In order to demonstrate the influence of the number of correct and incorrect output cells on the ranking result, we have performed an empirical evaluation using spreadsheets from both copora. Figure 6 shows the results for the ISCAS85 spreadsheets c5315_BOOL_tc1_71_1Fault and c7552_BOOL_tc1_96_1Fault. Figure 7 shows the results for the EUSES spreadsheets WorldPopPlay_1FAULTS_FAULTVERSION1 and my_financial_model_1FAULTS_FAULTVERSION5. We have chosen these spreadsheets, because they are amongst the largest spreadsheets with respect to the number of output cells (when separately considering the number of erroneous and correct output cells).

Each sub-figure illustrates the influence of the number of incorrect (erroneous) and correct output cells on the ranking result for one spreadsheet and scenario

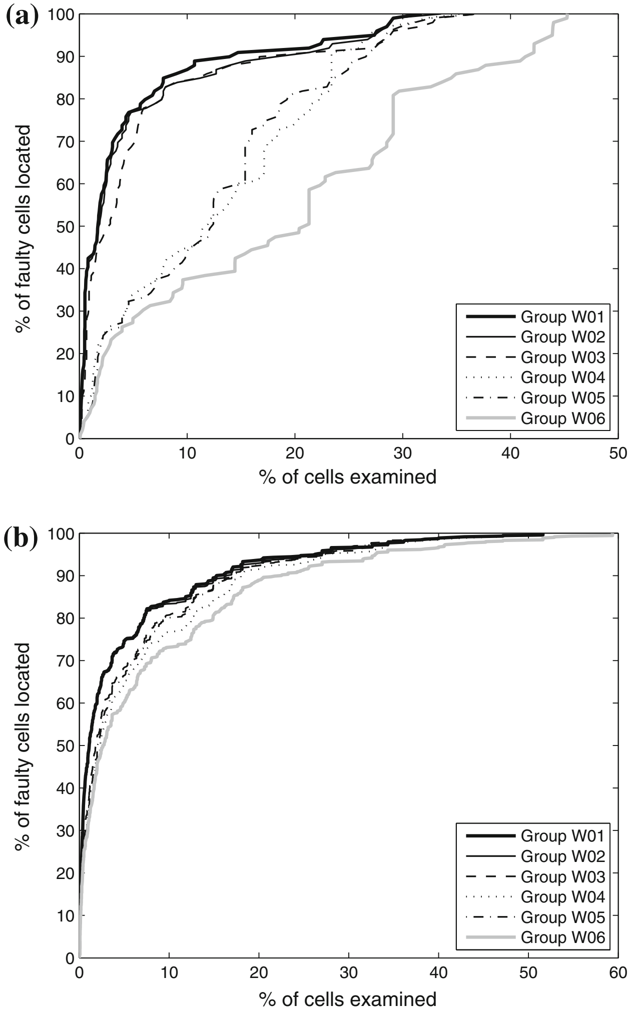


Fig. 5 Comparison of the worst case scenario clusters in terms of effort required to diagnose. **a** ISCAS85 Corpus, **b** EUSES Corpus

(best/average/worst). The concrete erroneous and correct output cells were randomly chosen from all available output cells. In order to eliminate outliers, each data point was calculated over the average of 100 runs with a different random selection of erroneous and correct output cells.

Concerning the number of erroneous output cells, we derive from the figures that indicating more incorrect output cells yields better results. For the spreadsheet `c7552` and `WorldPopPlay`, every additional faulty output cell increases the fault localization performance. However, for the spreadsheets `c5313` and `my_finanical_model`, 2 and 5 faulty output cells, respectively, are sufficient for localizing the faulty cell.

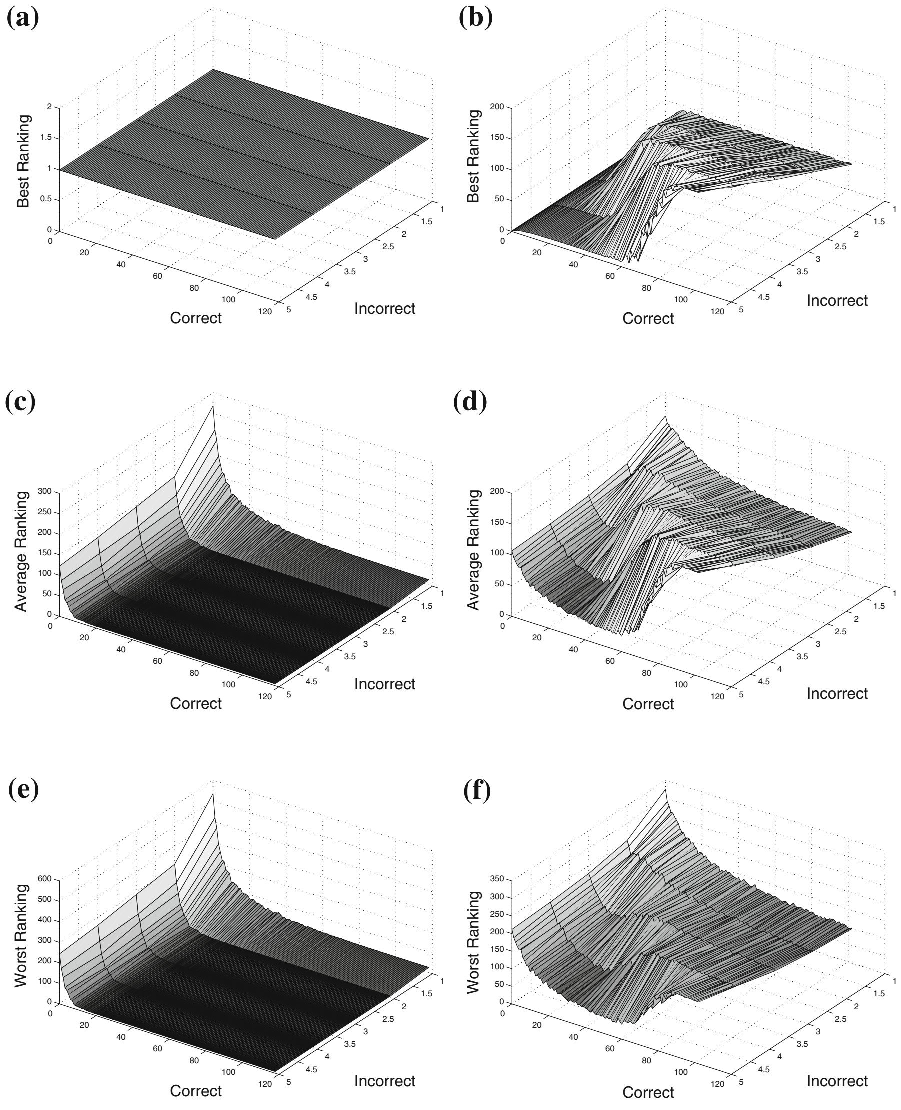


Fig. 6 Influence of the number of correct and incorrect output cells on the ranking result for the ISCAS85 spreadsheets `c5315_BOOL_tc1_71_1Fault` and `c7552_BOOL_tc1_96_1Fault`. **a** c5315 (Best), **b** c7552 (Best), **c** c5315 (Average), **d** c7552 (Average), **e** c5315 (Worst), **f** c7552 (Worst)

Concerning the correct output cells, the results for the four spreadsheets are quite different. While the Figs. 7a–f indicate that the number of correct output cells does not influence the ranking of the faulty statement, the Figs. 6a–f create other impressions: Figs. 6c and e indicate that at least 15 correct output cells should be indicated in order to obtain a good ranking of the faulty cell. Figures 6b, d and f show that too many correct output cells worsen the result. This phenomenon can be explained with the number of coincidental correct output cells: while the spreadsheets `c5315`,

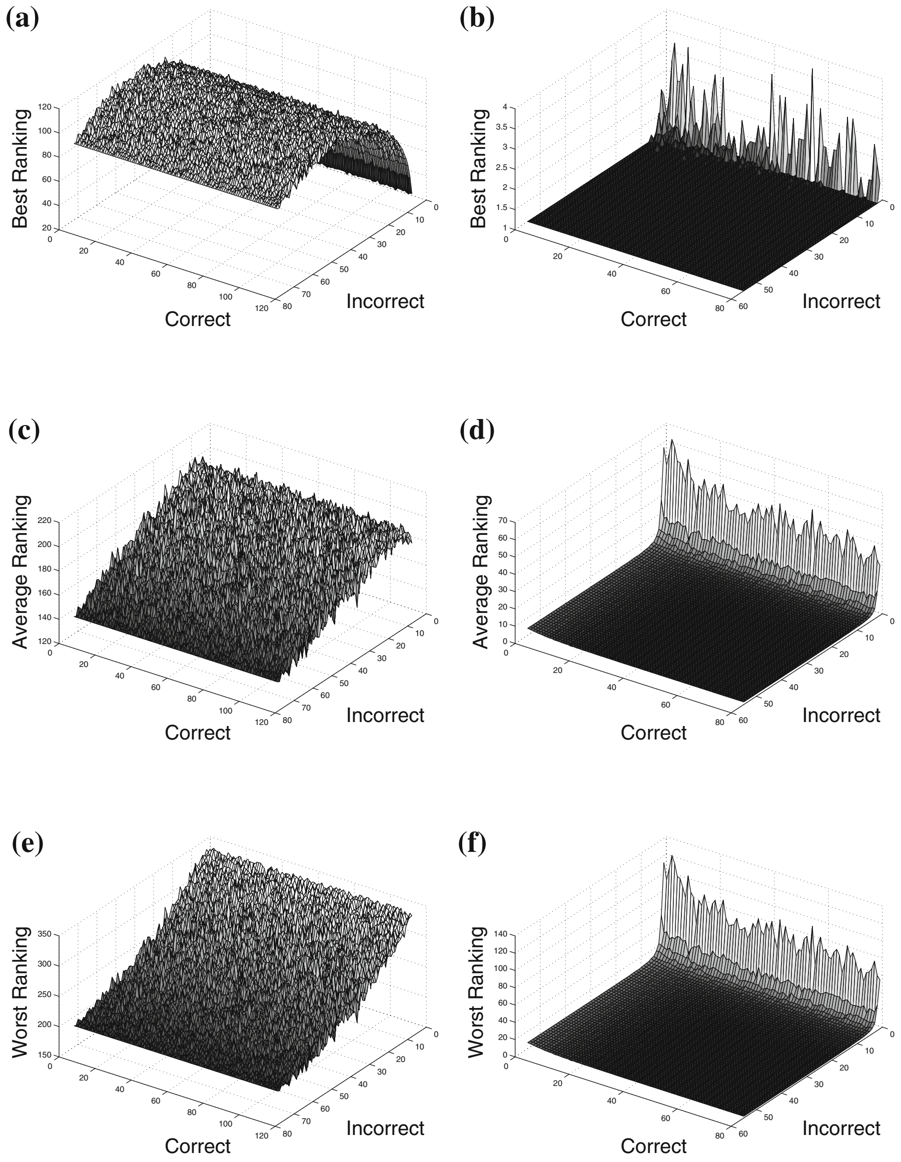


Fig. 7 Influence of the number of correct and incorrect output cells on the ranking result for the EUSES spreadsheets `WorldPopPlay_1FAULTS_FAULTVERSION1` and `my_financial_model_1FAULTS_FAULTVERSION5`. **a** WorldPopPlay (Best), **b** my_financial_model (Best), **c** WorldPopPlay (Average), **d** my_financial_model (Average), **e** WorldPopPlay (Worst), **f** my_financial_model (Worst)

`WorldPopPlay` and `my_financial_model` have no coincidental correct output cells, the spreadsheet `c7552` has 27 coincidental correct output cells.[2] As the coincidental correct output cells cannot be determined in advance and as correct output cells have only a small positive impact on the ranking result, we advice users to indicate a few but not more than 40 correct output cells.

6 Conclusion

While spreadsheets are used by a considerable number of people, there is little support for automatic spreadsheet debugging. We described what modifications are necessary for a traditional software debugging technique, SFL, to be applied to the spreadsheet domain. The main modification is the use of dependency cones instead of execution traces and slices. As SFL needs a similarity coefficient to compute a suspiciousness ranking of each component (cells in this case) being faulty, we investigate what coefficients are best suited to handle the diagnosis of spreadsheets.

We evaluated 42 different similarity coefficients using the EUSES spreadsheet corpus (Fisher and Rothermel 2005; Hofer et al. 2013), and a spreadsheet version of the ISCAS85 circuits (Brglez and Fujiwara 1985). We clustered the coefficients that yield similar diagnostic rankings, comparing them in terms of the number of cells inspected until the faulty one is reached. Our analysis shows that Jaccard (C_{16}), Ochiai (C_{24}) and Sorensen-Dice (C_{34}) are the best performing similarity coefficients to diagnose spreadsheets with SFL. In addition, we showed the influence of the number of correct and erroneous output cells on the diagnosis: (i) If the user indicates more erroneous output cells, the diagnostic ranking might be improved; (ii) Indicating more correct output cells might have a negative impact in the diagnostic ranking (due to coincidental correctness values of cells).

The comparison of SFL to other debugging approaches remains for future work. We did not compare SFL to GoalDebug (Abraham and Erwig 2005, 2007) and the work of Jannach et al. (2013) because a direct comparison is not possible for the following reasons: (1) Neither the tool GoalDebug nor the spreadsheets used in the GoalDebug's empirical evaluations are public available (Abraham and Erwig 2013). (2) The approach of Jannach et al. requires more information than our approach as they need more than one test case. However, SFL has a lower time complexity compared to other debugging methods like model-based diagnosis (MBD). In general, it is assumed that MBD delivers better diagnostic results than SFL. Despite this, preliminary studies (Hofer et al. 2013) have shown that SFL performs good in the spreadsheet domain when compared to MBD.

Finally, the debugging performance in case of multiple faults was not studied in this work and therefore remains an open research question. We expect to see an increase in the effort required to diagnose the bug, similar to SFL in software debugging (Abreu et al. 2009). However, the degree to which each similarity coefficient is affected when dealing with multiple faults is not known.

[2] As already mentioned, coincidental correct output cells have a negative impact on the ranking of the faulty cell.

Acknowledgments This work was supported by the Foundation for Science and Technology (FCT), of the Portuguese Ministry of Science, Technology, and Higher Education (MCTES), under Project PTDC/EIA-CCO/108613/2008, and the competence network Softnet Austria II (www.soft-net.at, COMET K-Projekt) funded by the Austrian Federal Ministry of Economy, Family and Youth (bmwfj), the province of Styria, the Steirische Wirtschaftsförderungsgesellschaft mbH. (SFG), and the city of Vienna in terms of the center for innovation and technology (ZIT).

References

Abraham, R., Erwig, M.: Goal-directed debugging of spreadsheets. In: Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing. VLHCC '05, pp. 37–44. IEEE Computer Society, Washington, DC, USA (2005)

Abraham, R., Erwig, M.: Autotest: a tool for automatic test case generation in spreadsheets. In: Proceedings of the 2006 IEEE Symposium on Visual Languages and Human-Centric Computing. VLHCC '06, pp. 43–50. Brighton, UK (2006)

Abraham, R., Erwig, M.: Goaldebug: a spreadsheet debugger for end users. In: 29th IEEE International Conference on Software Engineering, pp. 251–260 (2007)

Abraham, R., Erwig, M.: Ucheck: a spreadsheet type checker for end users. J. Vis. Lang. Comput. **18**, 71–95 (2007). doi:10.1016/j.jvlc.2006.06.001

Abraham, R., Erwig, M.: Mutation operators for spreadsheets. IEEE Trans. Softw. Eng. **35**(1), 94–108 (2009)

Abraham, R., Erwig, M.: Personal communication (2013)

Abreu, R., Riboira, A., Wotawa, F.: Constraint-based debugging of spreadsheets. In: Proceedings of the 15th Ibero-American Conference on Software Engineering (2012)

Abreu, R., Zoeteweij, P., van Gemund, A.: An evaluation of similarity coefficients for software fault localization. In: Proceedings of the 12th Pacific Rim International Symposium on Dependable Computing, PRDC '06, pp. 39–46. IEEE Computer Society, Washington, DC, USA (2006). doi:10.1109/PRDC.2006.18

Abreu, R., Zoeteweij, P., van Gemund, A.J.C.: On the accuracy of spectrum-based fault localization. In: Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques—MUTATION, TAICPART-MUTATION '07, pp. 89–98. IEEE Computer Society, Washington, DC, USA (2007). http://dl.acm.org/citation.cfm?id=1308173.1308264

Abreu, R., Zoeteweij, P., Golsteijn, R., Van Gemund, A.J.: A practical evaluation of spectrum-based fault localization. J. Syst. Softw. **82**(11), 1780–1792 (2009)

Aggarwal, C.C., Yu, P.S.: A new framework for itemset generation. In: Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'98), pp. 18–24 (1998)

Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: Proceedings of 20th International Conference on Very Large Data, Bases (VLDB'94), pp. 487–499 (1994)

Agresti, A.: An Introduction to Categorical Data Analysis. Wiley, New York (1996)

Anderberg, M.R.: Cluster Analysis for Applications. Technical Report. Academic Press, London (1973)

Ayalew, Y., Mittermeir, R.: Spreadsheet debugging. Bilding Better Business Spreadsheets—from the ad-hoc to the quality-engineered. In: Proceedings of EuSpRIG 2003, Dublin, Ireland, 24–25 July 2003, pp. 67–79 (2003)

Bregar, A.: Complexity metrics for spreadsheet models. The Computing Research Repository (CoRR) abs/0802.3895 (2008). http://arxiv.org/abs/0802.3895

Brglez, F., Fujiwara, H.: A neutral netlist of 10 combinational benchmark circuits and a target translator in fortran. In: Proceedings of the IEEE International Symposium on Circuits and Systems, pp. 663–698 (1985)

Brin, S., Motwani, R., Ullman, J.D., Tsur, S.: Dynamic itemset counting and implication rules for market basket data. SIGMOD Rec. **26**(2), 255–264 (1997)

Burnett, M.M., Cook, C.R., Pendse, O., Rothermel, G., Summet, J., Wallace, S.: End-user software engineering with assertions in the spreadsheet paradigm. In: Clarke L.A., Dillon L., Tichy W.F. (eds.) ICSE, pp. 93-105. IEEE Computer Society (2003)

Campos, J., Riboira, A., Perez, A., Abreu, R.: Gzoltar: an eclipse plug-in for testing and debugging. In: Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, ASE 2012, pp. 378–381. ACM, New York, NY, USA (2012). doi:10.1145/2351676.2351752

Chadwick, D., Knight, B., Rajalingham, K.: Quality control in spreadsheets: a visual approach using color codings to reduce errors in formulae. Softw. Qual. J. **9**(2), 133–143 (2001)

Cheng, H., Lo, D., Zhou, Y., Wang, X., Yan, X.: Identifying bug signatures using discriminative graph mining. In: Proceedings of the 18th International Symposium on Software Testing and Analysis (ISSTA '09), pp. 141–152 (2009)

Clark, P., Boswell, R.: Rule induction with cn2: some recent improvements. In: EWSL, Lecture Notes in Computer Science, vol 482, pp. 151–163. Springer (1991)

Coblenz, M.J., Ko, A.J., Myers, B.A.: Using objects of measurement to detect spreadsheet errors. In: VL/HCC, pp. 314–316. IEEE Computer Society (2005). doi:10.1109/VLHCC.2005.67

Cohen, J.: A coefficient of agreement for nominal scales. Educ. Psychol. Meas. **20**(1), 37–46 (1960)

Cunha, J., Fernandes, J.P., Mendes, J., Saraiva, J.: Mdsheet: a framework for model-driven spreadsheet engineering. In: Glinz M., Murphy G.C., Pezzè M. (eds.) ICSE, pp. 1395–1398. IEEE (2012)

Dice, L.R.: Measures of the amount of ecologic association between species. Ecology **26**(3), 297–302 (1945)

Fisher, M., Cao, M., Rothermel, G., Cook, C.R., Burnett, M.M.: Automated test case generation for spreadsheets. In: Proceedings of the 24th International Conference on Software Engineering, ICSE '02, pp. 141–151. ACM Press (2002)

Fisher, M.I., Rothermel, G.: The EUSES spreadsheet corpus: a shared resource for supporting experimentation with spreadsheet dependability mechanisms. In: 1st Workshop on End-User, Software Engineering, pp. 47–51 (2005)

Geng, L., Hamilton, H.J.: Interestingness measures for data mining: a survey. ACM Comput. Surv. **38**(3), 1–32 (2006)

Goodman, L.A., Kruskal, W.H.: Measures of association for cross classifications. J. Am. Stat. Assoc. **49**(268), 732–764 (1954)

Hand, D.J., Smyth, P., Mannila, H.: Principles of Data Mining. MIT Press, Cambridge (2001)

Healey, J.: Statistics: A Tool for Social Research. Wadsworth Publishing, Belmont (1993)

Hermans, F., Pinzger, M., van Deursen, A.: Automatically extracting class diagrams from spreadsheets. In: D'Hondt T. (ed.) ECOOP, Lecture Notes in Computer Science, vol 6183, pp. 52–75. Springer (2010). doi:10.1007/978-3-642-14107-2_4

Hermans, F., Pinzger, M., van Deursen, A.: Breviz: visualizing spreadsheets using dataflow diagrams. The Computing Research Repository (CoRR) abs/1111.6895 (2011). http://arxiv.org/abs/1111.6895

Hermans, F., Pinzger, M., van Deursen, A.: Detecting code smells in spreadsheet formulas. In: ICSM, pp. 409–418. IEEE Computer Society (2012a)

Hermans, F., Pinzger, M., van Deursen, A.: Measuring spreadsheet formula understandability. The Computing Research Repository (CoRR) abs/1209.3517 (2012b)

Hermans, F., Pinzger, M., van Deursen, A.: Detecting and visualizing inter-worksheet smells in spreadsheets. In: Proceedings of the 2012 International Conference on Software Engineering, pp. 441–451. IEEE Press (2012c)

Hermans, F., Sedee, B., Pinzger, M., van Deursen, A.: Data clone detection and visualization in spreadsheets. In: Notkin D., Cheng B.H.C., Pohl K. (eds.) ICSE, pp. 292–301. IEEE / ACM (2013). http://dl.acm.org/citation.cfm?id=2486827

Hofer, B., Riboira, A., Wotawa, F., Abreu, R., Getzner, E.: On the empirical evaluation of fault localization techniques for spreadsheets. In: Cortellessa, V., Varró, D. (eds.) FASE, Lecture Notes in Computer Science, pp. 68–82. Springer (2013)

Jannach, D., Baharloo, A., Williamson, D.: Toward an integrated framework for declarative and interactive spreadsheet debugging. In: 6th International Conference Evaluation of Novel Approaches to Software Engineering (ENASE), Angers, France, pp. 117–124 (2013)

Jannach, D., Engler, U.: Toward model-based debugging of spreadsheet programs. In: 9th Joint Conference on Knowledge-Based Software Engineering (JCKBSE'10) 25–27 Aug 2010, Kaunas, Lithuania pp. 252–264 (2010)

Janssen, T., Abreu, R., van Gemund, A.: Zoltar: A toolset for automatic fault localization. In: Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering, ASE '09, pp. 662–664. IEEE Computer Society, Washington, DC, USA (2009). doi:10.1109/ASE.2009.27

Jones, J.A., Harrold, M.J.: Empirical evaluation of the tarantula automatic fault-localization technique. In: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering, ASE ’05, pp. 273–282 (2005)

Klösgen, W.: Explora: a multipattern and multistrategy discovery assistant. In: Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R. (eds.) Advances in Knowledge Discovery and Data Mining, pp. 249–271. American Association for Artificial Intelligence, Menlo Park (1996)

Ko, A.J., Abraham, R., Beckwith, L., Blackwell, A., Burnett, M., Erwig, M., Scaffidi, C., Lawrance, J., Lieberman, H., Myers, B., Rosson, M.B., Rothermel, G., Shaw, M., Wiedenbeck, S.: The state of the art in end-user software engineering. ACM Comput. Surv. **43**(3), 1–44 (2011)

Lo, D., Jiang, L., Budi, A., et al.: Comprehensive evaluation of association measures for fault localization. In: 2010 IEEE International Conference on Software Maintenance (ICSM), pp. 1–10. IEEE (2010)

Lucia, Lo, D., Jiang, L., Thung, F., Budi, A.: Extended comprehensive study of association measures for fault localization. J. Softw. (2013). doi:10.1002/smr.1616

MacQueen, J.: Some methods for classification and analysis of multivariate observations. In: Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability, University of California 1965/66, 1, pp. 281–297 (1967)

Mittermeir, R., Clermont, M.: Finding high-level structures in spreadsheet programs. In: van Deursen, A., Burd, E. (eds.) WCRE, pp. 221–232. IEEE Computer Society (2002). http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1173080&tag=1

Nica, I., Pill, I., Quaritsch, T., Wotawa, F.: The route to success—a performance comparison of diagnosis algorithms. In: Rossi F. (ed.) IJCAI. IJCAI/AAAI (2013)

Ochiai, A.: Zoogeographic studies on the soleoid fishes found in japan and its neighbouring regions. Bull. Jpn. Soc. Sci. Fish. **22**, 26–520 (1957)

Ohsaki, M., Kitaguchi, S., Okamoto, K., Yokoi, H., Yamaguchi, T.: Evaluation of rule interestingness measures with a clinical dataset on hepatitis. In: Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases. PKDD ’04, pp. 362–373. New York, NY, USA (2004)

Piatetsky-Shapiro, G.: Discovery, analysis, and presentation of strong rules. In: Piatetsky-Shapiro, G., Frawley, W.J. (eds.) Knowledge Discovery in Databases, pp. 229–248. AAAI/MIT Press, Menlo Park (1991)

Rogers, D.J., Tanimoto, T.T.: A computer program for classifying plants. Science **132**(3434), 1115–1118 (1960)

Rothermel, K.J., Cook, C.R., Burnett, M.M., Schonfeld, J., Green, T.R.G., Rothermel, G.: WYSIWYT testing in the spreadsheet paradigm: an empirical evaluation. In: Proceedings of ICSE’00, pp. 230–239. ACM (2000). http://doi.acm.org/10.1145/337180.337206

Ruthruff, J., Creswick, E., Burnett, M., Cook, C., Prabhakararao, S., Fisher II, M., Main, M.: End-user software visualizations for fault localization. In: Proceedings of the 2003 ACM symposium on Software visualization, SoftVis ’03, pp. 123–132. ACM, New York, NY, USA (2003). doi:10.1145/774833.774851

Shortliffe, E.H., Buchanan, B.G.: A model of inexact reasoning in medicine. Math. Biosci. **23**(3–4), 351–379 (1975)

Smyth, P., Goodman, R.: An information theoretic approach to rule induction from databases. IEEE Trans. Knowl. Data Eng. **4**(4), 301–316 (1992)

Sokal, R.R., Michener, C.D.: A statistical method for evaluating systematic relationships. Univ. Kans. Sci. Bull. **38**, 1409–1438 (1958)

Tan, P.N., Kumar, V., Srivastava, J.: Selecting the right interestingness measure for association patterns. In: Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD’02), pp. 32–41 (2002)

Tikir, M.M., Hollingsworth, J.K.: Efficient instrumentation for code coverage testing. In: Proceedings of the 2002 ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA ’02, pp. 86–96. ACM, New York, NY, USA (2002). doi:10.1145/566172.566186

Wang, X., Cheung, S.C., Chan, W.K., Zhang, Z.: Taming coincidental correctness: coverage refinement with context patterns to improve fault localization. In: ICSE, pp. 45–55. IEEE (2009)

Wilcoxon, F.: Individual comparisons by ranking methods. Biom. Bull. **1**(6), 80–83 (1945)

Xu, X., Debroy, V., Wong, W.E., Guo, D.: Ties within fault localization rankings: exposing and addressing the problem. Int. J. Softw. Eng. Knowl. Eng. **21**(6), 803–827 (2011)