Copy-Paste Detection in Spreadsheets

Ben Sedee

Copy-Paste Detection in Spreadsheets

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Ben Sedee
born in Delft, the Netherlands

Software Engineering Research Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

Infotron
Reduitlaan 33
Breda, the Netherlands
www.infotron.nl

Copy-Paste Detection in Spreadsheets

Author:	Ben Sedee
Student id:	1263153
Email:	`bmwsedee@gmail.com`

Abstract

When a company is in need of a reporting tool, the most commonly made decision is to choose for Excel. In fact, over 90% of the world's companies base their decisions on a report made using Excel. This shows that the number of spreadsheet designers, of end-user programmers, is large. It has been estimated to be 5 times as large as the number of software programmers in the traditional sense. This is one of the reasons spreadsheets are error-prone, possibly leading to erroneous decisions. One of the causes of problems within spreadsheets is the prevalence of copy-pasting. In this thesis we have studied this problem and we present an algorithm to detect data clones within spreadsheets: formulas whose values are copied in a different location. Aside from this algorithm, which we based on existing algorithms for code clone detection in software engineering, we present a classification scheme for the found data clones. We evaluated both the algorithm and the classification using the EUSES corpus, resulting in the conclusion that data clones in spreadsheet are as common as code clones in source code. We also show that we are able to detect these data clones with precision rates similar to those achieved by state-of-the-art code clone detection algorithm.

Preface

I would like to thank a number of people for their help and support during the writing of this thesis. Firstly, there are Martin, my daily supervisor at the TU Delft, who has guided me during my research and kept me from on track and on time, and Felienne, my supervisor at Infotron, who had the patience to read my lengthy e-mails and reply with an invitation to discuss the e-mail in person. Secondly, there are Arie van Deursen and Willem-Paul Brinkman, the other two members of my thesis committee, whom I would like to thank for taking place in this committee.

Obviously, I would not have been able to begin this thesis without all the previous education of my Master's and Bachelor's degree, but since this is obvious, I probably should not even mention it. The same holds for the support my parents have shown me throughout the seven-and-a-half years I've been a student now, but in spite of its self-evidence, my appreciation knows no bounds.

By name I would also like to mention my dad, again, and Timo, who both had the patience to proof-read this thesis and look for any mistakes I made in my grammar or spelling. My guess is that they both learnt a lot they didn't want to know about spreadsheets, and I can only hope they also learnt that copying formulas as values is bad practice. Elise, this may sound harsh, but thank you for leaving to Salzburg for half a year for your own studies. This freed-up a lot of time for me in the evenings and weekends to work on my thesis, without which I would possibly not even be half-way done by now. Your long-distance support and confidence in me has definitely given me an extra push in times I needed it. And Peter, thanks for the advice of storing an offline back-up, even though I'm glad I didn't need it.

Finally I would like to thank Herman, for starting my interest in Excel spreadsheets roughly 4 years ago and I would like to thank my employers at aNewSpring for allowing me to take the time off when I needed it.

Ben Sedee
Delft, the Netherlands
January 23, 2013

Contents

List of Figures

Chapter 1

Introduction

It is generally accepted that Excel is a very powerful tool for a number of applications, among which are data reporting and manipulation and the accompanying ease of use. It is primarily in its capacity as reporting tool that over 90% of the world's companies base the majority of their decisions on reports created in Excel. Given these numbers, it would seem logical that the models that serve as the basis of these reports are thoroughly checked. Alas, this is not the case, which leads to error-prone spreadsheets. Even in modern-day society, errors within spreadsheets still occur, leading from an overbooking in the latest Olympics to false examination results at a prestigious university and to missing taxable properties worth well over a billion US Dollars[1].

To address this issue, researchers have analysed spreadsheets, their users and their practices [22, 29, 45]. One of the findings was that spreadsheets tend to be full of references, be it to other cells, other worksheets or other spreadsheets. While most of these links are visible by looking at the formula typed in the cells, for large spreadsheets users lose track of the links and are unable to keep an overview. A second common issue is that the visible links only work in one direction, i.e. only the source of the data is visible. It is often not visible where a certain piece of data is being referenced and what other cells may change when a cell is edited. Having built a tool to overcome this weakness, (informal) research at Infotron[2] has revealed another, even less visible, source of possible data corruption, i.e. the copy-pasting of data.

This thesis focusses on exactly that: the detection of copy-pasted data. When users copy the value of a formula to another cell, the link that implicitly exists between the two cells is not stored anywhere, so updating either side of this implicit link will unknowingly sever it. Therefore the users are greatly helped if these links can be visualised for the users. This visualisation also serves the users to encourage them to make the links explicit by referring explicitly instead of copying the contents.

The algorithm presented in this theses resulted in a presentation to be held at the International Conference on Software Engineering 2013 [30]. In addition, we are looking into possibilities of writing a new paper or extending other papers based on the classification of

[1]`http://www.eusprig.org/horror-stories.htm`
[2]`http://app.infotron.nl`

copy-paste relations presented in this thesis.

1.1 Relevance

This thesis shows that it is possible to apply proven code clone detection techniques like Johnson's culling [31] and Ducasse et al.'s storing of previous results [20] to spreadsheet research. These techniques have been applied to spreadsheets to find the copy-paste relations, while other techniques like the categorisations of Balazinska et al. [11] and Davey et al. [18] have been used to categorise the retrieved data.

Aside from the applied techniques, this thesis has helped improve the notion of what a "good" spreadsheet is. By uncovering the hidden relations that exist between different cells in a spreadsheet, the existence of these links is shown and due to the quantitative analysis conclusions can be drawn regarding the frequency of these relations. For spreadsheets where these relations are numerous, this thesis also proposes a classification schema, which can also be used to help users maintain an overview of these relations. Lastly, this thesis provides numerous handles for future research into improving spreadsheet quality.

1.2 Research Questions

By watching users work with spreadsheets during other projects at Infotron it was found that users tend to copy-paste data and outcomes of formulas and that the resulting copies could persist through numerous versions of a spreadsheet. These conclusions led to the start of this thesis, and a general question "How can the copies of data in spreadsheets be detected?".

Looking closer at the type of data that was mostly copied, it appeared that the copying mostly concerned the outcomes of formulas. Moreover, we needed to develop a way of reporting the findings, allowing the spreadsheet-user to understand the results. These two observations led to a change of the main question to "How can the copies of formula-data in spreadsheets be detected" and "How can the results be reported to the user?".

Another obvious issue that needed to be addressed regarded the reporting of the results: "How can the results be reported in a way that is usable for the users?". Exploring the reporting side a bit further, the issue of coping with the possibly large amount of found copies came up. To address this, the question "How can the results be ordered in a useful way?" arose and is answered in this thesis. Finally, to validate the effort and the algorithm, the question "How well does the proposed solution work?" is answered.

All in all, this leads to the following research questions:

R.Q. 1 How can the copies of formula-data in spreadsheets be detected?

R.Q. 2 How can the results be reported to the user?

- a) How can the results be reported in a way that is usable for the users?
- b) How can the results be ordered in a useful way?

R.Q. 3 How well does the proposed solution work?

1.3 Outline

The posed research questions lead to a sectioned outline of this thesis. Firstly, some groundwork is presented in Chapter 2. Following Chapter 2, the Chapters all represent one of the questions. In Chapter 3 a first attempt at an algorithm is presented, originating from the original research question. This algorithm is abandoned in favour of the algorithm presented in Chapter 4. The next chapter concerns both the ordering of the results and the closely related reporting of the results. Chapter 6 handles the validation of the presented algorithm and ordering, and the thesis is concluded with a discussion of the results and some pointers to future research possibilities.

Chapter 2

Related Work

For this thesis there were several research areas to look at and to get inspiration from. Looking for copies of formula outcomes within the same spreadsheet intuitively felt like looking for code clones in software programs. Two algorithmic concepts found here could be transcribed into algorithms used for copy-paste detection in spreadsheets. The first concept entails keeping an ordered and culled list of values that occur within a spreadsheet (as explained Johnson in [31]). The second important concept is to store the results of previous comparisons and to utilise this to reduce the total number of comparisons, which is explained by Ducasse et al. [20]. Also, a definition of equality has been formulated.

In addition to code clone detection, previous research has also focussed on categorising the different kinds of code clones. The existing categories and taxonomies can be used as a basis for the categorisation of the found copy-paste relations in spreadsheets. Combining the categorisations of Davey et al. [18] and Balazinska et al. [11] results in a two-level categorisation of copy-paste relations, with both levels containing three categories. When, like Kapser and Godfrey [34] proposed, the locations of the relations are also taken into account, a distinction into five levels can again be created, each of these levels possibly containing these 6 categories. This leads to a possible categorisation into 30 different categories.

Existing research into spreadsheets themselves can be divided into a number of areas. We looked closely at the two areas we felt were closest related to this thesis: the testing of spreadsheets and the auditing of the sheets. Both these areas had to be looked at, to establish the idea of copy-detection within spreadsheets as a new concept. Another reason to examine previous studies into spreadsheets is because there might have been some research initiatives that posed ideas specific to spreadsheets that could be used. In the area of spreadsheet testing, a very promising line of research is that of automatic header inference (as described by Abraham en Erwig in [1]), but this is not fully useful for the purposes of this thesis. Other useful ideas and (partial) definitions are taken from spreadsheet auditing. The ideas taken from spreadsheet auditing, however, also need to be adapted before they can be used to look for formula-copies.

2.1 Code Clone Detection

Code clone detection has been a topic of research for software engineers for the past 20 years. Most research done before 2007 has been surveyed by Roy and Cordy [51] and from their survey it is clear that two major approaches have been pursued. Algorithms developed until now can, with some exceptions, be divided in two trends; being either based on the unprocessed source code, or being based on an abstract version of the source code like an abstract syntax tree. Exceptions to this division are hybrid approaches like Kontogiannis et al. [36], or approaches based on metrics of the source code (e.g., Mayrand et al. [38]), but these are not as widely researched. Since the basic building blocks of spreadsheets are cells and the copy-paste detection of formula outcomes is based on these cells, algorithms based on an analysed version of the source code are deemed unusable. The algorithms using the unprocessed source code can be divided into two main categories:

- Text-based algorithms
- Token-based algorithms

The difference between these two is the size of the two parts that are being compared. In text-based algorithms whole lines of source code are compared, while in token-based algorithms this comparison is done on a per-token basis, where the token size can differ among the various algorithms.

Most of the drawbacks of the text-based algorithms can be ignored when the algorithmic ideas are transcribed into a spreadsheet environment, leaving the underlying ideas available. The most promising of these ideas are the culling of the list of values (as done by Johnson [31]) and storing the results of comparisons in a matrix (as done by Ducasse et al. [20]). Drawbacks of text-based algorithms that cannot be ignored in spreadsheets are alleviated by using token-based algorithms. Based on the definition given by Baker [9], token-based algorithms yield a useful definition of equality among cells: “two cells are considered to be identical if they contain the same values after evaluating the formulas; the semantics of the formulas are not analysed”.

2.1.1 Text-based algorithms

In [51], Roy and Cordy mention four main drawbacks of a text-based algorithm in general when it is applied to code clone detection. These drawbacks are based on specific characteristics of software languages:

1. Line breaks, which can be in different places without disrupting the meaning or workings of the program.
2. Identifier changes, which revolve around the naming of variables and functions.
3. Parentheses, which can be added or removed without disrupting the meaning or workings of the program, much like line breaks.
4. Transformation, which means that a normalization sometimes has to be applied.

Considering the use of text-based algorithms for copy-paste detection in spreadsheets, these drawbacks need to be addressed. Examination shows, however, that of the four mentioned characteristics only line breaks have a spreadsheet equivalent, because they can be translated as a different ordering of the cells.

Having concluded that most of the drawbacks of this type of algorithms do not apply, three specific algorithms have implemented useful concepts. The first of these algorithms, and one of the first in general to use a text-based approach, was developed by Baker [9]. Baker's definition of equality among lines of code, "Two lines of code are considered to be identical if they contain the same sequence of characters after removing comments and white space; the semantics of the program statements are not analysed", can rather easily be translated into a definition for equality among spreadsheet cells. Analogous to this definition, this equality would mean that two blocks of cells are considered to be identical if they contain the same values after evaluating the formulas; the semantics of the formulas are not analysed. Evaluating the formula without analysing in this case means that only the results of the formula count, not the way these results are gathered. Embedded in the definition of equality as stated above is the idea that comments and white spaces should be removed when comparing two lines of source code. Considering blank cells to be the spreadsheet equivalent of white spaces, and headers to be the equivalent of comments, the definition becomes too coarse because the blocks considered for equality could potentially span the complete file. Also, too keep this definition analogous to source code lines, the order of cells within two blocks of cells must be equal, while in spreadsheets it is a one-click operation to disrupt this.

Johnson's text-based algorithm [31] identifies code clones based on fingerprints created of lines of source code. His goal is to find redundant code, which he identifies as "any characteristic of the source that could be used by a data compression algorithm to encode the source in fewer bits." Even though it is not the purpose of this thesis to encode spreadsheets in fewer bits, the underlying principles are still useful. Johnson's algorithm is based on a number of steps, of which a number are useful:

1. Fingerprints are calculated for substrings in the source code. The substrings used by Johnson are called *snips* and can consist of multiple lines, but are still substrings.

2. The calculated fingerprints are collected in a file and sorted by value.

3. The values that occur only once are removed with a process called *culling*. The remaining values identify redundant code.

Translating these steps so they can be used for copy-paste detection in spreadsheets, the calculation of fingerprints can be skipped. Since the goal is to find multiple cells with the same value, the value of the cells can be used as a fingerprint, making the calculation of a fingerprint for a cell equal to evaluating the cell's formula. The other steps, i.e. collecting, sorting and culling, remain the same.

The last algorithm with useful ideas was created by Ducasse et al. [20]. Ducasse compares every line with every other line in the file, storing the results of the comparisons in a boolean matrix. The results are then extracted from this matrix to determine if a number of subsequent lines, i.e. lines forming a block of code, are copies of each other. The idea of storing the comparison results in a matrix is a useful one for copy-paste detection in spreadsheets. Consider, for example, a situation where two formula cells have the same outcome, as do three value cells. For the first of these formula cells, a comparison is done with all three value cells and the results of these comparisons are stored. The second formula cell is now compared to the first of the three value cells, and found to be equal. Given the results of the comparisons with the first formula cell it is now known that the second formula cell is also equal to the other two value cells. Since the comparison with the other two cells can now be skipped, a speed-up in the algorithm has been achieved.

2.1.2 Token-based algorithms

Token-based algorithms are a generalisation of the text-based algorithms discussed above. That is to say, every text-based algorithm is basically a token-based algorithm with a line of source code as token. Not having the restriction to only look at complete lines of source-code, however, gives the possibility to alleviate some of the discussed drawbacks of text-based algorithms, most notably the line breaks. Since tokens can start in the middle of a line of source code and end in the middle of the next line, line breaks will have to be ignored when comparing the tokens. Knowing that line breaks are the only drawback of text-based algorithms that have an equivalent in spreadsheets, and seeing as how token-based algorithms alleviate this drawback, token-based algorithms served as an inspiration for this thesis.

One example of a text-based algorithm that is also token-based is Baker's algorithm [9]. Since the tokenizer used in the algorithm allows substrings to be of arbitrary length they do not necessarily have to be complete lines. Regardless of the choice for substrings, the algorithm matches the substrings using a suffix tree [39]. This techmeannique is also used by one of the most prominent algorithms, CCFinder [32]. Applying one of these algorithms to copy-paste detection in spreadsheets, would indicate that spreadsheets need to be fitted to the structure of such a suffix tree. However, thinking about the structure of such a tree and the reasons for the algorithms to use this structure leads to the conclusion that the benefit of the structure a suffix tree offers is available in spreadsheets by default. In its most basic form, the structure of a suffix tree does not differ from the structure of a tree, with a root node containing (pointers to) child nodes, where every child node can be the root node of a sub-tree, eventually ending in the leaf nodes of the tree. The advantage of using this kind of structure for the token matching is that the next token to be matched can easily be retrieved, without having to consult the original source code. Considering that easy access to subsequent elements is the reason to use a tree, the structure of a spreadsheet itself is such that it can directly be used, without transforming it to a suffix tree.

Having already mentioned Baker's algorithm to be also token-based, the definition of equality can be revisited. Previously, the definition applying to spreadsheets was deemed too coarse, since, analogous to source code lines, it had to be based on blocks of cells. This disadvantage can now be overcome, since smaller portions of source code also apply to the original definition. Using blocks of cells, the definition was "two blocks of cells are considered to be identical if they contain the same values after evaluating the formulas; the semantics of the formulas are not analysed”. Narrowing the blocks of cells to cells yields a useful definition for equality among cells.

2.2 Code Clone Categorization

Given that in this thesis we don’t only want to be able to detect copy-paste relations in spreadsheets, but we also want to be able to report these to the user, a categorization is deemed to be useful. Since the copy-paste detection has established analogies to code clone detection in software engineering, is was logical that we looked at existing code clone classification schemas and taxonomies for the categorisation of the code clones.

Categorisation of code clones has been done in a number of different ways (see for example Davey et al. [18] and Balazinska et al. [11]). Taking a combination of these different categories gives an overall categorisation:

Type_S I	Identical values and formatting
Type_S II	Identical values
Type_S III	Modified values:
	a) One-cell difference
	b) Multiple one-cell differences
	c) Multiple differences

Also, a differentiation can be made based on the location of the copy-paste relation, analogous to Kapser and Godfrey [34]. The differences in location that are differentiated are, in an ordinal ordering:

1. Different cell-locations within the same block of cells.
2. Different block-locations within the same worksheet.
3. Different worksheet-locations within a spreadsheet.
4. Different spreadsheets within the same directory.
5. Spreadsheets within different directories.

2.2.1 Clone Types

One kind of distinction between different types of code clones is a division in Type I – Type IV. Since this division has been used by a number of researchers [12, 18, 51] to categorise code clones, these types have also been analysed for an analogy in copy-paste relations in spreadsheets. The definitions of the Types for code clones are as follows:

Type I	Identical clones, except for differences in white space and comments.

Type II	Structurally identical clones. Like Type I, but also allows for changes in types and identifiers.
Type III	Modified clones. Like Type II, but allows for more changes like added or removed statements.
Type IV	Functional clones. Not based on syntax but on semantics, i.e. pre- and post-conditions must be equal.

Transcribing this to cell relations in spreadsheets, Type IV is a type of clone that has an analogy between two different formula cells and thus has no analogy in the copy-paste relations. For the three remaining types an analogy has been found by introducing a concept of *formatting differences* between cells. From the definitions of the code clone types it is clear that every Type is an extension of the previous Type with Type I being the first, and thus most strict, type of clone. To be able to transcribe this to a spreadsheet analogy, a distinction must be made possible between different copy-paste relations found. Since in spreadsheets, unlike in most software languages, values can get a different meaning when the formatting becomes different, using this formatting was a way of distinguishing between multiple copy-paste relations. Contrastingly, the concept of identifiers is one that does exist in software languages but not in spreadsheets. Exchanging these two concepts in the definitions of the Types led to three Types for spreadsheet. To avoid confusion of the code clone Types and spreadsheet Types, we annotate the Type with a subscripted "c" resp. "s":

Type_S I	Identical values and formatting, meaning no changes in formatting between two cells is allowed.
Type_S II	Identical values. Like Type_S I, but small changes in formatting between two cells are allowed.
Type_S III	Modified values. Like Type_S II, but the values are also allowed to differ.

It stands to reason that this definition of Type_S III allows for every formula cell to be matched to every value cell. To remedy this, a Type_S III clone is only allowed to exist in a relation between two blocks of cells, where a threshold of a minimum number of value-equal cells should be specified.

2.2.2 Clone Taxonomies

In addition to the discussed Types, several studies have devised some other form of classifying code clones. Balazinska et al. [11] created a schema of 18 different categories, split into three levels. Considering only the first two levels to avoid too specific constructs of software languages, four categories remain:

1. Identical. This is like Type_C I.

2. One-token difference. Only one token in the clone may differ.

3. Multiple one-token differences. The one-token differences are not allowed to be consecutive.

4. Multiple token differences. The one-token differences are consecutive, leading to a multi-token difference.

Converting these categories to a spreadsheet environment and using the formatting differences as defined above results in:

1. Identical. This is like Type$_S$ I.

2. One-cell difference. This allows for either one value difference or one formatting difference.

3. Multiple one-cell differences. This allows for non-bordering cells to have either a value or formatting difference.

4. Multiple cell differences. This allows for any cell to differ, in both value as formatting. This is like Type$_S$ III.

Two of the four categories have a clear relation with the defined three Types$_S$, and this relation is stated in the definition of the category. None of these categories match with Type$_S$ II, since Type$_S$ II does not allow for value differences and these categories do. Therefore, the second and third category should be seen as sub-categories of Type$_S$ III.

Kapser and Godfrey [33, 34] have attempted to not only divide the clones by similarity, but to also take the location of the clone into account. Such a division for copy-paste detection in spreadsheets can be done on three levels. Firstly, a location division for clones can be performed within one spreadsheet itself. Cells reside in blocks of cells and can thus have a location based on the block of cells. Secondly, the blocks of cells themselves are elements of a worksheet and have a location based on the worksheet. Finally, different worksheets make one total spreadsheet, constituting three levels of location within one spreadsheet. This is one level more than Kapser and Godfrey define, as they distinguish only “Regions” and “Same file”. When relations are to be found within multiple spreadsheets, the same levels as used by Kapser and Godfrey can be used, i.e. “Same file”, “Same directory” and “Different directory”.

2.3 Spreadsheet Testing

Before the testing of spreadsheets can be done, it needs to be established what they should be tested for. In order to analyse this, a survey of possible risks was done by Panko and Halverson [45]. The resulting error-classification was later extended by Rajalingham et al. [46, 47] and Panko [44]. Following the classification proposed by Rajalingham et al., the copying of formula results is a Duplication error, by either the Developer or the Data Inputter. The definitions mention “re-creation of element of the model” (for the Developer) and “re-entering of data” for the Data Inputter. Since copying of data can, without stretching the definition of copying, be incorporated in both these definitions, copying seems a logical classification of the types of errors that we intend to find. The fact that the copying of formula outcomes is a high risk, stems from the definition of Rajalingham et al.'s Temporal Semantic errors. This type of error "is produced due to the use of data which has not been updated", which easily occurs when the inputs of a formula are changed and the results are not updated. The fact that it is easy to include the copying of formula outcomes into existing definitions of error-classes proves that it is useful to detect this kind of error, so users will be able to build better spreadsheets.

Having determined the types of errors the formula-copies are part of, a closer look can be taken at spreadsheet testing. In spreadsheet testing, there is a main type of testing based on the software programming paradigm of unit testing, besides which there are some minor other courses.

2.3.1 Unit Testing

One of the first challenges to face when considering unit testing is to determine the units to test. One of the important works concerning this is Erwig and Burnett's Unit-system [21]. Their Units are wrappers of a cell, where the wrapping is based on header inference and the contents of a cell. Along with these Units a formal grammar to describe spreadsheets is presented, that can be used to detect erroneous entries in cells. In this publication it is only mentioned that the system works on the basis of header inference, but how this header inference works is not elaborated upon any further. In order to perform this header inference automatically, Abraham and Erwig devised a method [1], based on a certain expectation of what a spreadsheet should look like. For example, one of these expectations was that it must be built as a collection of tables where each table has its own row- and column-headers. Using this approach they reasoned about logical errors within a spreadsheet with an automated tool in [2]. The logical errors this tool finds are determined to be errors by comparing the inferred unit of a formula cell with the calculated unit of this cell. Even though this is not the type of errors that originate from the copying of a formula outcome, the underlying system of header inference can be promising. The Units inferred by this technique can be used to check whether two cells that have the same data, and might thus have been copied, have equal Units. To be able to use this, some drawbacks of the automatic header inference will have to be overcome. One of the most important drawbacks is that the inference of Units heavily depends upon the textual representation of the headers of the data. If the users are unwilling to provide sensible headers, or purposely provide the copies with different headers, the Units will differ so the cells will not be marked as copies. The reverse also holds; when a user gives the same headers to multiple tables, while they are in fact not copies. To overcome this, the automatic method for header inference was extended by Chambers and Erwig [15] to dimension inference. While this alleviates some of the drawbacks, it adds a dependency on the textual meaning of the headers. This dependency is a drawback in and of itself because of the assumed underlying domain knowledge.

Ahmad et al. [3] propose another type of Units, which is related to the Units defined by Erwig and Burnett, but includes two extra types of relations between the Units and supports extra operations in the grammar. The Units described pass more validation tests on real-world spreadsheets than those of Erwig and Burnett, however, they assume header inference through manual annotation. This manual annotation removes the dependency of Erwig et al.'s textual representation and meaning, but is unwanted for a web-application. If the unwanted, extra user effort of annotating the headers is skipped, these Units will also suffer the same drawbacks as the Units of Erwig and Burnett and are therefore not useful for our algorithms.

For unit-testing, Antoniu et al. [4] have created XelDa, a stand-alone tool to analyse Excel-spreadsheets. The tool performs the same kind of checks that Erwig and Burnett do, so they can test for the same kind of errors. The difference lies in the way the units are derived; where Erwig and Burnett accomplish this based on the assumed structure of a spreadsheet, Antoniu et al. do this based on user annotations. While these user annotations give them the opportunity to report on errors if the calculated unit does not match the expected unit instead of the inferred unit, it also makes the approach unusable for us, because we seek to achieve an algorithm without user interaction.

2.3.2 Other Testing Methods

Rothermel et al. [48–50] have made an effort to use methods based in testing imperative programs, even though there are significant differences between the programming paradigm and spreadsheets. Since the devised methods rely on users to identify useful test cases, Rothermel et al. [22] have made an effort to automate this process. While this automation is a big improvement, the outcome of the test cases still need to be assessed by users. The script designed to perform this task worked reasonably well in a lab-setting, but the authors also note that users might very well assess the outcome differently than their script in a real-world situation. This means that for finding real errors the outcome must still be validated by users, which is a downside of this method. Another drawback for this effort is that it is based on the Forms/3 language [14] and not on general spreadsheets. Furthermore, [22] focuses on the correctness of the output of the formulas, whereas the copying of formula outcomes produces more structural errors.

Another testing approach that uses the formulas present in a spreadsheet is the interval-based testing methodology as proposed by Ayalew and Mittermeier [5–8,41]. This methodology however regards only the functionality of the formulas, while for this thesis the formulas are presumed to work correctly.

2.4 Spreadsheet Auditing

Besides testing, another way to detect errors in a spreadsheet is to audit it. One of the primary auditing tools for this thesis is Breviz, the tool built by Infotron. Breviz offers a framework that is able to read and analyse spreadsheets and to create a dataflow-diagram out of it [28]. The analysis works on the basis of a number of metrics [26, 29] derived from known code-smells in software systems. Breviz is now deployed as a stand-alone web-application, where users can upload their spreadsheets and receive an analysed version in return. This means that any addition to this tool should also run without needing any information about the spreadsheet from the user.

Another tool that can be used for auditing purposes is SheetDiff [16]. This tool is able to compare two spreadsheets, and present the cells that have changed between the two spreadsheets. Because the tool relies on the structural equality of the two evaluated spreadsheets it is not applicable here. It would seem that the tool is useful when a second version of the spreadsheet for which the formula copies must be found is created with some values altered. Cells that were equal before a change but not after it have been formulas, and cells equal to the original value of the changed formula have been copies of the original formula. While at first glance this seems to be a solution, there are two major problems. The first problem is that not every cell that has stayed the same is guaranteed to not have been a formula. The second problem is to find out what cells have the largest effect and need to be changed to affect a formula but not its copied outcome. Computationally, it is not viable to perform one cell-change at a time, because after a cell-change the complete spreadsheet must be re-evaluated and then compared to the original. Also, since it is unknown what cells are copies, when changing every cell the copies will also be changed, leaving them undetected.

Automated auditing of spreadsheets is also explored by Clermont and Mittermeier [17, 40]. They introduce the notion of logical areas, which are basically cells where a predetermined set of properties is equal. There are three sets of properties defined, where for this thesis the "copy equivalence", i.e. both the format and the formulas must be equal [41, 52], is relevant. Since we focus on finding copies of the formulas, this should be altered to equality of the outcome of a formula and a non-formula cell. The areas themselves are further supposed not to be more than d cells separated from each other. While the definition of these areas seems promising, upon close inspection it fails to meet the requirements. The areas enclosed by the definition all contain the same value, which, while it may make sense for formulas, is useless for the constant values that should be matched against the formulas. Also, as stated by Clermont and Mittermeier themselves, "The dangerous parts ... are those where irregularities in the geometrical pattern ... occur". Since we strive to find an algorithm that works for different kinds of spreadsheets, this difficulty must be overcome automatically, so different definitions will have to be found.

Noting that code inspection in traditional software engineering can be seen as a form of software auditing, Galette et al. [24] have had users analyse spreadsheets, almost like traditional code inspection. The main difference is in the fact that with code inspection there usually is a team-phase in which the findings of the individual group members are combined and discussed, which leads to a higher rate of found errors. Panko [43] also described this difference and has addressed this by adding group sessions to the inspection phases. The results of Panko's study show that group inspections indeed yield better results than individual inspections. The downside of both studies, however, is that user-inspections are used, which is something that cannot be automated for an online environment.

In addition to efforts to create new auditing tools, there have also been some surveys of existing auditing tools. One of these surveys is performed by Nixon and O'Hara [42] and concerns 5 auditing tools. While this survey addresses a number of possible errors that the tools should be able to find, the copying of formula-data is not one of them, so it remains unknown whether these errors will be found by these tools. Since the survey mentioned 5 auditing tools and these tools might have been updated in the meantime, the current features of the tools were compared. The tools surveyed were:

- Excel's built-in tool's Built in Auditing functions
- The Spreadsheet Detective[1]
- The Excel Auditor[2]
- The Operis Analysis Kit[3]
- Spreadsheet Auditing for Customs and Excise (SpACE)[4]

It is clear that Excel's built-in features do not provide any means to automatically find formula-copies and the SpACE program does not list any features, so the usefulness of these two packages has not improved. For the other three packages a comprehensive description can be found on the products' websites, but none of the tools provide any functionality to find the copies.

A second survey, of two tools, was performed by Davis [19]. Davis first explains the importance of data dependencies in spreadsheets, by explaining that these dependencies are basically the internal links that exist between cells based on their formulas. What he fails to recognise, however, are exactly the kind of dependencies that exist when formula-data is copied. He describes the dependencies by posing the question "what are the dependent cells of this cell?" while the formula-copies are those cells that do not answer his question, but are part of the answer of the question "what should be the dependent cells of this cell?". Given the questions asked for this survey and the conclusions, it is safe to say that both tools under scrutiny are unable to find formula-copies.

[1]`http://www.spreadsheetdetective.com/`

[2]`http://www.bygsoftware.com/auditor/auditor.htm`

[3]`http://www.operisanalysiskit.com/`

[4]`http://customs.hmrc.gov.uk/channelsPortalWebApp/channelsPortalWebApp.portal?_nfpb=true&_pageLabel=pageVAT_ShowContent&id=hmce_cl_001449&propertyType=document`

Chapter 3

Clone Detection - The First Approach

To find copy-paste relations within spreadsheets, these relations first have to be defined. In the initial approach to an algorithm, such a relation was defined to encompass the copying of every kind of cell. In other words, not only the copy-pasting of formula outcomes but also the copy-pasting of values had to be detected. To be able to detect this, first some definitions need to be defined. These definitions already were partially defined by others (e.g., Hermans [28]), and partly they are new for this thesis. Next, we present the algorithm itself and we explain its four steps in detail. Generally speaking, the algorithm detects clones by determining the data blocks within the spreadsheets and comparing these data blocks with each other. A number of settings can be changed according to specific needs of the data being tested, including minimum match percentage and minimum data block size. The reader should take note of the fact that the algorithm described here is **not** the final version of the algorithm. We added this algorithm because it served as a fruitful basis for discussions; as an inspiration for the final version and as a description of an approach that we feel is not viable. The reasons this algorithm was abandoned will be explained in the final part of this chapter.

3.1 Definitions

In order to detect copy-pasted data within a spreadsheet, we need to specify what copy-pasted data is. To do this, some definitions that will be used throughout this thesis are presented below. These definitions include basic elements like a cell and a worksheet, but also some elements used in previous research by Infotron, e.g., data blocks. After that, new definitions are introduced that are needed to reason about clones in spreadsheets, including definitions for the different types of clones that are distinguished by the algorithm.

3.1.1 Building Elements

Firstly, some definitions are given for elements of spreadsheets that are known to everyone working with spreadsheets.

Cell: A cell (C) is the basic building block of a spreadsheet.

Cell Type: The type of a cell, being either “Formula”, “Value” or “Empty”. The type of a cell can be found using a function T of type C $\rightarrow$ T.

Cell Value: The value that a cell contains. This can be either a string, a number, the result of the formula in the cell or nothing. It can be retrieved using a function V of type C $\rightarrow$ V.

Value Type: The type of the cell value, being either “Text”, “Number” or “Empty”. A function T of type V $\rightarrow$ T can be used to retrieve it.

Location: A tuple $\{$Column, Row$\}$ representing the position of a cell within a worksheet.

Column: A function *Column* of type Location $\rightarrow$ Column representing the column of the given location.

Row: A function *Row* of type Location $\rightarrow$ Row representing the row of the given location.

Worksheet: A worksheet is defined as the set that contains all cells that are located in the worksheet.

Spreadsheet: A spreadsheet is defined as a set that contains all worksheets contained in the spreadsheet.

Three definitions follow for elements presented in earlier work of Infotron [28]. The definitions themselves have been altered slightly to fit the purpose of this thesis.

Data Block: A data block (*DB*) is the smallest rectangle around a cell containing non-empty cells and bounded by empty cells or the borders of the worksheet. Data blocks can never be a part of other data blocks. A data block has two dimensions, $Width$ and $Length$, which can be requested using formulas of that name.

Data Block Location: A function L of type C $\times$ DB $\rightarrow$ L representing the relative location of a cell in a data block.

Precedents: Precedents P is a function of type C $\rightarrow$ $\{$C$\}$ representing all precedents of a given cell. Precedents are the cells that a formula refers to, including the precedents of those cells itself.

Having defined a data block as a rectangle around a certain cell, a set of the cells neighbours can be created, where the neighbours are automatically part of the same data block. The definition of this set is:

Neighbours: Neighbours N is a function of type C $\rightarrow$ $\{$C$\}$ representing all neighbours of a cell. The neighbours of a cell are those cells that have a location wherein either the Column or the Row differs only one unit from the location of the cell itself.

To detect clones, we introduce the notion of two cells c_1 and c_2 being **Copy Equal**. Intuitively, this means that the content of c_2 is the result of copy-pasting the cell value of c_1. Since text in spreadsheets is mostly used to provide meaning to the data (see also Abraham and Erwig [1]), copy equivalence is only defined for cells where the value type is a number.

Also, if c_1 contains a formula, then c_2 cannot be in the in the set of precedents of c_1. Note that this means that c_2 must always be a value cell, but c_1 can be a formula cell. For a formula cell to be equal to a value cell, the value of the — evaluated — formula needs to be copied to the value cell. The way in which this copying is performed, e.g., manually or through Excel's "paste special, values only" option, makes no difference. To be equal to another formula cell, both formulas must evaluate to the same value. Formally, copy equivalence ($\equiv$) is then defined as

Copy Equivalence: $c_1 : C \equiv c_2 : C \Leftrightarrow V(c_1) = V(c_2) \wedge c_1 \notin P(c_2) \wedge T(c_2) \neq \text{Formula} \wedge T(V(c_1)) = \text{Number}$.

3.1.2 Clone Types

As is the case for code clones in software engineering, there is also a minimum size involved when defining clones in spreadsheets. This means that a clone is always defined between two data blocks and not between individual cells. Using the definition of copy equivalence, a distinction can be made between different types of clones.

The first of these types is a **Total Clone**. A total clone occurs when two data blocks have copy equal cells at all locations and are of the exact same size. The formal description of a total clone is then:

Total Clone: $\forall c_A : C \in A, c_B : C \in B : L(c_A, A) = L(c_B, B) \wedge c_A \equiv c_B \wedge Length(B) = Length(A) \wedge Width(B) = Width(A)$

The second type is an **Edited Clone**. In the case of an edited clone, the dimensions of data block A and B do not need to be equal. This means a relative location is needed to define the location of a cell in data block B with respect to a location in data block A:

Relative Location: A relative location of a cell in data block B with respect to a location in data block A is defined as the tuple (r, s). This tuple gives the translation for both the columns and the rows to get from the location in data block A to the location in data block B.

Data block A is an edited clone of data block B when some of the cells on the same relative location are copy equal, or more formally:

Edited Clone: $\exists c_A : C \in A, c_B : C \in B : L(c_A, A) + \{r, s\} = L(c_B, B) \wedge c_A \equiv c_B$.

Since an edited clone does not need to encompass the total data block, a definition is needed for a container of the cells that are included in the clone. For this container a **Cell Block** is used, which always contains a subset of the cells in a certain data block and might include all cells in the data block. Formally, a cell block is defined as:

Cell Block: A cell block (CB) is a rectangle around a cell containing non-empty cells and is always part of a data block. Within a data block, cell blocks can be nested. As such, like a data block, it has two dimensions, $Width$ and $Length$.

In order to reason about this type of clones, an indication of how heavily the clone is edited is needed. For this indication, the measure **Edit Ratio** is introduced. This ratio is calculated as the percentage of copy equal cells with respect to all cells present in the clone, or, more formally:

Edit Ratio: A function ER of type A $\times$ B $\rightarrow$ ER where ER = $\frac{|c_A:C\in A \equiv c_B:C\in B|}{|c_A:C\in A|}$

The third basic type of clones is a **Transposed Clone**. A transposed clone exists if either a total clone or an edited clones exists between a transposed data block A (A^T) and data block B. Formally a transposed data block A^T is defined as:

Transposed Data Block: $\forall c_{AT} : C \in A^T, c_A : C \in A : Row(L(c_{AT}, A^T)) = Column(L(c_A, A))$ $\wedge Column(L(c_{AT}, A^T)) = Row(L(c_A, A))$

Since either a total clone or an edited clone can exist between data blocks A^T and B, the transposed clone can be combined with either of these two types, resulting in a **Transposed Total Clone** and a **Transposed Edited Clone**. The transposed total clone is formally defined as follows, using the definition of a transposed data block:

Transposed Total Clone: $\forall c_A : C \in AT, c_B : C \in B : L(c_A, A^T) = L(c_B, B) \wedge c_A^T \equiv c_B \wedge$ $Length(B) = Length(A^T) \wedge Width(B) = Width(A^T)$

Just as a transposed total clone can formally be described using transposed data blocks, the formal definition of a transposed edited data block is:

Edited Data Block: $\exists c_A : C \in A^T, c_B : C \in B : L(c_A, A^T) + \{r, s\} = L(c_B, B) \wedge c_A^T \equiv c_B$.

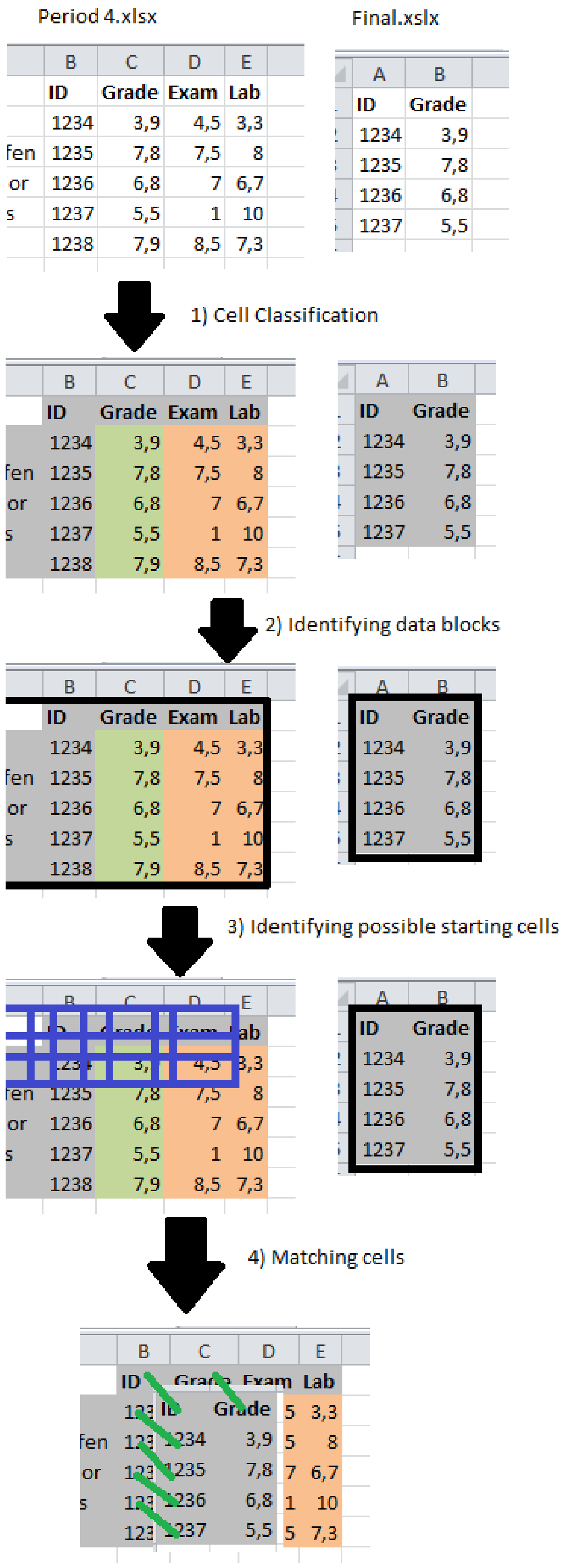
In total, this results in 4 types of clones that are distinguished:

- Total Clone
- Edited Clone
- Transposed Total Clone
- Transposed Edited Clone

3.2 Algorithm

Having formulated these definitions, an algorithm to detect any of these 4 types of clones can be created. This algorithm consists of a number of steps, as depicted by Figure 3.1. The pseudocode for this algorithm is given in Algorithm 1.

1. Cell classification. For each cell in the worksheet the type is determined.
2. Identifying data blocks. The data blocks in the different sheets are identified.
3. Identifying possible starting cells. All cells that can serve as possible starting points for a clone, based on the size of the smallest data block are identified.
4. Matching cells. All cells in the two data blocks are matched with the cells that have the same relative position to the first cell.


Period 4.xlsx
Final.xslx
1) Cell Classification
2) Identifying data blocks
3) Identifying possible starting cells
4) Matching cells

Period 4.xlsx

	B	C	D	E
	ID	Grade	Exam	Lab
	1234	3,9	4,5	3,3
fen	1235	7,8	7,5	8
or	1236	6,8	7	6,7
s	1237	5,5	1	10
	1238	7,9	8,5	7,3

Final.xslx

A	B
ID	Grade
1234	3,9
1235	7,8
1236	6,8
1237	5,5

Figure 3.1: Overview of the first clone detection algorithm

Algorithm 1 First clone detection algorithm

```
 1: Cell classification
 2: while $\exists c : C \in$ worksheet : $c \notin$ data block do
 3:     Find left-most upper-most cell not in a data block and start $DB$
 4:     repeat
 5:         Expand $DB$
 6:     until $\forall c : C$ surrounding $DB : T(c) =$ Empty
 7: end while
 8: Create a list $L \leftarrow \emptyset$ of possible clones
 9: for all $DB_A$ do
10:     Find all other data blocks $DB_B$ with corresponding dimensions
11:     for all $DB_B$ do
12:         Create cell blocks $CB_B$ if the dimensions still correspond to $DB_A$
13:         $L \leftarrow L \cup CB_B$
14:     end for
15: end for
16: Create a list $K \leftarrow \emptyset$ of clones
17: for all $CB_B \in L$ do
18:     Create clone $CL \leftarrow \{c_1\} : c_1$ is the left-most upper-most cell of $CB_B$
19:     for all $c : C \in DB_A$ do
20:         Calculate $\{r, s\}$ with respect to the left-most upper-most cell in $DB_A$
21:         Find $c_B : C \in CB_D$ using $\{r, s\}$.
22:         $CL \leftarrow CL \cup c_B$
23:     end for
24:     Calculate edit ratio of $CL$ and $DB_A$
25:     $K \leftarrow K \cup CL$
26: end for
27: return $K$
```

3.2.1 Step-by-Step Explanation

1. Cell Classification To correctly determine whether two cells are copy equivalent, it is necessary to determine the cell type of every cell. Breviz, the tool upon which this algorithm was built, provides this functionality, as explained in Hermans [27, 28]. The cell types that are distinguished here differ slightly from the cell types we use in the rest of this thesis, as the cell types found are “Data”, “Label”, “Formula” and “Empty”. “Data” and “Label” are both part of the cell type “Value” that is used during this thesis. A distinction is made based on the value type of the cells, but this distinction is slightly different from the one used in Breviz, since both “Data” and “Label” can have value type “Text” and “Number”. For a more detailed explanation of the algorithm and the distinguished types we refer the reader to Hermans [27, 28]. Since the types determined by Breviz are slightly different from the types used throughout this thesis, we transformed the types so that only the three cell types used in this thesis result from this step.

2. Data Block Identification The second step consists of determining the data blocks that are present in a worksheet. These data blocks are found by looking for the left-most upper-most cell of a spreadsheet that is not yet contained in a data block. Starting from this cell, the data block is expanded horizontally and vertically, at all times remaining a rectangular shape. For this expansion, the neighbouring cells of those cells that are currently at the edge of the data block are continuously added to the data block, until such a neighbouring cell is empty. This step is detailed in pseudo code in the lines 2 to 7 of Algorithm 1. As said with the presentation of the different clone types, the clones are based on data blocks rather than on individual cells. This is done to prevent too many clones being found when a certain number occurs multiple times. This is reflected in papers on code clone detection (see also Roy and Cordy [51]), where a clone generally only exists for a number of tokens instead of for a single token. As with the cell classification, Breviz provides the functionality to extract the data blocks from worksheets, which is explained in detail in Hermans [28].

3. Starting Cell Identification For the third step (see lines 8 to 15 of Algorithm 1), the initial cells in a data block need to be found, along with the other data blocks that could possibly result in clones. For each data block (DB_A) in the worksheet, every other data block (DB_B) that has the corresponding dimensions is checked. According to the definitions of the total and edited clones, the corresponding dimensions in this case mean that DB_B must be at least as DB_A, in both Length and Width. After creating a list of data blocks with the corresponding dimensions, for every data block DB_B that is larger than DB_A cell blocks (CB_B) are created. For all created cell blocks CB_B with the corresponding dimension (cf. the corresponding dimensions of a data block) a possible clone is created by adding the top left cell as starting cell for the clone. In order to find cells in the cell blocks that are the basis of the possible clones on the same relative location as the other cells in DB_A, C_A is also the top left cell of DB_A. For these starting cells (C_B) a match is calculated by comparing the value $V(C_B)$ with the value of the starting cell C_A of data block DB_A. This comparison is an implementation of the copy equivalence definition. Since for edited clones not all cells need to be copy equal, the possible clones without copy equal starting cells are not discarded.

4. Cell Matching With the possible clones found, in the fourth step of the algorithm we can expand these clones, as done in lines 16 to 27 of Algorithm 1. For each cell C_A in the original data block DB_A, the relative position to the top left cell, i.e. the starting cell of DB_A, is calculated. This relative position is then used to find the corresponding cell C_B in the cell blocks of the possible clones. This corresponding cell C_B is then compared to C_A using the same implementation of copy equivalence as was used in the third step. When this is done for all cells C_A in the original data block DB_A, the result is a list of all clones for this data block. For every clone the edit ratio can then be calculated and presented to the user.

To be able to support transposed clones, some adaptations needed to be made to this algorithm. In the third step, the dimensions of cell and data blocks (CB_B and DB_B) are checked, making sure that the Width and Length of these blocks are at least as large as the Width and Length of the original data block DB_A. For transposed clones, the Width and

Length of CB_B and DB_B are compared to the Length and Width of DB_A. The starting cell of the different blocks remains the top left cell of the block. In the fourth step, cells in CB_B are found on the same relative location as the current cell in DB_A. Dealing with transposed clones, the coordinates of the relative location of the current cell in DB_A are inverted to find the corresponding cell in CB_B. The other steps of the process all remain the same.

3.3 Optimisations

In this algorithm, there are a number of ways to perform some optimisations. Some of these optimisations are implemented as options, so the user of the algorithm can directly influence its output, for example by changing the edit ratio. Other optimisations, e.g., direct calculation of the edit ratio, have been implemented to speed-up the algorithm. A third category of optimizations has not yet been implemented and we consider them to be future work, for instance changing the edit ratio to an edit distance.

3.3.1 Options

Some options are included in the algorithm to be able to control the number of clones found. The first of these options is the minimum size of a data block. This minimum size is given as the minimum number of non-empty cells that must be present in the data blocks for which to find the clones. Secondly, to control the number of clones found regardless of the size of the originating data block, a limit can be set on the edit ratio. This limit is a maximum ratio, given as a percentage, to prevent cell blocks with too many differences to the original block being registered as a clone. Using these two options proved insufficient when the algorithm was tested on data sets that included a large number of empty or blank cells. To support this type of spreadsheets, an option was included to provide a threshold for a maximum number of blank cells in a data block. To correctly handle data blocks of different sizes, this threshold is given as a percentage of the total number of cells.

3.3.2 Implemented Optimisations

The first of two implemented optimisations concerns the discarding of clones. During the fourth step of the algorithm, cells are continuously added to a clone in the list of possible clones. With every cell that is added, the total edit ratio for that clone either increases or decreases. Having set a minimum value for this edit ratio, and knowing how many cells are in the original data block, the minimum number of cells in CB_B that must be copy equal to a cell in DB_A can be determined. And vice versa, the maximum number of cells in CB_B that is not copy equal to a cell in DB_A can also be calculated. After each addition of a new cell to the clone, this maximum can be compared to the current number of non copy equal cells that is present in the clone. When there are more cells in the clone present that are not copy equal than is allowed, the clone can be disregarded altogether, thus improving the runtime of the algorithm.

The second implemented optimisation originates from the field of software engineering. Ducasse et al. [20] present a technique to store the results of previous comparisons in a matrix, and to use this matrix as a lookup to substitute a lengthy number of comparisons. This technique was also implemented in the algorithm by storing the cell block and corresponding edit distance for every found clone. Thus, when a clone with a cell block CB_B had been found, the clones that existed for the data block DB_B CB_B resides in could also be added as edited clones. When CB_B happened to be equal to a data block DB_B all total clones of this data block could also directly be added as total clone for the current data block DB_A.

3.3.3 Future Work

There were two optimisations conceived that were not implemented in the algorithm and that we considered to be future work. The first of these two was a fingerprint checking algorithm, comparable to the fingerprinting explained by Johnson [31]. If a fingerprint could be constructed for a data block, two data blocks could be compared by fingerprint. This would lead to a fast algorithm for finding the total clones in a spreadsheet, without having to compare every cell. To be exceptionally effective, a fingerprint would have to be devised that would allow for some indication of how far removed in terms of edit distance the two cell blocks are. If such a fingerprint is created, it can be used to quickly disregard the cell blocks that are too far removed.

The second optimisation was the transformation of the edit ratio to a measure that more closely resembled the Levenshtein Distance [37]. Since the Levenshtein Distance is considers the number of changes needed for one string to mutate into the other, we wanted the edit ratio to reflect this. As the measure would then change to a distance instead of a ratio, a new definition was deemed necessary, the **Edit Distance**. The new definition of an edit distance is:

Edit Distance: The number of blocks of adjacent cells needed to change in order to mutate one cell block into the other.

A block of adjacent cells is counted as 1 edit here, because spreadsheet tools like Excel provide easy functionality to edit adjacent cells with one click or keystroke. For non-adjacent cells more actions are needed, leading to the conclusion that this is done on purpose and therefore should count as more edits.

3.4 Abandonment

As stated before, the algorithm described above has been abandoned. For this abandonment there are a number of reasons. One of the three main reasons was the fact that we found that there are more types of clones in spreadsheets we would like to be able to detect. In and of itself the failure to detect certain kinds of clones was not enough of a reason to abandon the algorithm, but combined with the large runtime and inspiration for another approach it was. This other approach is what the final algorithm was based upon, so we refer the reader to the next chapter for an in-depth explanation.

3.4.1 Clone Types

The clones that can be detected using the described algorithm are clones of which the values can possibly be changed, but the order of the values must have remained equal. Since the detected clones all have an analogy in software engineering, this makes perfect sense, because in software engineering changing the order of computations potentially has serious consequences. In a spreadsheet, on the other hand, changing the order of the entries by sorting a table slightly different, does not change the meaning of the values, only the way in which the user perceives the data. Because the meaning of the data does not change, it is possible for users to sort a column by ascending values one time, and later sort the column by descending values. The current implementation will fail to notice the clone in at least one of the two situations and possibly in both, when one half of the clone is not sorted at all. Since the meaning of the values does not change when they are ordered differently, a new type of clone should be defined that allows for two cell blocks to be equal when they contain cells that are copy equal, without having the same location. This new type of clone is an **Ordered Clone**:

Ordered Clone: $\exists c_A : C \in A, c_B : C \in B : c_A \equiv c_B$.

This definition can be altered to contain all cells in A, or cells in A^T for the total resp. transposed clones, leading to a total of 8 types of clones. Of these 8 types of clones, 4 are exactly equal to the ones described above. The other four are almost equal, but lack the restriction on equal relative locations. The restrictions on the sizes remain for the total clones, because the size of a column does not change when the ordering is changed.

3.4.2 Runtime

A second large downside of this algorithm, and thus one of the reasons this algorithm was abandoned, was its runtime. With the described improvements to the algorithm, it took the algorithm over half an hour to process a random selection of 24 sheets taken from the EUSES corpus [23]. Discussing several ideas on how to improve this runtime we came up with the optimizations described above. After implementing two of them, we still found a large, though a bit improved, runtime of over half an hour. Combining this with the fact that certain clones were not found, we started to consider a completely different approach for clone detection in spreadsheets. Since this approach would be able to detect the ordered clones, we decided to implement this new approach in a new algorithm and compare its runtime results with this algorithm. When it became apparent that the new approach (as described in Chapter 4) was significantly faster than this approach, using six and a half hours for the complete corpus consisting of almost 4500 files, we abandoned this algorithm and further developed our second algorithm.

Chapter 4

Copy-Paste Relations - The Second Approach

Our second approach to an algorithm has been influenced strongly by Johnson's work [31] on code clone detection. His techniques of fingerprinting and keeping a lookup table have been implemented here, with satisfactory results. Applying these techniques allowed us to reinvent our definitions and to design a method that would overcome the shortcomings of our previous attempt. Having developed an algorithm that performed according to our expectations, we also had to consider ways to report the results to the user. For this feedback we extended Breviz' existing feedback system and we found a new way to do present this.

4.1 Definitions

For the algorithm described here, most of the definitions as they are described in Section 3.1 are also used. Of these definitions, some are reused exactly as they were, while others needed to be redefined, which is done below. Also, a small number of new definitions is needed to reason about this algorithm, which means they are introduced as well.

4.1.1 Previously Defined

Of the previously defined definitions for the building elements of a spreadsheet, only the definition of **copy equivalence** is altered for this algorithm. To illustrate the difference, the definition of copy equivalence was given as:

Copy Equivalence: $c_1 \equiv c_2 \Leftrightarrow V(c_1) = V(c_2) \wedge c_1 \notin P(c_2) \wedge T(c_2) \neq \text{Formula} \wedge T(V(c_1)) = \text{Number}$.

This definition meant that no restriction was put on the first cell, as long as its value type was a number. For this new algorithm, a restriction on the type of the first cell is added, for a number of reasons. The first reason has to do with the runtime of the algorithm. Restricting the type of cells that are eligible to be part of a clone, means that there are (possibly) fewer cells part of such a clone. Having fewer cells that are part of a clone, means the algorithm will be able to compute the clones that are present in a spreadsheet in less time. Since the algorithm is designed to be used as part of a web-application, this speed is an important factor.

The second reason we believe this restriction is justified is because of the nature of a copy-paste relation. Generally speaking, there are two kinds of copy-paste relations; the first is one where the value of a value cell is copied to another cell, the second one is where the value of a formula cell is copied. In both of these cases it is obvious that when the origin of the data is altered, its copy should also be altered. The most dangerous of these two is however when a formula cell is copied, because the outcome of a formula can change without the user knowing it has changed. When a user alters the value in a value cell, he or she *knows* he has altered that cell, but he or she possibly does not know that some formula depends on that cell and therefore also changes. If a user does not know a formula has changed, he or she will most definitely not know a copy of this formula should also change. In a way, this kind of copy-paste relation creates an even more invisible link between two cells than the copying of a value cell does, so in this new algorithm we only look at the copying of formula cells. The definition of copy equivalence needs to change according to this restriction, and becomes:

Copy Equivalence: $c_1 \equiv c_2 \Leftrightarrow V(c_1) = V(c_2) \wedge c_1 \notin P(c_2) \wedge T(c_2) \neq \text{Formula} \wedge T(V(c_1)) = \text{Number} \wedge T(c_1) = \text{Formula}$.

This extra restriction does not reduce the need for the restriction of the value type of cell c_1 to be a number, because there are also formulas that manipulate text, e.g., formulas for string concatenation.

Some definitions that were defined for the previous algorithm are not needed for this version. Due to the different approach we have taken here, there is no longer a need for the definitions of a **data block** and a **data block location**. The clone types for **total clone** and **edited clone** are not needed as such, but new objects closely resembling them are used. Not having a definition for an edited clone means that the definition for a **relative location** is also superfluous, as is the definition of a **cell block**. For a cell block however there is a new object closely resembling a cell block. Without a definition for a data block, there can obviously not be a definition for a **transposed data block**, without which **transposed clones** cannot exist. Being one of the reasons the first algorithm was abandoned, **ordered clones** are expected to be found by the algorithm described below. Though the type of clone as such does not exist, thus having no definition, this is indeed the case, as will be explained below. To accompany the new version of the edited clones, the definition for **edit ratio** has been preserved.

Since a data block location is no longer needed in this version of the algorithm, the definition of a **Location** can be slightly altered to:

Location: A function L of type C $\rightarrow$ $\{$Column, Row$\}$ representing the position of a cell in a worksheet.

4.1.2 New Definitions

For the algorithm described here, some new definitions need to be clarified. Intuitively, the algorithm tries to find a formula cell whose value is copied to another cell, the very basis of a copy-paste relation. From this intuitive definition, it follows that such a relation always has two sides; a formula cell as origin and a value cell on the other side. These two sides of such a relation together form a tuple, that we refer to as a **Clone**. The definition of a clone thus becomes:

Clone: A tuple $\{c_1, c_2\} : T(c_1) = \text{Formula} \wedge T(c_2) = \text{Value}$.

This definition is different from what we used in the previous algorithm, where we stated that "a clone is always defined between two data blocks and not between individual cells" (cf. Section 3.1.2). Following the same reasoning used for the first algorithm, a minimum size is required, so a definition is needed for a group of cells that are part of a clone. Such a group of cells is called a **Clone Cluster**. More precise, the definition of a clone cluster is:

Clone Cluster: A set $\{CC\}$ where $\forall c_1 : C \in CC, c_2 : C \in CC : T(c_1) = T(c_2) \wedge$ $(T(c_1) = \text{Formula} \vee T(c_1) = \text{Value}) \wedge \exists Cl_1 : \text{Clone} \ni c_1 \wedge \exists Cl_2 : \text{Clone} \ni c_2$

From this definition it follows that all cells in a clone cluster must have the same cell type, either all being a formula cell or all being a value cell. The definition also states that a cell can only be part of a clone cluster when the cell is part of at least one clone. Following from this definition, a clone cluster has a specific type, based on the type of cells present in the cluster:

Clone Cluster Type: A function T of type CC $\rightarrow$ T representing the type of cells present in the clone cluster, being one of "Formula" or "Value".

Having definitions for clone clusters, the equivalents for a total clone and an edited clone must be considered. According to the definition in Section 3.1.2, the two data blocks of a total clone must have copy equal cells at all locations and they must be of the same size. Since clone clusters are used in this algorithm instead of data blocks, the definition needed to be altered. To avoid confusion and to use consistent naming, there is now a definition for **Matching Clusters**:

Matching Clusters: $(\forall c_1 : C \in CC_1, \exists c_2 : C \in CC_2 : c_1 \equiv c_2) \wedge |CC_1| \leq |CC_2|$

In plain English, this means that two clusters match if they contain the same values. There is no restriction on size, which means that this definition in theory allows for more clones to be found. Having an equivalent to a total clone as it was defined for the first algorithm, an equivalent for the edited clone is needed and defined here. Equivalent to terminology used in code clone detection, clusters that do not all contain the same values are named **Near-miss Clusters**. The formal definition of such clusters is:

Near-miss Clusters: $\exists c_1 : C \in CC_1, c_2 : C \in CC_2 : c_1 \equiv c_2$

Needing at least one clone with a cell in both clusters, there are possibly a large number of near-miss clusters. Using the edit ratio as defined for the previous algorithm allows reasoning about different near-miss and matching clusters. For the sake of completeness, the definition of the edit ratio was:

Edit Ratio: A function ER of type A $\times$ B $\rightarrow$ ER where ER = $\frac{|c_A : C \in A \equiv c_B : C \in B|}{|c_A : C \in A|}$

In this definition both A and B are clone clusters, where A is the smallest cluster of the two.

4.2 Algorithm

Our new algorithm consists of a number of steps, depicted in Figure 4.1 and described in pseudo code in Algorithm 2. The steps of the algorithm are:

1. Cell classification. For each cell in the worksheet the type is determined.
2. Lookup creation. A lookup table is created with the locations of each value.
3. Pruning. The lookup table is emptied of values with a single location.
4. Cluster finding. Clusters are created with cells that are contained in a clone.
5. Cluster matching. The created clusters are matched and outputted to the user.

These five steps are added to the existing algorithms of Breviz, the tool created by Infotron. This means that, as previously stated, the algorithm was designed to be part of a web-application and was therefore required to be as responsive and fast as possible. In addition, care had to be given to the output that was reported to the user, since he or she is able to use the algorithm without the option of direct support from one of its designers.

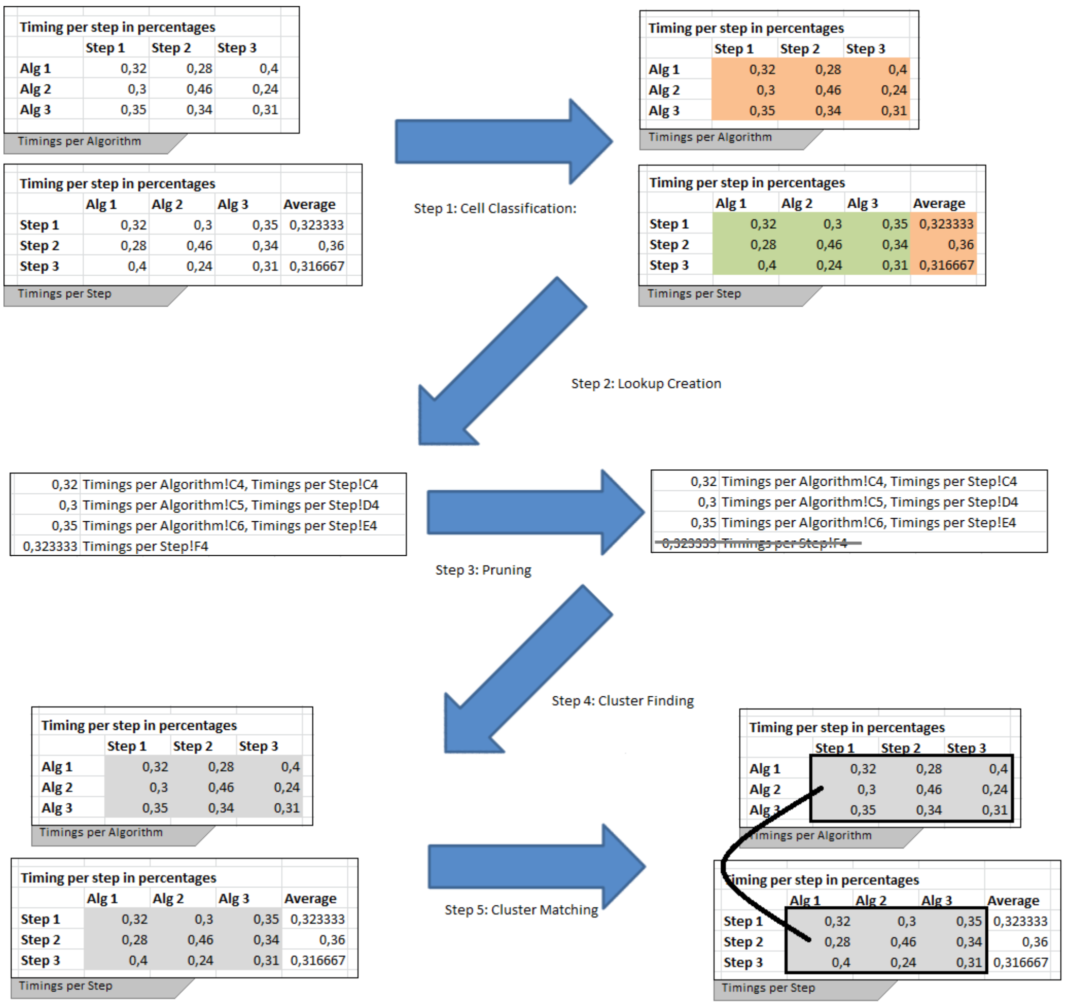
4.2.1 Step-by-Step Explanation

1. Cell Classification The first step of the algorithm, described in pseudo code in line 1 of Algorithm 2, is equal to the first step in the first version of the algorithm. For the sake of completeness a small description is given here as well, even though a more detailed explanation has been given in Section 3.2.1. Breviz provided the largest part of the functionality to classify the cells of the spreadsheets, as detailed in [27, 28]. Since we use slightly different types for this algorithm than are outputted by Breviz, a transformation is performed and the final output consists of cells with a cell type being one of “Formula”, “Value” or “Empty”. In Figure 4.1, after the first step the cells with type “Value” are marked green and those with type “Formula” are marked orange. To avoid confusion, only those cells that have a value type “Number” are marked in the figure, even though this is not wholly correct in this instance.

2. Lookup Creation As seen in lines 2 to 11 of Algorithm 2 and in Figure 4.1, in the second step of the algorithm a lookup table is created. This lookup table takes the cell values as keys and a list of cell locations as values. The idea to create such a lookup table is taken from Johnson’s algorithm [31] for code clone detection. Where Johnson creates a table using fingerprints of lines of code, we use the value of a cell as its fingerprint. This lookup table is initially empty, and for every cell that is encountered the table is extended, by either adding the cell location to the list of location for a value that already existed, of by adding a new entry to the table.

Algorithm 2 Final clone detection algorithm

```
1: Cell classification
2: Create lookup table $LT \leftarrow \{\varepsilon, \emptyset\}$
3: for all $c : C \in$ worksheet do
4:     if $V(c) \in LT$ then
5:         $\{V\} \leftarrow V(c) \subseteq LT$
6:     else
7:         $\{V\} \leftarrow \emptyset$
8:     end if
9:     $V \leftarrow V \cup L(c)$
10:    $LT \leftarrow LT \cup V$
11: end for
12: Create a list $FC \leftarrow \emptyset$ of formula cells and a list $VC \leftarrow \emptyset$ of value cells
13: for all $\{V, \{C\}\} \in LT : T(V) = \text{Number} \wedge |\{C\}| > 1$ do
14:     if $\exists c_1, c_2 \in \{C\} : T(c_1) = \text{Formula} \wedge T(c_2) = \text{Value}$ then
15:         for all $c \in \{C\} : T(c) = \text{Formula}$ do
16:             $FC \leftarrow FC \cup c$
17:         end for
18:         for all $c \in \{C\} : T(c) = \text{Value}$ do
19:             $VC \leftarrow VC \cup c$
20:         end for
21:     end if
22: end for
23: Create a list $\{CC_F\} \leftarrow \emptyset$ of clone cluster of type formula and a list $\{CC_V\} \leftarrow \emptyset$ of clone
    cluster of type value
24: for all $c \in FC \cup VC$ do
25:     Create clone cluster $CC \leftarrow \{c\}$
26:     if $c \in FC$ then
27:         Expand $CC$ using $FC$
28:         $CC_F \leftarrow CC_F \cup CC$
29:     else
30:         Expand $CC$ using $VC$
31:         $CC_V \leftarrow CC_V \cup CC$
32:     end if
33: end for
34: Create a list $R \leftarrow \emptyset$ of clone clusters
35: for all $CC_1 \in CC_F$ do
36:     for all $CC_2 \in CC_V$ do
37:         if $ER(CC_1, CC_2) >$ threshold then
38:             $R \leftarrow R \cup \{CC_1 \cup CC_2\}$
39:         end if
40:     end for
41: end for
42: return $R$
```



Timing per step in percentages

	Step 1	Step 2	Step 3
Alg 1	0,32	0,28	0,4
Alg 2	0,3	0,46	0,24
Alg 3	0,35	0,34	0,31

Timing per step in percentages

	Alg 1	Alg 2	Alg 3	Average
Step 1	0,32	0,3	0,35	0,323333
Step 2	0,28	0,46	0,34	0,36
Step 3	0,4	0,24	0,31	0,316667

0,32	Timings per Algorithm!C4, Timings per Step!C4
0,3	Timings per Algorithm!C5, Timings per Step!D4
0,35	Timings per Algorithm!C6, Timings per Step!E4
0,323333	Timings per Step!F4

0,32	Timings per Algorithm!C4, Timings per Step!C4
0,3	Timings per Algorithm!C5, Timings per Step!D4
0,35	Timings per Algorithm!C6, Timings per Step!E4
~~0,323333~~	~~Timings per Step!F4~~

Figure 4.1: Overview of the final clone detection algorithm

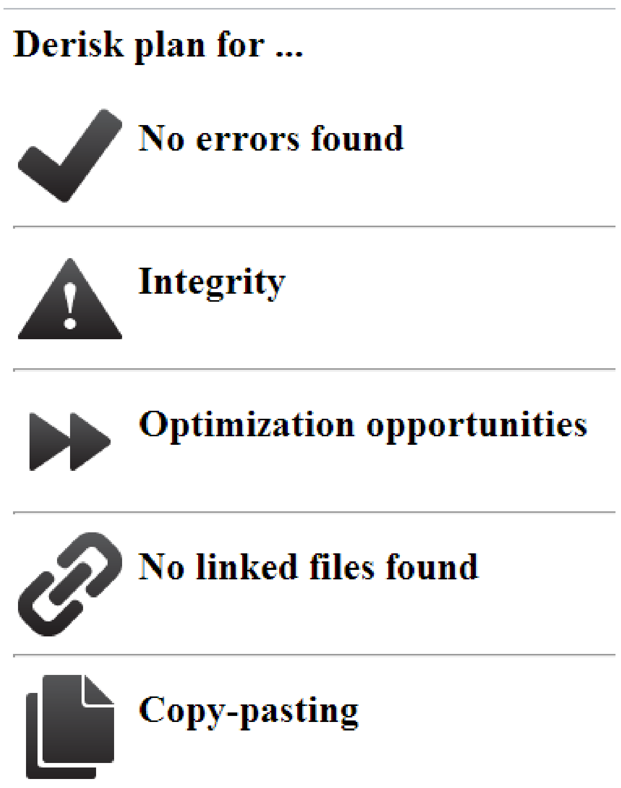
3. Pruning During the process of pruning, detailed in lines 12 to 22 of Algorithm 2, it is firstly ensured that only those cells of which the value is of type "Number" are considered. The second part of the pruning process is again inspired by Johnson's algorithm. Johnson removes all fingerprints from his lookup table that only occur once in the source files. Analogous to this, only those cells with values that occur multiple times are kept as eligible for clone creation. When this selection of cells has been made, for each cell value the algorithm checks the list of locations. The cells at the locations that are in the table entry are inspected and when both a cell of type "formula" and a cell of type "value" have been found, all cells in the entry are added to a list of either formula cells of value cells, depending on their respective types. Despite the way the algorithm is shown in Figure 4.1, the filtering for only cell with a value type "Number" is done here, and not in the first step. The pruning of the list so that only values that occur multiple times in the spreadsheet remain is depicted here by erasing the value $0,323333$ from the table.

4. Cluster Finding With the two lists resulting from the previous step, clone clusters are created. The process of creating these clusters is described in lines 23 to 33 of Algorithm 2 in pseudo code. For each cell that has been added to one of the two lists of cells, a clone cluster is created, provided the cell does not already belong to a clone cluster. When a clone cluster of one cell is created, the cluster is expanded to try to maximize its size. Where the expansion in the previous algorithm only worked downward and to the right, which was possible given that the algorithm always started in the left-most upper-most cell, and went on until it hit an empty cell, it works slightly different here. As was stated in the definition of a clone cluster, all cells in a cluster must have the same type. Since the algorithm started with a cell from a list that is known to contain only cells of the same type, the expansion is performed using this list. The expansion of a clone cluster works by looking at the cells that are currently present in the cluster. For each cell that is present, its neighbouring cells are looked up. If the neighbouring cells are not yet present in the clone cluster, but they are present in the list, they are added to the clone cluster. It is necessary here to check whether the neighbouring cells are present in the list, because they are only allowed in the clone cluster if they not only have the correct type but also are part of a clone, the two conditions based on which a cell is added to the list in the third step of the algorithm. As can be seen in Figure 4.1, the result of this step is block of neighbouring cells that are all of the same type, i.e. a cluster.

5. Cluster Matching In this final step, the clone clusters are compared with each other using the edit ratio as defined above (cf. lines 34 to 42 of Algorithm 2). Each clone cluster containing formula cells is compared to each clone cluster containing value cells and the edit ratio between those two clone clusters is calculated. Because during this step the location of the values within the clusters is irrelevant, a different ordering of the values in the two clusters will result in the same error ratio as when the two clusters were ordered the same. This means that the ordered clones as defined in Section 3.4.1 will also be found using this algorithm. If the edit ratio exceeds a given threshold, the clone clusters have enough commonalities to mark them as a copy-paste relation and present them to the user. Setting this threshold to 100% results in the algorithm only finding those copy-paste relations where the edit ratio between the two clone clusters is 100%, meaning that near-miss clone cluster will not be reported. The match between the two clusters is shown in Figure 4.1 with a curved line between the two clusters.

4.2.2 Options

The threshold that can be set for the edit ratio is one of the options the user can control himself, just as it was an option in the first algorithm (cf. Section 3.3.1) and called the **Minimum Edit Ratio**. In addition to the ability to control the minimum edit ratio, the user has a number of other options at his disposal to control the output. In our first algorithm, there was an option to control the minimum size of a data block. An equivalent of that parameter has been introduced in this algorithm, called **Minimum Cluster Size**. With this



Derisk plan for ...

No errors found

Integrity

Optimization opportunities

No linked files found

Copy-pasting

Figure 4.2: Example report as outputted by Breviz

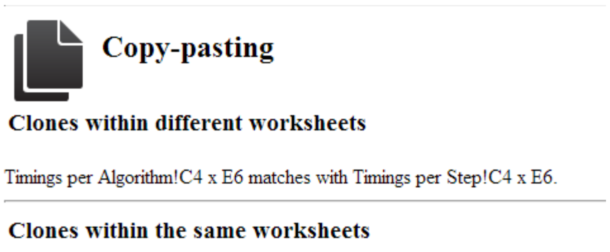
parameter, the outcome of the fourth step of the algorithm can be altered; since only those clone clusters are returned that have a number of cells that is minimally as large as the value of this parameter.

Another way to influence the fourth step is by changing a parameter called **Step Size**. This parameter influences the expansion of clone clusters by changing the search radius for neighbours. Setting this parameter to 1 means that only direct neighbours of a cell in a clone cluster are considered for addition to the clone cluster, thereby resulting in no gaps in a cluster. Increasing it, however, leads to possible gaps in a cluster, since not only the direct neighbours of the cells are found, but also cells being further away from the current cell, where the step size controls how far away such a cell may maximally be.

The third way to influence the fourth step is by changing a parameter that controls the minimal number of different values that must be present in a clone cluster. This parameter is dubbed **Minimal Different Values** and was introduced because we believe that if there are large clone clusters present with a very small number of different values, these might not be very interesting to the user in terms of copy-paste relations.

4.3 Visualisation

After the copy-paste relations have been detected, the results need to be presented to the user. Since the user is able to run the algorithm on his own, without support or a further explanation, the output of the algorithm needs to be presented in a way that the user is able to use. This comprehension of the output is aided by providing the output in a way Breviz also uses to output the results of its other parts. This means there are two ways the output is presented to the user: in a textual report (cf. Figure 4.3) and in a data flow diagram (cf. Figure 4.4) as is also done by Hermans in [29]. Besides this visualisation, another way of feedback has also been thought-out, but remains to be future work. This new form of feedback involves representing the output in the spreadsheets themselves.



Copy-pasting

Clones within different worksheets

Timings per Algorithm!C4 x E6 matches with Timings per Step!C4 x E6.

Clones within the same worksheets

Figure 4.3: Detail of Copy-pasting section of the report

4.3.1 Extending Breviz

The current implementation of Breviz allows for two types of feedback. Firstly, a textual report is presented after the analysis of a spreadsheet, with different section for the different parts of the analysis. An example of such a report can be seen in Figure 4.2.

It is beyond the scope of this thesis to explain all the different sections of the report, but we refer the interested reader to Hermans [26, 29]. The section added by this algorithm is the final section, titled Copy-pasting, will be explained here in detail. For this detailed description, a detailed example of the report is given in Figure 4.3. The first thing to notice is that the section is divided into two parts; one part indicating the clones occurring over different worksheets, the other for the clones occurring in the same worksheet. This difference is created to aid the user in understanding the outcome. Copy-paste relations occurring over multiple worksheets are often harder to notice than such relations within the same file, if only because Excel does not provide functionality to easily view two worksheets at once. A second reason for splitting the output is to enable the user to retain a comprehensive view of the results. Especially when a large number of copy-paste relations exist in a spreadsheet, a user is likely to become confused when all results are presented as a long list and the user does not know where to start correcting the relations. Splitting the output in this case possibly reduces the size of the list, so the users can focus more on the relations he wishes to solve. Within the two parts, the clones are textually represented by the first and last cell of both clone clusters that are part of a copy-paste relation. If there are multiple clones to be reported in a section, the ordering of the clones can be slightly influenced by the user, depending on what he or she feels is appropriate for his situation. The basic sorting is by edit ratio, where the copy-paste relations with the lowest edit ratio are presented first, i.e. matching clusters are presented before near-miss clusters.

In addition to the report, the second method of feedback Breviz provided were data flow diagrams. In the existing diagrams, the worksheets of a spreadsheet were visualised as rectangles and the links between the worksheets as arrows. The arrows used to indicate a link consisted of a solid line. To indicate that a copy-paste relation has created a different kind of link between two worksheets, we used dashed lines to create the arrows for these relations. An example of what this looks like can be seen in Figure 4.4.

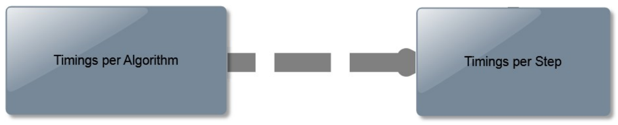


Figure 4.4: Example of a data flow diagram

4.3.2 Future Work

Feedback that has to be considered as future work concerns the feedback that can be given within a spreadsheet. Breviz is already capable of outputting a spreadsheet where the cells are coloured according to different criteria, e.g., cells that share a common formula are given a common colour. We propose as future work to create such a spreadsheet, where the different copy-paste relations are coloured and annotated. Using this form of feedback, the user gets a new version of his own spreadsheet in which he can directly see what copy-paste relations are present. The clone clusters that contain value cells should be annotated with a note stating the origin of the data, while the clone clusters containing formula cells should have a note regarding the copy or copies of the data. Using this kind of feedback, the user is more likely to be able to improve on the design of the spreadsheet by changing the copy-paste relation to a link.

Chapter 5

Categorisation of Copy-Paste Relations

As shown in Figure 4.3 the result of our algorithm is printed in a report as a list of copy-paste relations. However, this list turned out to be quite long in some instances. When this list is too long, a user is inclined to lose track of what he is supposed to change to his spreadsheet to improve its design. This is the reason some distinctions needed to be made between the different relations. A very coarse distinction has already been introduced before and concerned the location of the relation, i.e. whether the concerning clone clusters were located within one worksheet or not. In this chapter we introduce a number of other methods to distinguish between different copy-paste relations. We relate this distinction to the different types of code clones described in Section 2.2.

5.1 Clone Cluster Categories

As identified in Section 2.2 there are a number of categories for copy-paste relations that have some analogy to categories defined for code clone detection. The different categories were defined as Types$_S$ I – III, and are given here again as a reminder:

Type$_S$ I	Identical values and formatting
Type$_S$ II	Identical values
Type$_S$ III	Modified values:

a) One-cell difference
b) Multiple one-cell differences
c) Multiple differences.

5.1.1 Explanation per Type$_S$

These Types$_S$ are defined in Section 2.2 without paying attention to the implementation of the algorithm used to find the copy-paste relations. Since we have defined an algorithm that uses clone clusters, these descriptions need to be rewritten in terms of these clone clusters before they are useful.

Types$_\mathrm{S}$ I The first type was described as a copy-paste relation wherein all cells have identical values and formatting. In our algorithm, a copy-paste relation is defined as a matching between two clone clusters, so the description of this first type changes to use these clone clusters. Since the values of the cells are already incorporated in the definition of a clone cluster, the description changes to: "Two clone clusters where for every cell in the smallest cluster there is a cell in the other cluster with the same value and formatting". To be able to say two cells have the same formatting, we need both a notion of formatting and a notion of equality or differentiation. The formatting of a cell in a spreadsheet can be very extensive and is handled in detail below, in Section 5.2, but for our purpose here a very coarse definition suffices:

Cell Formatting: A function F of type $\mathrm{C} \rightarrow \mathrm{F}$ representing the cell format.

With this definition of the formatting of a cell, two cells are **Formatting Equal** if their formatting is equal. Defining formatting equality like this means that two cells can also be formatting equal if they are not copy equal (cf. definition of copy equivalence in Section 3.1). Formatting equality ($=_\mathrm{F}$) is formally defined as:

Formatting Equality: $c_1 : C =_\mathrm{F} c_2 : C \Leftrightarrow F(C_1) = F(C_2)$.

Every part of the description of a Types$_\mathrm{S}$ I relation now having a meaning, a formal definition can be given as:

Types$_\mathrm{S}$ I: $(\forall c_1 : C \in CC_1, \exists c_2 : C \in CC_2 : c_1 \equiv c_2 \wedge c_1 =_\mathrm{F} c_2) \wedge |CC_1| \leq |CC_2|$

As this definition is very similar to the definition of two matching clusters, as given in Section 4.1.2, the informal description of a Types$_\mathrm{S}$ I can be changed to: "Two matching clone clusters where for every cell value there is a clone with formatting equal cells".

Types$_\mathrm{S}$ II As previously discussed in Section 2.2, the defined Typess$_\mathrm{S}$ allow for more differences when the Types$_\mathrm{S}$ increases. This means that Types$_\mathrm{S}$ II will allow for more differences than Types$_\mathrm{S}$ I. From the informal description of Types$_\mathrm{S}$ II we see that there is no restriction on the formatting differences, where this restriction is present in Types$_\mathrm{S}$ I. Removing this restriction from the informal description of Types$_\mathrm{S}$ I, we get the description of Types$_\mathrm{S}$ II: "Two matching clone clusters". This description leads to a formal definition for a Types$_\mathrm{S}$ II relation as:

Types$_\mathrm{S}$ II: $(\forall c_1 : C \in CC_1, \exists c_2 : C \in CC_2 : c_1 \equiv c_2) \wedge |CC_1| \leq |CC_2|$

Types$_\mathrm{S}$ III For Types$_\mathrm{S}$ III a general description exists, encompassing the different descriptions of its subtypes. This description is such that it allows a Types$_\mathrm{S}$ III relation to have more differences than a Types$_\mathrm{S}$ II relation, just as is the case with Types$_\mathrm{S}$ II versus Types$_\mathrm{S}$ I relations. As with a Types$_\mathrm{S}$ II relation, in a Types$_\mathrm{S}$ III relation there is no restriction on formatting equality. The difference between these two Typess$_\mathrm{S}$ is that in a Types$_\mathrm{S}$ III cell values may differ. In terms of the clone clusters of our algorithm, it means that the clone clusters that make up a copy-paste relation do not have to match, i.e. they can be near-miss clusters. This leads to a description of Type$_S$ III: “Two near-miss clone clusters”, which is formally defined as:

Type$_S$ III: $\exists c_1 : C \in CC_1, c_2 : C \in CC_2 : c_1 \equiv c_2$

As stated above, there are three subtypes for a Type$_S$ III relation. These subtypes restrict the way in which two clone clusters are allowed to differ.

Type$_S$ IIIa In Type$_S$ IIIa, only one **one-cell difference** is allowed, according to the description. To explain the concept of a one-cell difference, we need the notion of a cell’s neighbours. This notion has been introduced in 3.1 as:

Neighbours: Neighbours N is a function of type C $\rightarrow$ $\{$C$\}$ representing all neighbours of a cell. The neighbours of a cell are those cells that have a location wherein either the Column or the Row differs only one unit from the location of the cell itself.

This definition allows us to formally define a one-cell difference as:

One-cell difference: $c_1 : C \in CC_1 : \forall c_n : C \in N(c_1) \exists c_2 : C \in CC_2 \equiv c_n \wedge |CC_1| \leq |CC_2|$

Informally, this means that for each cell c_1 in the smallest cluster that is not copy equal to a cell in the larger cluster, all the neighbouring cells of c_1 must have a copy equal cell in the larger cluster. The description of a Type$_S$ IIIa relation is, using the notion of a one-cell difference, “Two near-miss clone clusters where only one one-cell difference is allowed between the two clusters”. More formally, this is:

Type$_S$ IIIa: $\exists c_1 : C \in CC_1, c_2 : C \in CC_2 : c_1 \equiv c_2 \wedge |\{c : C \in CC_1\} : \forall c_n : C \in N(c) \exists c_2 : C \in CC_2 \equiv c_n| = 1 \wedge |CC_1| \leq |CC_2|$

Type$_S$ IIIb In Type$_S$ IIIb again more leniency is allowed with respect to Type$_S$ IIIa. Where the number of one-cell differences was restricted to 1 for a Type$_S$ IIIa relation, here this is abandoned, so a number of one-cell differences can occur. This leads to a formal definition of a Type$_S$ IIIb relation as:

Type$_S$ IIIb: $\exists c_1 : C \in CC_1, c_2 : C \in CC_2 : c_1 \equiv c_2 \wedge c_1 : C \in CC_1 : \forall c_n : C \in N(c_1) \exists c_2 : C \in CC_2 \equiv c_n \wedge |CC_1| \leq |CC_2|$

This means that “Two near-miss clone clusters where only one-cell differences are allowed between the two clusters”.

Type$_S$ IIIc To capture all types of near-miss clone clusters with the given definition of a Type$_S$ III relation, we need a third subtype. The first two subtypes restrict the differences that are allowed to exist between the two clusters to one-cell differences, meaning that no two neighbouring cells can ever be different. Hence, Type$_S$ IIIc is defined as “Two near-miss clone clusters”, the same as the Type$_S$ III relation. Having the same description as the general Type$_S$ III, the formal definition is also the same:

Type$_S$ IIIc: $\exists c_1 : C \in CC_1, c_2 : C \in CC_2 : c_1 \equiv c_2$

5.2 Formatting Differences

In spreadsheets, a specific formatting of a cell can give its contents a specific meaning. For example, since dates are stored internally as numbers, one number can be formatted as a date, while its copy can be formatted as a monetary value. On the other hand, there are also a number of formatting options that do not impose a specific meaning on a cell, like the thickness of a cells borders. Using those formatting options that possibly change the meaning of a cell, we are able to distinguish a total of five main formatting differences. Each of these differences is notated as FD with a three-letter subscript as an indication of the kind of difference. However, since two of these main types are further divided, these subtypes have a six-letter subscript, as can be seen in Table 5.1.

Notation	Meaning
FD_{TYP}	Type difference
FD_{MON}	Monetary difference
FD_{MONSYM}	Monetary symbol difference
FD_{MONNOT}	Monetary notation difference
FD_{DAT}	Date difference
FD_{DEC}	Decimal difference
FD_{IND}	Independent difference
FD_{INDCOL}	Independent colour difference
FD_{INDCON}	Independent conditional difference
FD_{INDGEN}	Independent general difference

Table 5.1: Description of formatting differences

5.2.1 Categorisation

To generalise the options that can change the meaning of a cell, cells have a formatting property called **Numberformat**. Using this numberformat property, it is possible to colour cells when their value is within certain boundaries, or even to add text to a number. Since this numberformat is such a versatile, but possibly very meaningful, property, we include this property in our categorisation. To do this, a precise definition of a numberformat is needed:

Numberformat: A function NF of type C $\rightarrow$ NF that returns a cells NumberFormat-property as defined by Excel[1]

Since this is the only kind of formatting that can change the meaning of a cell's value, this is the only kind of formatting we use to determine formatting equality. This also allows for a redefinition of cell formatting, to make it more precise:

Cell Formatting: A function F of type C $\rightarrow$ NF representing the cell's numberformat.

[1]`http://msdn.microsoft.com/en-us/library/office/bb213677(v=office.12).aspx`

During the analysis of our results, we noticed a number of copy-paste relations that contained formatting differences. Some of these differences however looked similar, so we decided to group them into a number of different categories. These categories in turn allowed us to sort the found copy-paste relations, to provide more meaningful results to the user. Hence, we distinguish 5 categories, shown in Table 5.1.

As a definition of the formatting differences we define the formatting equality function between two cells as $=_{\text{F}}$, when the cell formatting differs according to the definition of that formatting difference. In addition to this formatting equality, there is also a **format string equality** defined for numberformats. This equality ($=$) is defined as the usual equality over strings, using the textual representation of the numberformats:

Format String Equality: $F(c_1) = F(c_2) : \text{toString}(F(c_1)) = \text{toString}(F(c_2))$

5.2.2 Explanation per Formatting Difference

FD$_{\text{TYP}}$ The first type of formatting difference we distinguish is a difference in **Formatting Type**. Since by changing the numberformat of a cell its value can have a completely different meaning, several general formatting types are defined. These types are defined in such way that each cell's numberformat is categorized using these types. Formally, these types are defined as:

Formatting Type: A function FT of type F $\rightarrow$ FT returning the formatting type of a numberformat. The formatting type is one of the following: "Standard", "Number", "Scientific", "Currency", "Date", "Percentage", "Text" or "Custom".

Using these formatting types, some notations still look the same and thus cannot imply any meaning to the cells. To prevent these similar looking types from creating too many FD$_{\text{TYP}}$ differences, we have grouped some of them together. This grouping resulted in a situation where "Number", "Text" and "Standard" are considered equal and all other types do differ. Formally, this leads to the following definition of FD$_{\text{TYP}}$:

FD$_{\text{TYP}}$: $F(c_1) \neq_{\text{F}} F(c_2) : FT(F(c_1)) \neq FT(F(c_2)) \wedge \neg((FT(F(c_1)) = \text{Number} \vee FT(F(c_1)) = \text{Standard} \vee FT(F(c_1)) = \text{Text}) \wedge (FT(F(c_2)) = \text{Number} \vee FT(F(c_2)) = \text{Standard} \vee FT(F(c_2)) = \text{Text}))$

FD$_{\text{MON}}$ The second type concerns itself with differences between cells where both cells have a formatting type "Currency". In contrast to the different types of copy-paste relations (Types$_{\text{S}}$), this is a clear example where the types are not subsets of each other. For two cells that both have the formatting type "Currency", we have defined two subtypes. The first of the subtypes occurs when the two cells use a different symbol for their monetary value, e.g., one cell uses a US-dollar sign ($) while the other uses a Pound sign (£). When a copy-paste relation is found and the cells in the relation use different **Monetary Symbols**, it is very possible that some conversion should have taken place between the two clone clusters. Because of the possible severity of this type of difference, we made this a special type of formatting difference. For the formal definition of this type of difference, we first need a function to retrieve the monetary symbol from the numberformat, which is defined as:

Monetary Symbol: A function MS of type F $\rightarrow$ MS returning the country-specific monetary symbol used. This function is only defined when $FT(F) = \text{Currency}$.

Using this definition of the monetary symbol, we can define a $\text{FD}_{\text{MONSYM}}$ as:

$\text{FD}_{\text{MONSYM}}$: $F(c_1) \neq_{\text{F}} F(c_2) : FT(F(c_1)) = FT(F(c_2)) \wedge FT(F(c_1)) = \text{Currency} \wedge MS(F(c_1)) \neq MS(F(c_2))$

The second subtype describes those differences where both cells have the same monetary symbol, but have a difference in their formatting nonetheless. An example of this difference is when a minus sign is used in one cell to indicate a negative value, while the value in the other cell is given between brackets. Since using such different cell formatting for different cells can lead to confusion of the user, we have created a $\text{FD}_{\text{MONNOT}}$ as:

$\text{FD}_{\text{MONNOT}}$: $F(c_1) \neq_{\text{F}} F(c_2) : FT(F(c_1)) = FT(F(c_2)) \wedge FT(F(c_1)) = \text{Currency} \wedge MS(F(c_1)) = MS(F(c_2)) \wedge F(c_1) \neq F(c_2)$

FD_{DAT} This type of formatting difference is defined for two cells that have a formatting type "Date". Since a date can be formatted in a large number of ways without changing the meaning of the date, it would not serve our purpose to simply record every instance where two of these cells have a different formatting. There is however one possible difference in date formatting that could confuse a user. If one cell puts the day in front of the month, and the other formats the date using the month first, a date like January 11th can suddenly look like November 1st[2]. For the definition of a FD_{DAT} difference, we need a way to find which part of a date is first in the numberformat, the day or the month. For this, we use a function **Day First**, defined as:

Day First: A function DF of type F $\rightarrow$ DF returning whether the day is mentioned first in the string representation of a Numberformat or not.

Utilizing this definition, we define a FD_{DAT} difference as:

FD_{DAT}: $F(c_1) \neq_{\text{F}} F(c_2) : FT(F(c_1)) = FT(F(c_2)) \wedge FT(F(c_1)) = \text{Date} \wedge DF(F(c_1)) \neq DF(F(c_2))$

FD_{DEC} FD_{DEC} is about differences in the shown number of decimal places. Since the mathematical concept of decimal places is only defined for numbers, this type is only valid for a subset of formatting types. The formatting types that (possibly) concern a number, are "Standard", "Currency", "Percentage", "Scientific" and "Number". When both of the cells have a formatting type concerning a number, the number of decimal places that is shown in the numberformat is calculated. This is important, because intuitively, a 4 is not equal to a 4.5, but when the original value was 4.45 we can achieve both these values by restricting the number of decimal places. Since this might also lead to confusion for the user, we wanted to specifically indicate this type of difference. The number of decimal places indicated by a numberformat, can de be found using a function **Decimal Places**, which is defined as:

[2]Though we did not find this kind of difference in the spreadsheets we tested, exactly this happened to me personally with my ex employer, who switched the day and month of my birthday and congratulated me with my birthday 10 months too late

Decimal Places: A function *DP* of type F $\rightarrow$ DP returning the number of decimal places specified in the number format. This function is only defined when $FT(F) = \text{Standard} \lor FT(F) = \text{Currency} \lor FT(F) = \text{Percentage} \lor FT(F) = \text{Scientific} \lor FT(F) = \text{Number}$.

From this definition, the definition of a FD_{DEC} difference is:

FD_{DEC}: $F(c_1) \neq_{\text{F}} F(c_2) : FT(F(c_1)) = FT(F(c_2)) \land (FT(F(c_1)) = \text{Standard} \lor FT(F(c_1)) = \text{Currency} \lor FT(F(c_1)) = \text{Percentage} \lor FT(F(c_1)) = \text{Scientific} \lor FT(F(c_1)) = \text{Number}) \land DP(F(c_1)) \neq DP(F(c_2))$

FD_{IND} Where the previous types of formatting differences were defined for a certain subset of cells based on the formatting types, FD_{IND} differences can occur between any kind of cells. After finding one type of such an independent difference, looking at the specification for the NumberFormat-property in Excel[3] we found another possible difference. Needing one more category for all the format differences we could not fit into any of the previous types, we ended up with three subtypes. $\text{FD}_{\text{INDCOL}}$ concerns all numberformats where a colour is specified. Mostly this is used for displaying negative numbers, but since the specification allows for colours to be used in any format, we could not restrict it to the subset of formatting types used in FD_{DEC}. Instead of indicating whether a colour is specified in a numberformat or not, we have chosen to specify the colour used itself using a function **Format Colour**:

Format Colour: A function *FC* of type F $\rightarrow$ FC indicating which, if any, colour is used in a numberformat.

This definition for a format colour is used to formally define a $\text{FD}_{\text{INDCOL}}$ difference as:

$\text{FD}_{\text{INDCOL}}$: $F(c_1) \neq_{\text{F}} F(c_2) : FC(F(c_1)) \neq FC(F(c_2))$

From the specification of the NumberFormat-property we found that it is possible to specify a condition in a numberformat, for example to colour a value green if it has passed a certain threshold. If a user knows a value will turn green when a certain threshold is reached, it is important that this threshold is the same for all cells, so a difference in these conditions should be noted. To find this type of difference, that can occur in cells of any formatting type, we have defined a function **Condition** as:

Condition: A function *Con* of type F $\rightarrow$ Con indicating which, if any, condition is used in a numberformat.

We use this definition to formally defined a $\text{FD}_{\text{INDCON}}$ difference as:

$\text{FD}_{\text{INDCON}}$ Vb: $F(c_1) \neq_{\text{F}} F(c_2) : Con(F(c_1)) \neq Con(F(c_2))$

[3]`http://office.microsoft.com/en-us/excel-help/create-or-delete-a-custom-number-format-HP005199500.aspx?CTT=3`

A final type of formatting differences is created to capture all format differences that occur between two cells besides the differences found in the previously defined types. To avoid duplicate difference with the FD_{MON} and FD_{DAT} types, this difference is only defined for cells formatting type not being "Monetary" or "Date", in contrast to the two other subtypes of FD_{IND}. It is required however for the formatting types of the two cells to be equal, because a cell with formatting type "Date" will always have a formatting difference with a cell with formatting type "Percentage". To retrieve the numberformat of a cell without the parts checked for in the previous defined types of formatting differences, e.g., colours and conditions, we define a function **Format Remainder** as:

Format Remainder: A function FR of type $\text{F} \rightarrow \text{FR}$ returning a textual representation of the numberformat, such that $Con(F) = \varepsilon \wedge FC(F) = \varepsilon \wedge DP(F) = 0$. The function FR is not defined when $FT(F) = \text{Monetary} \vee FT(F) = \text{Date}$.

The definition of $\text{FD}_{\text{INDGEN}}$ is, using this definition of the format remainder:

$\text{FD}_{\text{INDGEN}}$: $F(c_1) \neq_{\text{F}} F(c_2) : FT(F(c_1)) = FT(F(c_2)) \wedge FT(F(c_1)) \neq \text{Monetary} \wedge FT(F(c_1)) \neq \text{Date} \wedge FR(F(c_1)) \neq FR(F(c_2))$

5.3 Reporting

The distinction of these different Types_{S} and types of formatting differences is made for the benefit of the reporting of the results to the user. This reporting should first concern the ordering of the different copy-paste relations, which should be done in such a way that the user can easily use the results to improve the design of the spreadsheet. To achieve this, an ordering based on the different Types_{S} and types of formatting differences has been defined.

Besides this ordering, we created some feedback to report the different formatting differences to the user. If the reported copy-paste relations have formatting differences, fixing these differences will in most cases improve the understandability of the spreadsheet. Since improving both the design and understandability of spreadsheets is one of the main goals of both this thesis and Infotron, feedback was also included in the algorithm.

5.3.1 Ordering

When the algorithm had become capable of differentiating between different copy-paste relations, these different types needed to be put to use to order the different relations. To do this, we defined an ordering of the different clone cluster types. Since per Type_{S} there can be a large number of relations, we also needed to find another way of ordering the relations within a Type_{S}. Therefore, we defined an ordering based on the combined formatting differences of the cells in a copy-paste relation. Using both orderings, we are able to present a list of found copy-paste relations to the user that the user can use to improve the design of his spreadsheets. The complete ordering takes place first on the location of the relation, i.e. whether it is spread across two worksheets or not. The next level of ordering is based on the Type_{S} of the clone clusters. For clusters that are ordered equally on this level, the third level of ordering will be based on the combined formatting differences of the cells of the clones.

Clone Cluster Types Having defined the different Types$_S$ for clone clusters, a basic ordering of the results can be performed based on these Types$_S$. As discussed before, this ordering is needed to provide the users with a handle of which clone clusters he or she should look at first. Since there may be many clones of a single Type$_S$ that we need to order, we need a measure on the clusters based on which this ordering takes place. For such a measure, we noted that the different Types$_S$ are related to the error ratio between two clone clusters. Type$_S$ I clusters for example only occur between two matching clusters, whereas for Type$_S$ IIIc clusters the two concerning clone cluster are always near-miss clusters. In terms of the edit ratio, this means that the edit ratio of Type$_S$ I clusters is always 100%, and for Type$_S$ IIIc clusters this ratio is always less than 100%. If two clusters have an edit ratio of 100%, by the definition of the edit ratio this means that there is no difference between the individual cells of the clusters. In other words, the probability that the two clusters are indeed part of a copy-paste relation is very high. If, on the other hand, the edit ratio is for example 50%, half the cells in the clusters would differ, lowering the probability that the two clusters form a copy-paste relation. If the error ratio is thus expressed as a probability of a true copy-paste relation, we call it a **Confidence Level**:

Confidence Level: An expression for the probability with which two clone clusters together form a copy-paste relation, i.e. one is truly copied from the other.

Using these confidence levels as a measure, we have been able to sort the list of copy-paste relations that were found by our algorithm before presenting them to the user. The user will be presented with a list of relations where the top-most has the highest probability of actually being a copy-paste relation.

Combined Formatting Differences The formatting differences we defined above are all based on a comparison of two cells, whereas the clone cluster categories were defined between two clone clusters. To be able to report both the formatting differences and the cluster categories in an equal style, we needed to find a way to define formatting differences on clone clusters. We achieved this by adding the formatting differences for all cells in the smallest clone cluster. This addition means that for each type of formatting difference we count the clones suffering from a formatting difference of this that type, resulting in an **FD Count**:

FD Count: $TC(cc_1 : CC, cc_2 : CC) = |\{c_1 : C \in cc_1, c_2 : C \in cc_2 : F(c_1) \neq_F F(c_2)\}|$ with $\neq_F$ defined for each type of formatting difference.

The FD Count of each type of formatting difference is then used as the **Combined Formatting Difference**:

Combined Formatting Difference: A function CD of type CC $\rightarrow$ {FD Count} where every type of formatting difference has an entry in CD.

Formatting Weight We use these combined formatting differences to order the clone clusters. To do this, we created a function **Formatting Weight** that calculates a single value for the combined formatting difference. To calculate the formatting weight of a combined

Notation	Weight
FD_{TYP}	1
FD_{MON}	n/a
$\text{FD}_{\text{MONSYM}}$	0.8
$\text{FD}_{\text{MONNOT}}$	0.3
FD_{DAT}	0.5
FD_{DEC}	0.3
FD_{IND}	n/a
$\text{FD}_{\text{INDCOL}}$	0.1
$\text{FD}_{\text{INDCON}}$	0.1
$\text{FD}_{\text{INDGEN}}$	0.1

Table 5.2: Description of formatting differences

formatting difference, every FD count that is part of the combined formatting difference is assigned an individual weight. This weight is then multiplied by the FD count, and the resulting, weighted, FD counts are added to find a weighted total. An extra advantage of this approach is that the user can influence the outcome of the sorting process for formatting differences by attaching a different weight to a specific FD count. This is useful when the user wants to focus on a specific type of formatting difference, e.g., the $\text{FD}_{\text{MONSYM}}$ for differences in monetary symbols. To be able to give a formal definition of the formatting weight function, we need a function that will return the weight set for a specific FD count, called **FD Weight**. The two functions are defined as:

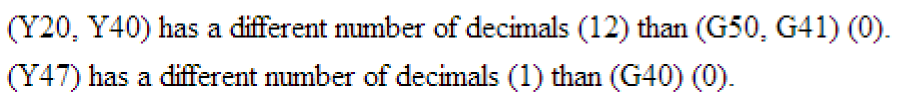
FD Weight: A function TW of type FD $\rightarrow$ TW that returns the weight assigned to a specified FD.

Formatting Weight: A function FW defined as CD $\rightarrow$ FW where $FW = \Sigma(TW(TC \in CD) * TC)$

The default FD weights have been defined by us, based on preliminary test results, as in Table 5.2.

5.3.2 Reporting a Formatting Difference

The output of a copy-paste relation with formatting differences includes a description of these differences. For these descriptions, the various formatting differences of the individual cells within a clone cluster are first sorted, so that the results can be given per cell. The way in which these cells are ordered is such that those cells belonging to the most individual types of formatting difference of differences are reported first. When the number of types of formatting difference two cells belong to are equal, the cells are ordered according to the types they belong to. For this ordering, the types of formatting difference are compared using the order defined by the FD weights, i.e. a cell belonging to FD_{DAT} is sorted before a cell belonging to $\text{FD}_{\text{MONNOT}}$, but after a cell belonging to $\text{FD}_{\text{MONSYM}}$. This way, the cells that are reported first are also the cells that possibly have the most impact on improving the quality of the spreadsheet when altered.



```
(Y20, Y40) has a different number of decimals (12) than (G50, G41) (0).
(Y47) has a different number of decimals (1) than (G40) (0).
```

Figure 5.1: Example output for formatting differences

Once this ordering is complete, a description of the different types of formatting difference is output, with the locations of the cells entered. These locations are both the locations of the originating cell, i.e. the formula cell, and the copy of this cell, i.e. the value cell. An example of such an output is given in Figure 5.1 As can also be seen in this example, the output is clustered such that multiple cells with the same formatting differences are reported in the same line, also in order to shorten the output for the user and make it more readable.

Chapter 6

Evaluation

Does our algorithm indeed find copy-paste relations in spreadsheets? And are they categorised in the correct manner? Those questions should be answered before we incorporate our algorithm in the online analysis software of Infotron. What we also needed to find out was the number of spreadsheets that contain copy-paste relations. To do this, we performed a quantitative analysis on the EUSES corpus that we report in this chapter.

6.1 Validation

To validate our algorithm we performed a quantitative analysis on the EUSES corpus [23] of 4223 spreadsheets. We ran our algorithm 52 times on this set op spreadsheets while changing some parameters in every run. This parameter tuning was performed in order to find an optimal set of parameters to use in the online version of the algorithm. The results of every run were analysed by hand, to check the results of the algorithm and to distinguish between the false and true positives. The true and false negatives were not analysed in this study, since it would be too presumptuous to assume we can find them all without having contact with the owners and creators of the spreadsheets.

6.1.1 Set-up

There were three parameters changed during several runs of the algorithm. The parameters changed were the **Minimum Cluster Size**, the **Minimal Different Values** and the **Minimum Edit Ratio**. Because these three options are described in Section 4.2.2 in detail, we will provide only a small summary here. The minimum cluster size governs the minimum number of cells that should be part of a clone cluster. The minimum number of different values defines the minimal number of different values that should be present in a clone cluster. Obviously it makes no sense to increase this number above the minimum cluster size. The third parameter we changed, the minimum edit ratio, determines whether a found copy-paste relation is reported based on the edit ratio of the relation.

For the validation, the applied values for these parameters can be found in Table 6.1. These parameters were changed in order to find the optimal value per parameter to be used in

Parameter	Values
Minimum Cluster Size	{5, 6, 7, 8, 9, 10, 11, 12}
Minimal Different Values	{3, 4, 5, 6, 7, 8, 9, 10, 11, 12}
Edit Ratio	{75%, 80%, 85%, 90%, 95%, 100%}

Table 6.1: Tested values for the given parameters for the algorithm validation

the online analysis tool. An optimal solution is one where the precision rate of the algorithm is highest, i.e. where the number of false positives is lowest.

Starting the validation with a minimum cluster size of 5 was done because we believe that clusters of less than 4 cells are too small to be meaningful. The same reasoning holds for the value chosen for the minimum number of different values. We have run the algorithm for all combinations of these two parameters, as long as the minimum number of different values was lower than the minimum cluster size. Since we were not able to contact all the owners of the spreadsheets, evaluating the near-miss clones found by our algorithm when the minimum edit ratio is set to less than 100% would lead to too much room for speculation. Therefore, as a first benchmark, we have set this minimum edit ratio at 100%.

When the optimal values for the minimum cluster size and the minimal number of different values were found, we performed tests to find the optimal value for the edit ratio. For this test we fixated the values for the minimum cluster size and the minimal number of different values, and varied the edit ratio between 75% and 100%. Because we wanted to be lenient in our online tool and not discard too many true positives, the values we chose for the two minima were slightly lower than the optimal values.

6.1.2 Analysis

For the first benchmark, with the minimum edit ratio set at 100%, we needed to determine when a found copy-paste relation was a false positive and when it was truly a copy-paste relation, or a true positive. To do this, we inspected every found relation manually on three criteria. First we checked if the two found clusters share the same data, after which we checked if one of the two clusters indeed contained only formula cells while the other cluster contained only value cells. The third criterion we checked is whether the headers of both clone clusters indicated that the data was conceptually the same. When all these three criteria were met, the found copy-paste relation was a true positive for this benchmark.

The second set of tests involved changing the minimum edit ratio to decreasingly lower values, from 100% to 75%. This decreasing also means that when evaluating a found copy-paste relation, there is progressively more room for speculation on this relation. This speculation stems from the fact that when dealing with near-miss clones, it can become increasingly difficult to determine whether the data in both clone clusters is conceptually the same data. Without the option to consult the owners and creators of the spreadsheets in the EUSES corpus, we needed to find a way in which we could still reason about these near-miss clones. To accomplish this, we chose to redefine our indication of a false positive slightly for the relations with an edit ratio of less than 100%. Where we first looked at the headers of the clone clusters to find out if they *meant* the same, we now regarded clones to be true positives if the headers were *exactly* the same. If they were, and the other two criteria were also met, the found copy-paste relation was a true positive; if one of the criteria was not met the found relation was a false positive.

Using these different runs, we calculated a number of indicators. First of all, we determined the precision of our algorithm. To do this, we compared the number of spreadsheets containing copy-paste relations to the number of spreadsheets containing true positives. This was done to avoid the results being skewed by large numbers of clones within a single spreadsheet. Next, we calculated a percentage of files containing copy-paste relations with respect to the total number of files we tested, 4223. Of these 4223, we found that only 1711 contained formula cells, which allowed us to calculate the percentage of spreadsheets that contained copy-paste relations, if the spreadsheets contained a formula.

Using the results of the Types and formatting difference classification, we were able to compute a number of percentages on these Types and formatting differences as well. With this result, we have calculated the percentage of copy-paste relations with a formatting difference. This was also related to either the total number of spreadsheets, or the number of spreadsheets which contained formula cells. Per type of formatting difference we have also calculated the frequency of the type occurs. Knowing what type of formatting difference is most common can provide information about how users can be helped to improve the design of their spreadsheets.

Minimum Size	Minimal Different Values									
	3	4	5	6	7	8	9	10	11	12
5	54.8%	59.1%	63.7%	-	-	-	-	-	-	-
6	54.2%	59.2%	62.9%	70.1%	-	-	-	-	-	-
7	53.8%	59.1%	62.5%	69.5%	70.9%	-	-	-	-	-
8	56.1%	60.2%	63.6%	70.1%	71.6%	72.9%	-	-	-	-
9	56.6%	60.6%	64.3%	71.2%	72.9%	74.6%	**81.7%**	-	-	-
10	55.1%	58.6%	62.3%	69.7%	71.4%	73.3%	80.0%	79.2%	-	-
11	56.3%	57.7%	60.9%	68.3%	70.2%	71.4%	78.4%	77.6%	78.3%	-
12	56.6%	58.1%	60.6%	67.8%	69.6%	70.9%	78%	77.1%	77.8%	81.0%

Table 6.2: Benchmark for the precision of the algorithm

6.1.3 Results

The first test's primary goal was to create a benchmark for the optimal values. The results of the algorithm are shown in Table 6.2. The values in this table show the precision of the algorithm, calculated by dividing the number of spreadsheets containing true positives by the number of spreadsheets found to contain a copy-paste relation by the algorithm. Here we chose the number of spreadsheets over the number of copy-paste relations because we wanted to avoid a skewing of the results by spreadsheets that contained many clones. Highlighted in this table is the highest precision (81.7%) which was reached when both the minimal number of different values and the minimum cluster size are set to **9**. In absolute numbers, this 81.7% means that true positives have been found in 49 spreadsheets (which can be seen in Table 6.3, where the algorithm found relations in 60 files. Regarding this

	Minimal Different Values									
Minimum Size	3	4	5	6	7	8	9	10	11	12
5	86	81	71	-	-	-	-	-	-	-
6	77	74	66	61	-	-	-	-	-	-
7	70	68	60	57	56	-	-	-	-	-
8	64	62	56	54	53	51	-	-	-	-
9	60	57	54	52	51	50	**49**	-	-	-
10	54	51	48	46	45	44	44	42	-	-
11	49	45	42	41	40	40	40	38	36	-
12	47	43	40	40	39	39	39	37	35	34

Table 6.3: Number of files containing true positives

in light of the 1711 files containing formula cells this means that 2.86% of all spreadsheets containing formula cells also contain a copy-paste relation, given these parameters.

Looking more closely at the absolute number of files containing clones, we see that for the minimal values for the parameters there are 86 files containing true positives. This amounts to approximately 5% of all 1711 files containing formula cells, which is comparable to percentages of code clones in source code (for example Cordy and Roy [51] and Mayrand et al. [38] mention this 5%).

These numbers were calculated by verifying the copy-paste relations found by our algorithm per spreadsheet. During this process of verification however, we also found false positives of clones. A large portion of the detected false positives have a certain set of data in common, for example the marks 1 to 10 in a spreadsheet concerning grades. This occurred frequently when the minimum size for clones was set to values of 6 and less. Another part of the false positives could be attributed to the use of formula's in header cells. If a formula is, for example, used to calculate a year and this year is used in other parts of the spreadsheet as an input value, the algorithm will detect this as a clone. Since we did not use any form of meta notation of the cells like header inference or manual annotation, this form of false positives can occur. Using any form of meta notation on the other hand has other disadvantages, as explained in Chapter 2. A final noteworthy form of false positives occurred because of limitations of the third party library we used to read spreadsheets, Gembox[1]. This software does not recognize array-formulas as formulas, so any cell containing such a formula is considered to be a value cell. Another disadvantage is that it detects the values used in charts as separate values, even though the charts refer to formula cells. This means that any graph using data from formula cells will be detected as a clone, since Gembox somehow reads this data twice. At the moment of writing this thesis an update for Gembox is available that fixes this particular issue, but it has not yet been implemented in Breviz.

Given the results of Table 6.2, the optimal values for both the minimal cluster size and the minimum number of different values is 9. To find the optimal value of the edit ratio we set both these values to **7**, to be more lenient and allow for more relations to be found. The results in terms of precision are shown in Table 6.4. As can be seen, the precision of

[1]`http://www.gemboxsoftware.com/`

	Edit Ratio					
	75%	80%	85%	90%	95%	100%
Precision	55.3%	57.1%	60.6%	63.0%	67.4%	**70.9%**
Number of files	114	105	99	92	86	79
True Positives	63	60	60	58	58	56

Table 6.4: Precision of the algorithm for different values of the minimum edit ratio

the algorithm goes down when the minimum edit ratio goes down. There are however files containing true positives for near-miss clusters that were not found when only matching clusters were considered. Despite this, because the total number of files increases faster than the number of files containing true positives, we consider a minimum edit ratio of **100%** to be optimal.

6.2 Evaluation of Categorisation

After we validated the algorithm and determined both the optimal values and the values used in the online tool, we evaluated our categorisation of Chapter 5. To do this, we ran our algorithm 10 times, each time using a different set of parameters. This evaluation resulted in an indication of the frequency in which the different Types$_S$ and the different types of formatting differences occur. The evaluation of the Types$_S$ and formatting differences was done over the outcome of the algorithm. Since this evaluation meant that some assumptions of the results had to be made we used all relations found by the algorithm, including those relations that were determined to be false positives in the previous analysis. Only using the true positives for this evaluation implicates that a wrong assumption in the first evaluation influences this second evaluation, which we wanted to avoid.

6.2.1 Set-up

The analysis of the determination given by the algorithm was done according to the criteria described in Chapter 5. To be able to determine a wrong classification, we analysed the values in the clone and determined whether, based on this value, the classification was correct or not. We refrained from making any assumptions on whether or not the found copy-paste relations were true or false positives during this Types$_S$ and formatting difference evaluation. This was done because any such assumption could influence the outcome of the evaluation. Especially for the types of formatting differences, since these are determined on a cell-level. If a single copy-paste relation would be wrongly classified by us, the clones in that relation could potentially influence the outcomes of this evaluation, which we wanted to avoid.

For these evaluations we varied the values for different parameters according to Table 6.5. Choosing these values we had a good cross section of all three parameters. Starting with a minimum cluster size and minimal number of different values of 7, we had the values that we chose in out online algorithm. To be able to evaluate the algorithm however, we felt

Parameter	Values
Minimum Cluster Size	{5, 7, 9, 11}
Minimal Different Values	{5, 7, 9, 11}
Edit Ratio	{100%}

Table 6.5: Tested values for the given parameters for the validation of the classification

we needed some tests of values surrounding these optimal values, so we chose 5 and 9 for both parameters. Since 9 was the determined to be the optimal value for both parameters we extended the parameters for this evaluation to also include 11. This gave us the opportunity to look for trends in this evaluation that were similar to the trend of the validation of the algorithm as seen in Table 6.2. Using these 10 combinations we calculated the number of times a given copy-paste relation is of Types I or of Types II. Types III relations could not be found using this set-up because for that type of relations we would need edit ratios of less than 100%.

Minimum Size	Minimal Different Values			
	5	7	9	11
5	74.2% / 25.8%	-	-	-
7	72.1% / 27.9%	71.4% / 28.6%	-	-
9	69.1% / 30.9%	70.9% / 29.1%	67.6% / 32.4%	-
11	67.6% / 32.4%	69.6% / 30.4%	67.2% / **32.8%**	73.4% / 27.6%

Table 6.6: Percentages of Types I and Types II occurrences in files containing copy-paste relations

6.2.2 Results

For the Types I and Types II classification, the results are given in Table 6.6. In this table, we see a trend roughly similar to the trend we see in Table 6.2. The highest chance a Types II copy-paste relation occurs in a spreadsheet that contains a copy-paste relation is 32.8%, achieved with the minimal number of different values set to "9" and the minimum cluster size set at "11". In Table 6.2 this maximum was achieved with "9" for both these parameters but despite the discrepancy between the value for the minimum cluster size the trend is the same. According to our settings, i.e. with an minimum edit ratio of 100%, this means that given a random spreadsheet that contains a copy-paste relation, there is more than 30% chance that the spreadsheet contains at least one copy-paste relation with a formatting difference. This occurs when using "9" as the value for the minimal number of different values and for the minimum cluster size, even though this percentage will increase when this last parameter is set to "11". Setting both parameters to "9", there was a 2.86% chance that a random spreadsheet containing a formula also contains a copy-paste relation (cf. Table 6.3). This means that in in one out of 110 spreadsheets containing a formula there is a copy-paste relation of Types II, i.e. two matching clusters with at least one formatting difference.

	5	7	9	11	7	9	11	9	11	11
	5	5	5	5	7	7	7	9	9	11
Number of Relations	224	205	197	163	143	136	116	81	76	47
FD_{TYP}	29%	29.3%	28.9%	33.7%	13.3%	11.8%	12.1%	17.3%	15.8%	23.4%
FD_{MONSYM}	0.4%	0.5%	0.5%	0.6%	0.7%	0.7%	0.9%	1.2%	1.3%	0%
FD_{MONNOT}	7.6%	4.4%	3.6%	4.3%	6.3%	5.1%	6.0%	8.6%	9.2%	10.6%
FD_{DAT}	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
FD_{DEC}	51.8%	52.2%	52.3%	60.7%	71.3%	72.1%	81.9%	74.1%	75.0%	63.8%
FD_{INDCOL}	21.4%	21.5%	21.8%	8.0%	20.3%	20.6%	10.3%	16.0%	14.5%	21.3%
FD_{INDCON}	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
FD_{INDGEN}	5.8%	5.9%	6.1%	6.1%	6.3%	7.4%	6.9%	11.1%	10.5%	17.0%

Table 6.7: Occurrence frequency per type of formatting difference

When choosing the values as "11" and "9", there was a 2.34% chance of containing a copy-paste relation. This means that one in every 130 spreadsheets containing a formula also contains a copy-paste relation with a formatting difference.

After calculating the occurrences of these $Types_S$ I and $Types_S$ II relations, we evaluated the results of the differentiation based on the formatting differences. We did this by looking at every $Types_S$ II relation we found, and we determined, based on the values of the cells, whether the formatting differences as they were reported actually existed. This evaluation resulted in a classification per type of formatting difference, and for every type of formatting difference we have computed the frequency of occurence. The results of these calculations can be seen in Table 6.7. Here, in the top two rows the values for the parameters **Minimum Cluster Size** resp. **Minimal Different Values** are shown. In the columns we show the frequency of a specific type of formatting difference. This frequency is calculated using the number of copy-paste relations that contain at least one clone with the given type, divided by the total amount of found copy-paste relations.

In this table, there are a number of interesting things to notice. First of all, FD_{DAT} and FD_{INDCON} are always at 0%, meaning they have never occurred in the EUSES corpus. FD_{MONSYM} also has a continuous low value. These low values always translate to 1 or no copy-paste relation that was found in all configurations. In this particular instance, the notational symbol for the formula cell was a dollar sign ($), while the value cell had a quoted dollar sign ("$") as its notational symbol. Even though it is arguable whether this is an actual formatting difference or not, according to the definition of FD_{MONSYM} it is, so we counted it as one. Another interesting thing to note from this table is that the column-totals exceed 100%. The reason for this is that some copy-paste relations have more than 1 category of formatting differences.

The highest scores in this table are, for any combination of the two parameters, for FD_{DEC}. Upon close inspection of the relations containing these formatting differences, we found that the majority of these differences can be attributed to a difference in formatting type. First of all, the majority of these differences occur when one of the two cells in the clone is formatted with a different formatting type than the other, e.g., one has a formatting type "Standard" and the other a formatting type "Number". When a cell has a formatting type "Standard", the NumberFormat-property does not specify a number of decimals that is to be shown in the cell. The number of decimals shown is however dependent on a combination of the width and value of the cell. To be able to get some standardized measure for the number of decimals of these type of cells, we used the number of decimals the value (or the evaluated formula) contains. This is further justified because the user has the capability to show more decimals in the cell by only changing the width of the cell and keeping its NumberFormat-property unchanged. A special case here must be made for the case when one cell has a formatting type "Percentage". When for example Excel displays a value of 63% with a formatting type "Number", it shows 0.63. This difference means a formatting difference of type FD_{DEC} will occur. A part of these differences amount for the relatively high values found for FD_{TYP}. This is especially true for those clones where only 5 different values are needed, since for larger clones the difference between the values for FD_{DEC} and FD_{TYP} increase significantly. This has to do with the fact that, as explained in Section 5.2.2, a difference between formatting types "Number" and "Standard" does not constitute a formatting difference of type FD_{TYP}.

	5	7	9	11	7	9	11	9	11	11
	5	5	5	5	7	7	7	9	9	11
Relations	1378	1013	834	640	654	624	490	350	329	224
Types_S II Relations	224	205	197	163	143	136	116	81	76	47
Differences	4495	4414	4362	4242	1925	1882	1777	1416	1390	1027
Differences Per Types_S II	20.1	21.5	22.1	26.0	13.5	13.8	15.3	17.5	18.3	21.9

Table 6.8: Absolute numbers on formatting difference categorization

In Table 6.8 some absolute numbers are presented, regarding the copy-paste relations and the formatting differences. As with Table 6.7, the first two rows show the values for the minimum cluster size and the minimal number of different values. Next, the total number of found copy-paste relations is shown, and how many of those relations actually contained one or more formatting differences. The next row shows the total number of formatting differences found in clones. An average amount of formatting differences per relation that contains one such difference is given in the final row. Noteworthy is that all absolute numbers decrease when the number of different values or the minimum cluster size increases. However, when looking at equal values for the parameters that denotes the minimal number of different values in a clone, the average number of differences per copy-paste relation increases. This indicates that larger copy-paste relations are more prone to formatting differences than small ones.

We have also analysed the number of the Types_S II copy-paste relations, i.e. the ones that contain copy-paste relations, with respect to the total number of copy-paste relations. In Table 6.6 we have shown this number as it pertained to the files, calculating the frequency of occurrence in files that contained at least one copy-paste relation. In Figure 6.1 we have depicted this in relation to the total number of copy-paste relations. A large difference between the two is that in Table 6.6 there is a noticeable increase in the frequency in which

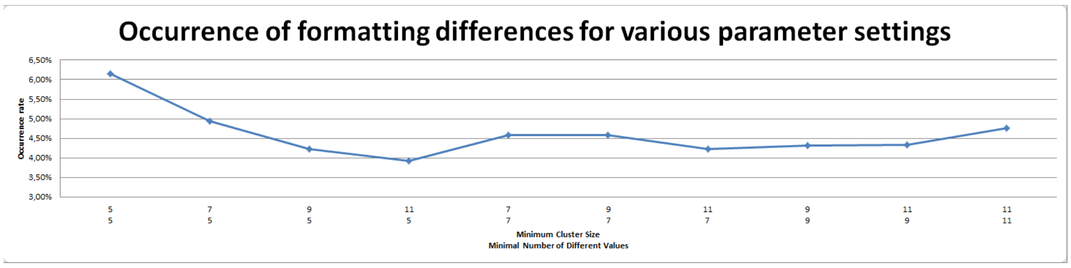


Figure 6.1: Occurrence of formatting differences

Types$_S$ II relations occur when the parameters get larger values. This increase illustrates that, given a file containing a copy-paste relation, the chance increases that this copy-paste relation contains a formatting difference when the values of the parameters are enlarged. This conclusion is further supported by the data from Table 6.8. In Figure 6.1, however, we see a line that fluctuates between 4% and 5%. This means that given any found copy-paste relation, regardless of its size, there is approximately 5% chance that the copy-paste relation contains a formatting difference.

6.3 Summary of Results

Depending on the chosen settings we found between 86 and 34 spreadsheets containing copy-paste relations out of the 1711 tested files, meaning about 5% of the files contain copy-paste relations. We have achieved a precision of the results of 81.7%, with an edit ratio set at 100%. Out of all copy-paste relations detected by our algorithm approximately one-third are of Types$_S$ II, meaning they contain at least one formatting difference. The three most common types of formatting differences are FD$_{DEC}$, FD$_{TYP}$ and FD$_{INDCOL}$. Other types, like FD$_{DAT}$ and FD$_{INDCON}$ have not been found during our evaluation. The most prominent type of formatting difference, FD$_{DEC}$, was found in more than half of the Types$_S$ II copy-paste relations, regardless of the chosen settings for the parameters.

Using the settings of the online algorithm, we found 654 copy-paste relations, of which 143 were of Types$_S$ II. These 143 relations contained a total of 1925 formatting differences, in 56 files in total. Of these 1925 differences, 71.3% were differences of type FD$_{DEC}$.

6.4 Discussion

After validating and evaluating our algorithm and the proposed classification of copy-paste relations, two main issues need to be addressed. The first of these issues is the potential threats to the validity of our validation and evaluation and how these threats were countered. The second issue concerns some alterations that we think would improve the results of the algorithm.

6.4.1 Implications

As can be seen, we can reach a precision of 81.6% using “9” for the minimal number of values and the minimum clone size and a precision of 70.9% using “7” for both these parameters. In Roy and Cordy’s survey of code clone detection techniques performed in [51], the precision is only measured as a precise percentage (100%) when looking at line-based techniques. For the token-based algorithms, which is what we based our algorithm on, the reported precision is “Low, due to normalisation and/or transformation returns many false positives”. Burd and Bailey compared three code clone detection tools and two plagiarism detection tools in [13]. Of the three code clone detection algorithms, CCFinder [32] is the only token-based tool, achieving a precision of 72%. This fits within the range we achieved, so we can conclude that, concerning the precision of the algorithm, our algorithm is comparable to well-known algorithms used for code clone detection. The two plagiarism detection tools (JPlag and Moss) achieve a precision of respectively 82% and 73%, also roughly comparable to our 81.6%.

Having achieved a precision that is comparable to de-facto standards in code clone detection and an occurrence rate of 5%, which was comparable to the occurrence rate of code clones, we conclude that our approach is applicable. Therefore we have implemented the algorithm in the online tool of Infotron[2]. Users of our tool benefit from the added detection of copy-paste relations. With a precision of 70% and higher, the reported copy-paste relations are likely to be true positives, meaning the spreadsheet the user tested can be improved. An occurrence rate of 5% and less means, on the other hand, that most users take care when creating spreadsheets, resulting in no detected relations.

For the research community we have shown that copy-paste relations occur in spreadsheets, with results similar to those achieved in other research areas. These similar results are an indication that the general problem of copy detection, be it of pieces of source code or of clone clusters, extends beyond traditional software engineering. The problem can, however, be tackled using known techniques for code clone detection in a different context. This indicates that improvements made in one of the two areas can possibly also impact the algorithms (and results) of the other.

6.4.2 Threats to Validity

Looking at the internal validity of the evaluation, discussion can arise over the assumptions made concerning the true and false positives. Any assumption made, was made without contacting the owners and creators of the spreadsheet, so every assumption could be erroneous. However, all assumptions that were made were based on the actual data in the spreadsheet. We have clearly indicated these assumptions and since we performed our validation on a publicly available set of spreadsheet, our results can be replicated. Furthermore, we have tried to minimise the number of assumptions we made over this data.

The external validity of the evaluation regards this publicly available set of spreadsheets, the EUSES corpus, and whether this corpus is sufficiently representative for spreadsheet research. This is countered by a combination of factors. Firstly, the sheer size of the corpus

[2] `http://app.infotron.nl`

needs to be taken into account. The corpus exists of over 4200 files, all collected from practice, which is the second factor. The third and final factor countering this threat is that the EUSES corpus has been used in numerous other papers on spreadsheet research.

Regarding the construct validity, there is some improvement possible in the way the formatting differences in two clone clusters are found. In the current algorithm, if the largest clone cluster has a number of cells with the same value, e.g., 5, all cells in the smallest clone cluster with the value 5 are paired with the first cell with the value 5 in the largest cluster in a clone. This also means that the formatting differences between the cells are all based on this first cell in the largest cluster. It could however be such that there is a formatting difference with this first cell 5, while this difference does not exist if the cells from the smallest cluster were paired with another cell from the largest cluster. A simplified example of this is shown in Figure 6.2.

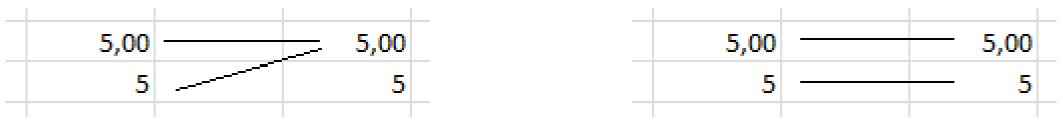


Figure 6.2: Simplified example for improved determination of formatting differences

The left side of the picture shows how the clones are currently formed. Both the values 5.00 and 5 of the left cluster are paired with the value 5.00 in the right cluster. When determining the formatting differences, this leads to a FD_{DEC} difference between 5 and 5.00. On the right side of the image we show the clones as they would be with this improvement. It can be seen that no formatting differences exist here. This improvement will only impact the calculation of the formatting differences, the precision and occurrence-rate of the algorithm will remain unchanged.

6.4.3 Improvements of the Algorithm

Having evaluated our algorithm and classification schemas, we have some suggestions for the improvement of the algorithm. The first of these improvements concerns the assumptions we made to identify the true and false positives. These assumptions were made based on the information derived manually from the headers of the data. In previous work by Hermans [27] and Abraham and Erwig [1] efforts have been made to automatically extract this data. These efforts could be incorporated in the algorithm as an extra measure of confidence in a detected clone, and by extension in a detected copy-paste relation. It can also be used in the calculation of the edit ratio or as another way of sorting the output of the algorithm. In addition to using this header information as a measure of confidence for the clones, it could also be used to improve the clones themselves. Two cells with different headers might form a clone in the current algorithm, while in the same clusters there is a “better” option for a clone when the information is taken into account. With a “better” option we mean a combination of cells where the headers match more.

A second improvement concerns the values for the parameters. In every run of our algorithm we have used a fixed set of parameters; one that paid no regard to the content of the spreadsheet. This means that small spreadsheets were evaluated using the same parameters as large ones. If, however, the existing clusters in a spreadsheet are all smaller than the minimum cluster size, copy-paste relations will never be found in that spreadsheet. We propose therefore that it might be beneficial to calculate the values of the parameters based on some properties of the spreadsheet, e.g., its number of cells.

Our third and final proposed improvement has to do specifically with the calculation of the number of decimals used for the FD_{DEC} formatting difference. As stated before, the formatting type “Standard” does not specify a number of decimals; this is governed by the value and width of the cell itself. Instead of using the total number of decimals present in the cell value, the number of cells that is shown could be used. In order to achieve this, some calculations would have to be made on the width of the cell and the width of the total value. Since these extra decimals are hidden from view to the user, it would be an improvement if they are not taken into account.

Chapter 7

Conclusions and Future Work

Concluding this thesis, we reflect on its contributions and we give some general conclusions. These contributions and conclusions are based on our initial research problem “How can the copies of formula-data in spreadsheets be detected?”. Most prominent among these conclusions is that we found that about 5% of spreadsheets contain a copy-paste relation and that our algorithm detects these relation with a precision of 81.7%. Both these numbers are comparable to those found in code clone detection in software engineering.

7.1 Contributions

This thesis has made a number of contributions:

- A definition of a notion of clones and copy-paste relations within spreadsheets.
- An approach to automatically detect these clone and relations in spreadsheets.
- An implementation of this approach in Infotron’s existing tool Breviz.
- A means of visualising and reporting the found relations.
- A categorisation of these relations closely related to well-known categorisations of code clones.
- A further classification specific to spreadsheets.
- A validation of the algorithm based on the EUSES corpus.
- An evaluation of the proposed categorisation and classification based on the EUSES corpus.

A number of these contributions have been published in paper [30], which will be presented at the International Conference on Software Engineering 2013, to be held in San Francisco.

7.2 Conclusions

For the conclusions of this paper, we revisit our research questions of Chapter 1. To recapitulate, please find the research questions below:

R.Q. 1 How can the copies of formula-data in spreadsheets be detected?

R.Q. 2 How can the results be reported to the user?

- a) How can the results be reported in a way that is usable for the users?
- b) How can the results be ordered in a useful way?

R.Q. 3 How well does the proposed solution work?

Concerning R.Q. 1, an algorithm has been developed for this detection, which is explained in Section 4.2. The algorithm works by first detecting all cells in the spreadsheet that share a value with other cells. These cells are then grouped into clone cluster, and the different clone clusters are matched. Matching clone clusters are reported as copy-paste relations.

This reporting directly touches on R.Q. 2. An extensive answer of R.Q. 2a is given in Section 4.3. The visualisation presented here is based on a textual report and extra lines shown in the data flow diagram. Also presented in this Section is an indication of a new type of reporting that has yet to be implemented. The answer of R.Q. 2b is given in Chapter 5. In short we developed a categorisation based on existing types of code clones and we developed a new categorisation based on the formatting differences that can exist between two cells in a clone. Note that even though R.Q. 2a and R.Q. 2b contain the words “usable” resp. “useful”, we did not evaluate the usefulness of the approach. This is detailed further in Section 7.3.

For the complete answer of R.Q. 3 we refer the reader to Chapter 6. In general, we can say that we found copy-paste relations in about 5% of the tested spreadsheets. We have also noted that the possibility that a given copy-paste relations contains a formatting difference increases as the relation itself becomes larger. Given a file containing copy-paste relations, we found that about 30% of these relations contain formatting differences. Additionally, we found an optimal setting for the parameters we introduced in our algorithm with which we can achieve a precision of 81.7%, meaning only 18.3% of the reported results is a false positive.

The occurrence rate of 5% and the precision of 81.7% are both comparable to results of well-known algorithms for code clone detection, like CCFinder [32]. This shows that the general problem of copy detection can be expanded beyond traditional software engineering and plagiarism detection. Spreadsheet users and creators benefit from this research as a result of these numbers; the high precision indicates that most of the results presented to the user are indeed copy-paste relations. We have not evaluated the absolute usefulness of the results, nor the possible improvement in the spreadsheet structure when the relations are made explicit.

7.3 Future work

Finally, we have some suggestions for future research opportunities. Three of these suggestions are based on the evaluation and feedback of the algorithm, while two provide options for possible extensions of the algorithm.

7.3.1 Evaluation and Feedback

As stated, we would like to point out three possibilities for future research based on evaluation and feedback. The first of these three has to do with the validation of the algorithm as it was performed in this thesis. As we said in Chapter 6, for this validation we have only looked at the true and false positives. The possibilities of study that remain for future research are those of the true and false negatives. For this type of study, the researcher would have to be in contact with the owners and creators of the spreadsheets, so this has to be done with a different data set than the EUSES corpus. To validate the algorithm with a representative set of spreadsheets, gathering the spreadsheets needed will take some time, as will the interviews with the owners and creators of the spreadsheets.

This leads to our second recommendation, i.e. the evaluation of the feedback that is given. If there is contact with a number of different users and their spreadsheets have been analysed by our algorithm, an evaluation of the feedback the algorithm gives can be done. This would answer open questions regarding the usefulness of the feedback and it could help to validate the ordering we have proposed for the output. It could also serve to improve the formatting weights we assigned based on some preliminary tests.

A set of users and their spreadsheets, as was needed in the previous two recommendations, is also needed for our third recommendation; the evaluation of the usefulness of the algorithm. Does the detection of the copy-paste relations in spreadsheets contribute to better design of a spreadsheet? Does it help to reduce errors and does it help to improve the understandability of spreadsheets? These are a number of questions that can only be answered by spreadsheet users, owners and creators, but they are important in the understanding of spreadsheets and spreadsheet design. In such a study it could also be tested if "better" spreadsheets are being created if this detection of copy-paste relations would be incorporated during the design of a spreadsheet, instead of it being used afterwards, as is now the case.

7.3.2 Extensions

The first of the two possible extensions of the algorithm relates to the original research question for this thesis, which was the following: "How can the copies of data in spreadsheets be detected?". This original question covers a larger spectrum of copy-paste relations than is covered in this thesis. In this thesis we only sought for copies for formula cells, while we could also have looked for copies of any kind of data. A problem that will have to be solved in this type of situation is the problem of the origin of the data, i.e. knowing which side of the clone is the copy. This problem has an equivalent in code clone detection, where there is also a problem of origin analysis [25, 53].

The second possible extension also has an equivalent in code clone detection. It relates to tracking copy-paste relations in an evolving spreadsheet, which is similar to clone genealogies in [10, 35]. One major difference here, is that for source code it is common practice to use some type of versioning system like SVN or Git, but there is no such commonly accepted equivalent for spreadsheets. Some companies store their spreadsheets in a SharePoint workspace, but this is not common practice for all spreadsheets. Another practice used is to store a copy of every version of the file somewhere on the company server, thus creating a crude versioning system for the company. These kinds of versioning for spreadsheets could possibly be used to track the evolution of copy-paste relations in spreadsheets, to gain more insight in how they arise and whether they persist for an extended period of time or not.

Bibliography

[1] R. Abraham and M. Erwig. Header and unit inference for spreadsheets through spatial analyses. In *Visual Languages and Human Centric Computing, 2004 IEEE Symposium on*, pages 165–172. IEEE, 2004.

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

[3] Y. Ahmad, T. Antoniu, S. Goldwater, and S. Krishnamurthi. A type system for statically detecting spreadsheet errors. In *Automated Software Engineering, 2003. Proceedings. 18th IEEE International Conference on*, pages 174–183. IEEE, 2003.

[4] T. Antoniu, P.A. Steckler, S. Krishnamurthi, E. Neuwirth, and M. Felleisen. Validating the unit correctness of spreadsheet programs. In *Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on*, pages 439–448. IEEE, 2004.

[5] Y. Ayalew. *Spreadsheet Testing Using Interval Analysis*. PhD thesis, PhD thesis, Institut für Informatik-Systeme, Universität Klagenfurt, 2001. At https://143.205.180.128/Publications/pubfiles/psfiles/2001-0125-YA.ps on 22 August, 2001.

[6] Y. Ayalew, M. Clermont, and T. Mittermeir. Detecting errors in spreadsheets. In *Proceedings of EuSpRIG 2000 Symposium: Spreadsheet Risks, Audit and Development Methods*, 2000.

[7] Y. Ayalew and R. Mittermeir. Interval-based testing for spreadsheets. In *Proceedings of International Arab Conference on Information Technology*, pages 414–422, 2002.

[8] Y. Ayalew and R. Mittermeir. Spreadsheet debugging. In *Symposium Proceedings EuSpRIG 2008, University of Greenwich, London, UK*. European Spreadsheet Risks Interest Group, 2008.

[9] B.S. Baker. A program for identifying duplicated code. In *Proc. Computing Science and Statistics: 24th Symposium on the Interface*.

[10] T. Bakota. Tracking the evolution of code clones. *SOFSEM 2011: Theory and Practice of Computer Science*, pages 86–98, 2011.

[11] M. Balazinska, E. Merlo, M. Dagenais, B. Lague, and K. Kontogiannis. Measuring clone based reengineering opportunities. In *Software Metrics Symposium, 1999. Proceedings. Sixth International*, pages 292–303. IEEE, 1999.

[12] S. Bellon. Vergleich von techniken zur erkennung duplizierten quellcodes. *Master's Thesis, Institut fur Softwaretechnologie, Universitat Stuttgart, Stuttgart, Germany*, 2002.

[13] E. Burd and J. Bailey. Evaluating clone detection tools for use during preventative maintenance. In *Source Code Analysis and Manipulation, 2002. Proceedings. Second IEEE International Workshop on*, pages 36–43. IEEE, 2002.

[14] M. Burnett, J. Atwood, R.W. Djang, J. Reichwein, H.J. Gottfried, and S. Yang. Forms/3: A first-order visual language to explore the boundaries of the spreadsheet paradigm. *Journal of functional programming*, 11(2):155–206, 2001.

[15] C. Chambers and M. Erwig. Dimension inference in spreadsheets. In *Visual Languages and Human-Centric Computing, 2008. VL/HCC 2008. IEEE Symposium on*, pages 123–130. IEEE, 2008.

[16] C. Chambers, M. Erwig, M. Luckey, et al. Sheetdiff: A tool for identifying changes in spreadsheets. In *IEEE Int. Symp. on Visual Languages and Human-Centric Computing*. Citeseer, 2010.

[17] M. Clermont and R. Mittermeir. Auditing large spreadsheet programs. In *Proceedings of the International Conference on Information Systems Implementation and Modeling*, pages 87–97. Citeseer, 2003.

[18] N. Davey, P. Barson, S. Field, R. Frank, and D. Tansley. The development of a software clone detector. *International Journal of Applied Software Technology*, 1995.

[19] J.S. Davis. Tools for spreadsheet auditing. *International Journal of Human-Computer Studies*, 45(4):429–442, 1996.

[20] S. Ducasse, M. Rieger, and S. Demeyer. A language independent approach for detecting duplicated code. In *Software Maintenance, 1999.(ICSM'99) Proceedings. IEEE International Conference on*, pages 109–118. IEEE, 1999.

[21] M. Erwig and M. Burnett. Adding apples and oranges. *Practical Aspects of Declarative Languages*, pages 173–191, 2002.

[22] M. Fisher, M. Cao, G. Rothermel, C.R. Cook, and M.M. Burnett. Automated test case generation for spreadsheets. In *Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on*, pages 141–151. IEEE, 2002.

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

[24] D.F. Galletta, D. Abraham, M. El Louadi, W. Lekse, Y.A. Pollalis, and J.L. Sampler. An empirical study of spreadsheet error-finding performance. *Accounting, Management and Information Technologies*, 3(2):79–95, 1993.

[25] M. Godfrey and Q. Tu. Tracking structural evolution using origin analysis. In *Proceedings of the international workshop on Principles of software evolution*, pages 117–119. ACM, 2002.

[26] F. Hermans, M. Pinzger, and A. van Deursen. Code smells in spreadsheet formulas. In *Proceedings of the International Conference on Software Maintenance (ICSM)*, page To appear.

[27] F. Hermans, M. Pinzger, and A. van Deursen. Automatically extracting class diagrams from spreadsheets. *ECOOP 2010–Object-Oriented Programming*, pages 52–75, 2010.

[28] F. Hermans, M. Pinzger, and A. van Deursen. Supporting professional spreadsheet users by generating leveled dataflow diagrams. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 451–460. ACM, 2011.

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

[30] F. Hermans, B. Sedee, M. Pinzger, and A. van Deursen. Data clone detection and visualization in spreadsheets. In *Proceedings of ICSE '13*, 2013. to appear.

[31] J.H. Johnson. Identifying redundancy in source code using fingerprints. In *Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research: software engineering-Volume 1*, pages 171–183. IBM Press, 1993.

[32] T. Kamiya, S. Kusumoto, and K. Inoue. Ccfinder: a multilinguistic token-based code clone detection system for large scale source code. *Software Engineering, IEEE Transactions on*, 28(7):654–670, 2002.

[33] C. Kapser and M.W. Godfrey. Toward a taxonomy of clones in source code: A case study. In *Proceedings of the Conference on Evolution of Large Scale Industrial Software Architectures (ELISA03)*, pages 67–78, 2003.

[34] C. Kapser and M.W. Godfrey. Aiding comprehension of cloning through categorization. In *Software Evolution, 2004. Proceedings. 7th International Workshop on Principles of*, pages 85–94. IEEE, 2004.

[35] M. Kim and D. Notkin. Using a clone genealogy extractor for understanding and supporting evolution of code clones. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 1–5. ACM, 2005.

[36] K. Kontogiannis, M. Galler, and R. DeMori. Detecting code similarity using patterns. In *Working Notes of the Third Workshop on AI and Software Engineering: Breaking the Toy Mold (AISE)*, pages 68–73, 1995.

[37] V.I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet Physics-Doklady*, volume 10, 1966.

[38] J. Mayrand, C. Leblanc, and E.M. Merlo. Experiment on the automatic detection of function clones in a software system using metrics. In *Software Maintenance 1996, Proceedings., International Conference on*, pages 244–253. IEEE, 1996.

[39] E.M. McCreight. A space-economical suffix tree construction algorithm. *Journal of the ACM (JACM)*, 23(2):262–272, 1976.

[40] R. Mittermeir and M. Clermont. Finding high-level structures in spreadsheet programs. In *Reverse Engineering, 2002. Proceedings. Ninth Working Conference on*, pages 221–232. IEEE, 2002.

[41] R. Mittermeir, M. Clermont, and Y. Ayalew. User centered approaches for improving spreadsheet quality. *Klagenfurt: Klagenfurt University*, 12, 2000.

[42] D. Nixon and M. O'Hara. Spreadsheet auditing software. *arXiv preprint arXiv:1001.4293*, 2010.

[43] R.R. Panko. Applying code inspection to spreadsheet testing. *Journal of Management Information Systems*, pages 159–176, 1999.

[44] R.R. Panko. Revisiting the panko-halverson taxonomy of spreadsheet errors. In *Symposium Proceedings EuSpRIG 2008, University of Greenwich, London, UK*. European Spreadsheet Risks Interest Group, 2008.

[45] R.R. Panko and R.P. Halverson Jr. Spreadsheets on trial: A survey of research on spreadsheet risks. In *System Sciences, 1996., Proceedings of the Twenty-Ninth Hawaii International Conference on,*, volume 2, pages 326–335. IEEE, 1996.

[46] K. Rajalingham. A revised classification of spreadsheet errors. In *Symposium Proceedings EuSpRIG 2005, University of Greenwich, London, UK*, pages 185–199. European Spreadsheet Risks Interest Group, 2005.

[47] K. Rajalingham, D.R. Chadwick, and B. Knight. Classification of spreadsheet errors. In *Symposium Proceedings EuSpRIG 2000, University of Greenwich, London, UK*, pages 23–34. European Spreadsheet Risks Interest Group, 2000.

[48] J. Reichwein, G. Rothermel, and M. Burnett. Slicing spreadsheets: An integrated methodology for spreadsheet testing and debugging. In *ACM SIGPLAN Notices*, volume 35, pages 25–38. ACM, 1999.

[49] G. Rothermel, M. Burnett, L. Li, C. Dupuis, and A. Sheretov. A methodology for testing spreadsheets. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 10(1):110–147, 2001.

[50] G. Rothermel, L. Li, C. DuPuis, and M. Burnett. What you see is what you test: A methodology for testing form-based visual programs. In *Software Engineering, 1998. Proceedings of the 1998 International Conference on*, pages 198–207. IEEE, 1998.

[51] C.K. Roy and J.R. Cordy. A survey on software clone detection research. *Queens School of Computing TR*, 541:115, 2007.

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

[53] L. Zou and M.W. Godfrey. Detecting merging and splitting using origin analysis. In *Proceedings of the 10th Working Conference on Reverse Engineering*, page 146. IEEE Computer Society, 2003.