Spreadsheet Visualisation to Improve End-user Understanding

Daniel Ballinger **Robert Biddle** **James Noble**

School Mathematical and Computer Sciences
Victoria University of Wellington,
PO Box 600, Wellington 6001, New Zealand
Email: {db, robert, kjx}@mcs.vuw.ac.nz

Abstract

Spreadsheets are an extremely common form of end-user programming that have many applications, from calculating student marks to accounting for global multinationals. Ways of studying the structure of a spreadsheet itself are normally constrained to the tools provided in the spreadsheet software. This paper explores new ways to visualise spreadsheets in a manner that is independent of the program they were created in, explains the technology involved, and presents examples of the visualisations that can be produced. The techniques involved in reading the spreadsheets also facilitate larger scale analysis of spreadsheets for performing corpus analysis.

Keywords: spreadsheets, visualisation, end-user programming

1 Introduction

Since spreadsheets made an entrance into the world of computing they have found extensive use in a diverse range of disciplines, as well as throughout the general population. Professionals in commerce, mathematics, engineering, science, medicine, the arts, social science, and education find the spreadsheet to be a natural tool for modelling, implementing and analysing algorithms, constructing laboratory reports, carrying out statistical analysis, and producing graphical displays.

The high uptake of spreadsheet usage can be attributed to their ability to allow end-users to model problems in a layout that is natural to them. The simplicity and popularity of spreadsheets makes them one of the most common forms of end-user programming currently in use. When first interacting with a spreadsheet, however, the user has to process a daunting amount of information in terms of layout and hidden inter-cell dependencies created by formulas. This problem with the organisation of code has been observed by users for some time and has been commented on by Bonnie Nardi:

> It is difficult to get a global sense of the structure of an individual formula that may have dependencies spread out all over the spreadsheet table. Users have to track down individual cell dependencies one by one, tacking back and fourth all over the spreadsheet.(Nardi 1993)

In this paper we explore visualisations that aid the end-user in understanding the underlying data flow structures that are present in spreadsheets. We investigate the creation of a set of images that could represent the contents at a more abstract level than is possible with the spreadsheet software. The user can start with very general information and then progress towards the actual details present in the spreadsheet. As the user progresses through these visualisations they are learning about the layout and dependency structures without being exposed to actual values and other lower level properties in the spreadsheet.

Spreadsheet software itself provides a limited ability to achieve such abstract views of the information they contain. Our design goal was to have the ability to quickly implement new visualisations in a manner independent from the application they were created in. To create these abstract diagrams, access to the internals of the spreadsheet, at the same level accessible by the user, is required. To avoid the limitations present in the spreadsheet software the information is extracted to a more versatile programming environment, which is Java in the case of this project. The greater flexibility achieved outside the application is a trade-off with the benefits that could be achieved by having visualisations directly integrated with the information in the spreadsheet.

Taking into account the strong market dominance and the potential resources, both in the size of the user base and relevance with other research, Microsoft Excel was chosen as the specific spreadsheet application to focus on. Excel is the most utilised example of a spreadsheet application. Based on data sourced from the analysts at Gartner Group, Excel's position as the market leader in terms of revenue share often tops more than 90% (Liebowitz 1999, Blackwood 2002, Krazit 2002), resulting in a vast potential user base.

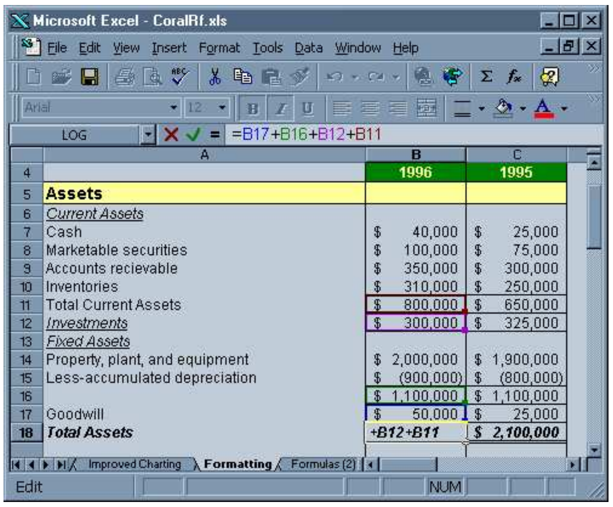
The rest of this paper is organised as follows. Section 2 presents background information on Excel's current support for visualisation of these structures and the related research. Section 3 presents some of the visualisations we created and section 4 contains the conclusions drawn from the use of these visualisations.

2 Background

2.1 Current support for visualisation

Modern versions of Microsoft Excel provide two techniques to help visualise the invisible dataflow model of a spreadsheet.

The first is called the "Range Finder", which is invoked by selecting a cell containing a formula and clicking in the formula bar. An example usage is shown in figure 1. This results in Excel colouring all the addresses in the formula and the respective regions in the spreadsheet with a rectangle of the same colour. The user can then directly manipulate the formula by moving and adjusting the rectangles. This technique is limited to showing the dataflow for a single, user selected, cell with no way of displaying the

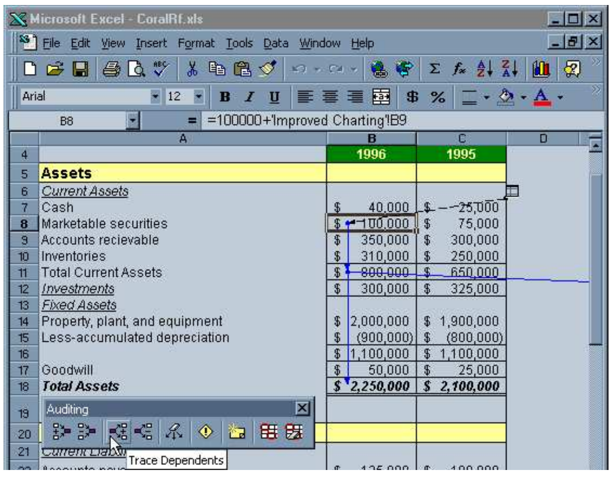

Microsoft Excel - CoralRf.xls
File Edit View Insert Format Tools Data Window Help
Arial
12
LOG
Improved Charting
Formatting
Formulas (2)
Edit
NUM

=B17+B16+B12+B11

	A	B	C
4		1996	1995
5	Assets		
6	Current Assets		
7	Cash	$ 40,000	$ 25,000
8	Marketable securities	$ 100,000	$ 75,000
9	Accounts recievable	$ 350,000	$ 300,000
10	Inventories	$ 310,000	$ 250,000
11	Total Current Assets	$ 800,000	$ 650,000
12	Investments	$ 300,000	$ 325,000
13	Fixed Assets		
14	Property, plant, and equipment	$ 2,000,000	$ 1,900,000
15	Less-accumulated depreciation	$ (900,000)	$ (800,000)
16		$ 1,100,000	$ 1,100,000
17	Goodwill	$ 50,000	$ 25,000
18	Total Assets	+B12+B11	$ 2,100,000

Figure 1: Using Excel's Range Finder to examine a formula.

overall structure of the spreadsheet. It is also of limited use when the dependencies span large distances, as the user will need to hunt round to find the highlighted boxes.


Microsoft Excel - CoralRf.xls
File Edit View Insert Format Tools Data Window Help
Arial
12
B8
Auditing
Trace Dependents

=100000+'Improved Charting'!B9

	A	B	C	D
4		1996	1995	
5	Assets			
6	Current Assets			
7	Cash	$ 40,000	$ 25,000	
8	Marketable securities	$ 100,000	$ 75,000	
9	Accounts recievable	$ 350,000	$ 300,000	
10	Inventories	$ 310,000	$ 250,000	
11	Total Current Assets	$ 800,000	$ 650,000	
12	Investments	$ 300,000	$ 325,000	
13	Fixed Assets			
14	Property, plant, and equipment	$ 2,000,000	$ 1,900,000	
15	Less-accumulated depreciation	$ (900,000)	$ (800,000)	
16		$ 1,100,000	$ 1,100,000	
17	Goodwill	$ 50,000	$ 25,000	
18	Total Assets	$ 2,250,000	$ 2,100,000	

Figure 2: Excel's built-in auditing tools. Note how the reference to a cell is not fully visible and a sheet icon is the only indication of the link between sheets.

The second built in auditing tool has the ability to display interconnections between cells by tracing cell precedents and dependents using arrows. An example is given in figure 2. For a cell A, precedent arrows will point to A from all cells that are referenced in A's formula, showing the location of the source data, or the ancestors, for A. Dependent arrows show the flow for data subsequently calculated using the data in the selected cell, that is, a cell B will have dependent arrows to all cells that reference it in their formulas (the descendants). These arrows also serve to allow the user to move between spatially disjoint, but logically connected, cells by double-clicking the arrowheads. This technique is referred to as semantic navigation by Takeo Igarashi et al. (Igarashi, Mackinlay, Chang & Zellweger 1998) who observe that it provides cues about the relations among cells rather than the superficial spatial continuity, serving to make the hidden dataflow patterns more transparent. They also remark that "complicated spreadsheets can create a tangle of arrows, making it difficult to see the relationship among cells."

An additional limitation of the current auditing tools is the treatment of ranged references, with Excel only depicting a minimal containing box and a single reference arrow. This is particularly an issue with intersection references, where the dependencies may be potentially greater than those cells immediately referenced. For example, a change in value of a remote cell could cause the intersection region to expand.

Another function of the auditing tools is for highlighting circular references to users, which are typically the result of an addressing error. Excel generally has acyclic relationships between cells (Yoder & Cohn 2002, Chadwick, Knight & Rajalingham 2002), which creates a tree like dependency structure (or a forest of trees due to multiple roots). It should be noted that this is not always the case as Excel also provides a bound iterative calculation mode for working with specialised circular reference problems.

Multiple trees can share common branches, and in such cases it can be difficult to trace all the connected trees using the inbuilt auditing functions. This is primarily due to Excel only tracing dependencies in one direction at a time, requiring multiple traversals by the user to trace the entire structure.

2.2 Related Work

Spreadsheets and other visual programming languages are currently an active area of research, with many fields of focus being addressed. These range from devising new ways to interact with applications, theoretical models to describe the underlying principles, methods for detecting and correcting errors, and the cognitive issues of programming.

Jorma Sajaniemi presented a theoretical model of spreadsheets along with a description of various spreadsheet auditing mechanisms employing the model (Sajaniemi 2000). Brad Myers created C32 (Myers 1991), which uses graphical techniques along with inference to specify constraints in user interfaces. These constraints are relationships that are declared once and then maintained by the system. Burnett makes the following observation about C32: "Unlike the other spreadsheet languages described, C32 is not a full-fledged spreadsheet language; rather, it is a front-end to the underlying textual language Lisp used in the Garnet user interface development environment." (Gottfried & Burnett 1997, pg 2) Wilde's work on the WYSIWYC spreadsheet (Wilde 1993) aims to improve traditional spreadsheet programming by making cell formulas visible and by making the visible structure of the spreadsheet match its computational structure.

2.2.1 Igarashi and fluid visualisation

Takeo Igarashi et al. describe spreadsheets as augmenting "a visible tabular layout with invisible formulas" (Igarashi et al. 1998). They observe that while the transparent nature of formulas allows the cells to be used for both presentation purposes and as programming variables, the access to the formulas and their resulting dataflow structure is often difficult, resulting in significant cognitive overhead for users. In 1998 they published a paper (Igarashi et al. 1998) documenting the creation of a set of fluid visualisations that help the user address the hidden dataflow graphs and superficial tabular layouts of spreadsheets. They were designed to improve the users understanding of the dataflow structure by enabling them to visually interact with the obscured structures, while maintaining the original appearance of the spreadsheet. An example image taken from the paper is shown in figure 3.

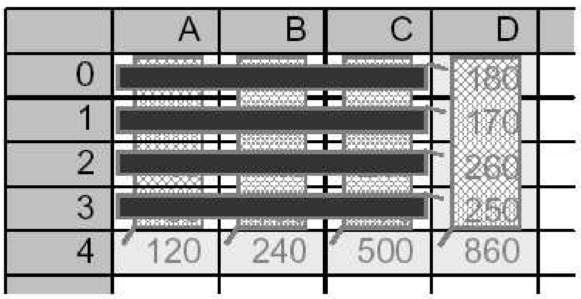


Figure 3: Static global view produced by Takeo Igarashi et al. to visualise the entire dataflow graph at once.

The invisible formulas can affect both the users who create the spreadsheet and those who may later try to understand and modify it. To fully understand a spreadsheet, Takeo Igarashi et al. observe that a new "user must repeatedly select a cell, read the formula, and move on to the next cell, until he has seen enough formulas to get an overview of the spreadsheet. As spreadsheets get larger and more complicated, the overhead of understanding shared spreadsheets increases dramatically." (Igarashi et al. 1998)

This cognitive overhead is undesirable for any user, who will typically be more interested in solving the problem at hand rather than tracking the exact structure of the spreadsheet.

The diagrams produced use a range of techniques to display the information of interest. Selections of them are static and can display large amounts of information at any one time. The drawback of these diagrams is that they can develop many overlapping features, resulting in a sometimes cluttered appearance. As such, Igarashi only suggests their use for gaining a general overview of the entire structure. More advanced visualisations make use of animation to reveal the structure as the user interacts with the spreadsheet. These visualisations can allow the user to navigate and edit the dataflow structure more effectively, and perform interactive graphical induction that is more expressive than the regular patterns achievable using the $ symbols. It is also noted that users often make errors when using absolute references, mainly due to not knowing where to place the $ symbol. Igarashi et al. make the following comment:

> A related problem occurs when a formula is used to fill a region of a spreadsheet. Current spreadsheet applications adjust the cell addresses in the formulas by the distance from the source formula, which again may or may not be what the user desires. To inhibit these adjustments, the user can specify a cell in the formula to be an absolute reference (by using the "$" symbol), but it is difficult to place the $ symbol correctly, which means the filled formulas must also be checked to make sure they are correct. (Igarashi et al. 1998)

Although the techniques presented showed potential in many areas, Takeo Igarashi et al. mention that in future work there is a need to integrate these diagrams into a more realistic spreadsheet program. This is one area that our project addresses, by creating many of the visualisations independently of the application vendor while still working with actual spreadsheets. It will however share the same limitations for visualisations that need to be used interactively at runtime.

2.2.2 Clermont and automated spreadsheet auditing

Markus Clermont has written papers discussing the conceptual differences between traditional programming practices and those undertaken with spreadsheets, and is currently researching techniques for debugging excel-spreadsheets (Clermont, Hanin & Mittermeir 2002, Ayalew, Clermont & Mittermeir 2000).

At the crux of the difference discussion is the assertion that "Software is written in a professional manner by Professionals; Spreadsheets are written by End-Users!" (Ayalew et al. 2000). This indicates the contrasting backgrounds between those who write traditional software, and those who write spreadsheet programs. People trained in software engineering should develop traditional software systems from a well-founded design. Those who write spreadsheets require no such training and possibly no formally described design.

Often this lack of training and design is a non-issue, as only a quick and dirty solution to a small problem is required. However, such write-and-throw-away spreadsheets, or "scratch pads", only contribute to a small portion of the actual spreadsheets created. Clermont points out that:

> there is a rather neglected proportion of spreadsheets that are periodically used, and submitted to regular update-cycles like any conventionally evolving valuable legacy application software. However, due to the very nature of spreadsheets, their evolution is particularly tricky and therefore error prone. (Clermont et al. 2002, pg 3)

While it is clear that most end-users have different background to software engineers, spreadsheets help level the playing field in some respects. One of the most useful abilities of a spreadsheet is the ability for users to enter raw data and formulas while being shielded from the low-level details of traditional programming (Ayalew et al. 2000). Clermont et al. reason that this allows the users to utilise the skills from their profession when expressing themselves on the spreadsheet without having to first relate their concepts to a corresponding programming concept. In many ways, the two-dimensional tabular arrangements of numbers interspersed with explanatory text seems familiar, and similar to how they would express the problem using pen, paper and a calculator.

In more recent work on errors, Clermont et al. have developed a classification system for the types of equivalence between different formulas (Clermont et al. 2002). As they have presented this system, when comparing two formulas they find either no-equivalence or:

- **Copy-Equivalence**, which exists if "the formulas are absolutely identical (i.e. the cell contents has been copied from one cell into the other, either by copy and paste, or by retyping the same formula.)"
- **Logical-Equivalence**, which exists if "the formulas differ only in constant values and absolute references."
- **Structural-Equivalence**, which exists if "the formulas consist of the same operators in the same order, but the operators may be applied to different arguments."

This classification system has significance for our project when looking for patterns through visualisation. Different types of equivalence can be used to conceptually group sets of formulas together before visualisation.

One particular form of error from this recent work, that also has significance to this project, is the result from the replication feature. If a user detects an error in a replicated block of values, rather then track down the source of the bug, they may opt to perform a quick fix by entering the correct value or a new formula only for the effected cells. This erroneous correction will only aggravate the problem, because the formula is showing an incorrect value due to an error in another cell and this relationship is destroyed by "this pseudo-corrective act in the value domain" (Clermont et al. 2002). The error detection methodology suggested in the research will help auditors detect such a pseudo-correction using the equivalence classes, as the irregularities are not based on causes of errors. Clermont et al. suggest that this technique allows correction to be focused and is thus easier to perform.

2.2.3 Panko and spreadsheet errors

Since the late eighties Raymond Panko has written numerous papers relating to the spreadsheet paradigm and end-user programming in general. The focus ranges from introducing basic rules for creating spreadsheets, to detailed explanations of the causes of errors and methods for detecting them.

The methodologies Panko and others are using to find these errors are developing in maturity and verifiability. In the earlier cases, much of the discussion was only speculation using anecdotal evidence. Now most research relies on empirical data derived from "the realm of systematic field audits and laboratory experiments" to back up their claims (Panko 1998).

Panko collected the data from a range of such research projects conducted before 2000, and collated the data in a table with comments on the methodology used, along with the cell error rates and percentage of models with errors. Across the diversity of techniques present in the complied data, he found a common pattern: "every study that has attempted to measure errors has found them and has found them in abundance." (Panko 1998)

He has observed that errors have a tendency to occur in a few percent of all cells, resulting in a question of not if errors exist, but rather how many errors there are in larger spreadsheets (Panko 1998). The percentage can vary considerably depending on the auditing methodology used and particular application the spreadsheet is being used for. Through practical experience most consultants gave the conservative estimate that between 20 and 40% of spreadsheets contain errors (Panko 1997*b*), with the number rising as high as around 90% for larger spreadsheets.

These errors are generally attributed to human error, with errors during programming typically occurring for 5% of all actions performed by the user. This number is itself derived from a series of empirical studies that Panko presents on "The Human Error Website" (Panko 1997*a*).

Many of the studies on spreadsheets involve using laboratory data, but a portion also involved the use of operational data from real world problems. This can be important for obtaining results that are more representative of current practices.

Despite the wealth of information that Panko and others have found to indicate the alarming error rates in spreadsheets, they have also found that many users are still overconfident of their abilities to program error free spreadsheets. As a result, a significant portion of the human errors go undetected due to programmers not taking steps to reduce the risk of errors. Panko reasons that this behaviour is partially due to the reluctance of people to do formal testing, and follow other tedious disciplines, allowing them to save time and avoid onerous practices. This is followed by the observation that the errors that are caught only serve to further convince the users of their efficiency (Panko 1997*b*, pg 14). More still may dismiss errors, as many syntactic errors are automatically detected and brought to the users' attention by the program.

One of the most interesting observations that Panko has made is that spreadsheets probably contribute the largest portion to the development of large-scale end-user applications in current times (Panko 1997*b*, pg 2). This view is important as many regard spreadsheets as tools for solving "small and simple scratch pad applications" by single individuals that are disposed of shortly after computation is complete. The truth is that there are a sizeable number of spreadsheets that are both large and complex, with their development involving multiple people and often spanning significant periods of time.

Another interesting result from surveys that Panko and others have undertaken with companies is that, although it is agreed that the error rate numbers are too high, there is a general consensus that comprehensive code inspection is simply impractical. Which Panko summarises as implying that companies "should continue to base critical decisions on bad numbers."

One conclusion that can be drawn from the high level of errors, but general reluctance to check for them, is the need for easier methods of reducing most causes of errors without consuming time and other valuable resources. Increasing user awareness of the structure and meaning of a spreadsheet through visualisation holds promise as one technique to partially address this need.

2.2.4 Burnett and spreadsheet visualisation

Margaret Burnett is an active researcher in the field of visual programming languages. Of particular relevance to this project is her work as the principal architect of the Forms/3 visual language. Forms/3 is a tabular form based visual language that has several features similar to spreadsheets, such as the parallel between its form linking mechanism and spreadsheet formulas. Using this language it has been possible to research areas and methodologies that would not have been possible, or at least difficult, with many closed source commercial spreadsheets due to the requirement for seamless integration.

The importance of seamless integration is to maintain the consistency of the spreadsheet paradigm. Burnett takes this requirement to mean that any approach "follows the declarative, one-way constraint paradigm of spreadsheets, emphasizing that it should follow the value rule for spreadsheets" (Gottfried & Burnett 1997, pg 1). The definition she gives for the value rule is taken from Kay (Kay 1984) and states "that a cell's value is defined solely by the formula explicitly given to it by the user".

Many of the additions that were made are related to a hypothesis about the spreadsheet model. This hypothesis is:

> that spreadsheet reliability can be improved if the spreadsheet users work collaboratively with the system to communicate more information about known relationships. Spreadsheet users know more about the purpose and underlying requirements for their spreadsheets than they are currently able to

communicate to the system, and our goal is to allow end users to communicate this information about requirements. (Beckwith, Laura, Burnett & Cook 2002, pg 1)

Motivated by the high degree of errors present in spreadsheets and the desire to reduce the cognitive load on the user, Burnett and others have developed a testing methodology that applies software visualisation techniques to support testing of Forms/3 programs (Rothermel, Cook, Burnett, Schonfeld, Green & Rothermel 2000). This testing methodology was designed to help end users with the correctness of their spreadsheet programming by allowing them to incrementally edit, test, and debug their spreadsheets in a visual way as the model evolved. The approach, referred to as WYSIWYT ("What You See Is What You Test"), augments the spreadsheets interface with additional information that provides visual feedback through several techniques about the degree a spreadsheet has been tested.

One of the additions to the spreadsheet interface were dataflow arrows that show dataflow paths among cells and, when formulas are showing, they also show the interactions between formula sub-expressions and the "testedness" of each cell via colour. These arrows are an optional part of the interface, and to avoid adding to much clutter, each cell's arrows are transient and appear/disappear when the user clicks on the cell (Burnett, Sheretov, Ren & Rothermel 2002, pg 23).

Early research undertaken on the WYSIWYT methodology worked at the granularity of individual cells. This approach worked well for smaller spreadsheets, as demonstrated by studies conducted on testing, debugging, and maintenance tasks with the help of WYSIWYT (Beckwith et al. 2002, pg 3). However, this technique often put an unnecessarily large burden on the user for more substantial spreadsheets, which would often contain large grids that were fairly homogenous, i.e. "they consist of many cells whose formulas are identical except for some of the row/column indices" (Burnett et al. 2002, pg 4). This led Burnett and her colleagues to address a matter of necessity for real-world spreadsheets: "how to establish scalable guard mechanisms that are viable for end-users when programming spreadsheets." (Burnett et al. 2002)

A research effort relevant to our project was the creation of a cell relation graph in earlier testing work (Burnett et al. 2002, pg 7). This model consists of a collection of nodes that each form part of a larger formula graph model. In the formula graphs an entry node models initiation of the associated formula's execution, an exit node models termination of that formula's execution, and one or more predicate nodes and computation nodes, modelling execution of if-expressions, predicate tests and all other computational expressions, respectively. Edges in this graph control the flow between pairs of formula graph nodes. Out edges from predicate nodes are labelled with the value to which the conditional expression in the associated predicate must evaluate for that particular edge to be taken.

Cells also form an integral part of this model because of their role as variables. Each cell has a corresponding node in the formula graph that represents the expression defined in that cell. This node also contains details that the cell is either for computational use (a non-predicate node refers to it) or a predicate use (an out-edge from a predicate node that refers to it) (Burnett et al. 2002, pg 8).

3 Toolkit

Spreadsheet processing for this project utilises an application toolkit that collectively provides the functionality to extract low level structures from spreadsheet files, analyse these structures, produce visualisations displaying the information analysed, and to find, download, and persistently store spreadsheets located around the Internet. The code for these applications resides in 6 core Java packages. Each package concentrates on providing a distinct function of interest, such as finding the spreadsheets on the Internet or producing the diagrams from the processed data.

The most essential component in the toolkit involved reading the binary Excel files and reconstructing the data present as Java objects. This is done by the Extractor application, which provides transparency between the binary forms of Excel files stored on disk and the toolkit's internal representation.

The main artifact of interest for extraction was the cell. For all the occupied cells found, it was necessary to extract its value and formula if present (both in a string format that matches that shown to the user). In addition to this data, it is also important to obtain row and column position as well as the worksheet that it resides in.

Further details on the extraction process are available in a prior paper on low level structure access and visualisation (Ballinger, Biddle & Noble 2003).

For the entire corpus, or a subset, the Analyser application examines a set of files for a feature of interest and collates the information in an output format suitable for the form of analysis undertaken. Within corpus linguistics such a tool is referred to as a concordance program.

With the representation of Excel files stored in the toolkit's easily handled Java format, it is possible to analyse and perform aggregation operations on any observable spreadsheet component. In the simplest cases this can be the position of occupied cells and the value they contain. In more complex cases the details of interest might be the results of running the formula parser to extract referencing operators, functions, and primitive components. The data obtained can then be combined using various methods to create the source information for the construction of visualisations.

4 Visualisations

In the previous sections, we have seen how studies show the effects of hidden structure in spreadsheets can be significant, and further show that there are strategies to address these effects. In this section, we demonstrate how our toolkit can be used to access the structure of spreadsheets and generate visualisations to show the otherwise hidden structure. We have developed a number of specific visualisations, based on the spatial structure of spreadsheets, the structure of logical dependencies within spreadsheets, the way these two structures work together, and the way these aggregate in corpus analysis. Together, these demonstration visualisations show the feasibility of using external visualisation to assist end-users.

4.1 Visability of spreadsheet layout

Understanding where the information is located is the first step in acquiring a greater depth of knowledge about what structures and patterns are present in a spreadsheet. It also provides a stable foundation for building new knowledge. When a user is first presented with an unfamiliar spreadsheet they will often scroll around the various sections looking for larger blocks of data and cells that output final results.

At any one time only a portion of the entire grid is displayed on the screen. The actual number of cells that are present in any one screen varies due to the width and height of the columns and rows respectively. One technique Excel supports to help this process is to use the zoom tool to increase the number of cells visible on the screen. Although generally effective for smaller sheets, this approach can miss data that has been deliberately positioned towards corners of a spreadsheet. Generally the technique doesn't scale well for larger spreadsheets, as the required zoom level can make it difficult to identify occupied cells.

During this early process of discovery most users are more interested in obtaining a general orientating view of the layout than the exact values and formulas present in each cell, which only become relevant for later tasks.

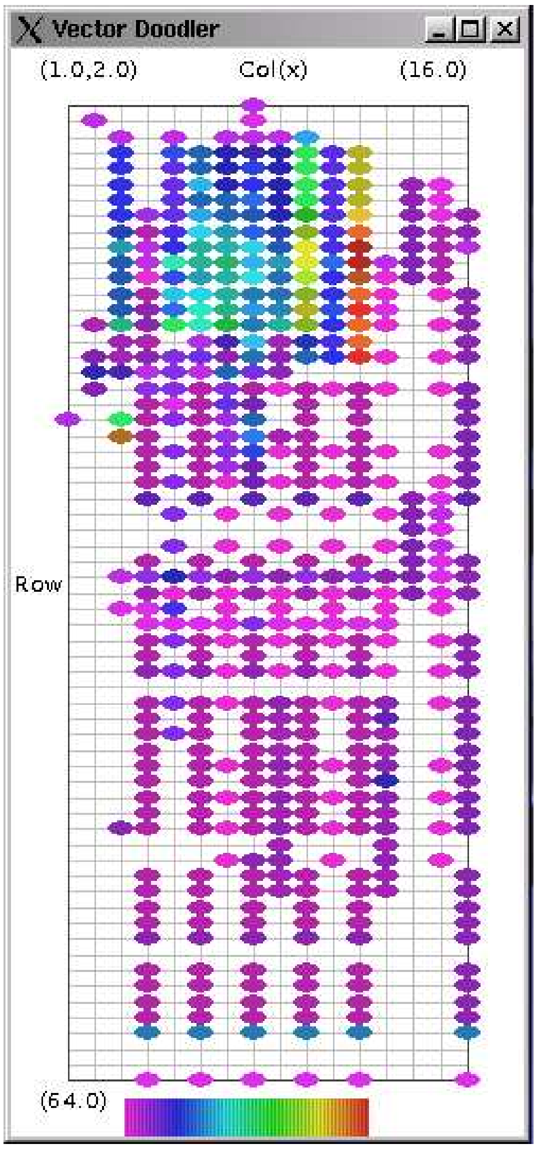


Figure 4: Real-estate utilisation diagram in 2D

An example of the 2D real-estate diagrams produced by the toolkit is shown in figure 4. This visualisation is produced using the Java Swing library's primitive components, such as lines and circles. Any coordinate with a cell count greater than zero is assigned a coloured circle. The colour for this circle is determined as to create a heat-map effect for all the data. Grid coordinates where a large number of cells occur will be coloured towards the red end of the colour spectrum while those with lower counts will be coloured towards the violet/blue end.

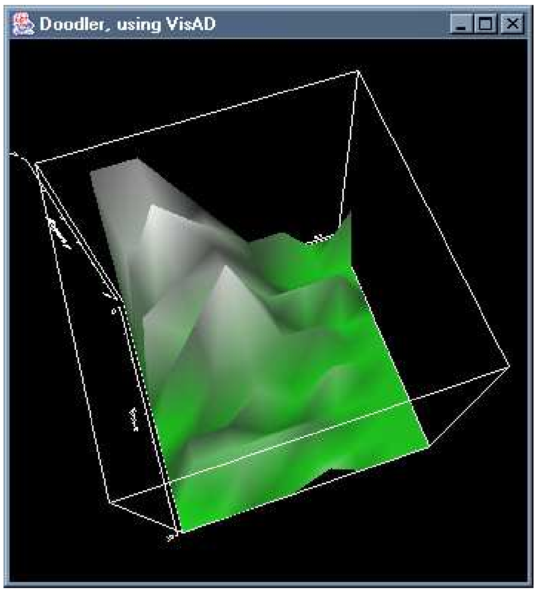


Figure 5: Real-estate utilisation diagram in 3D

To create an alternative view for this information, the data was projected into 3D to create a surface map. This transformation from discrete data to continuous surface can benefit the viewer by smoothing out the effects of any one cell and aiding understanding. It is possible to maintain hints of the discrete nature of the data by colouring the surface as a square grid rather than a continuous colour gradient.

The 3D rendering was produced using Java3D (Sun 2002) and VisAD (Hibbard 2002), a visualisation tool for numerical data. These tools had the benefit of allowing the user to interact with the image by rotating and zooming via the mouse and keyboard. Through this interaction the true benefit of the 3D model is gained, giving the image the feeling of solidity, continuity, and real existence. An example of this type of diagram is figure 5. In this figure the left axis contains the rows and the upper (obscured) axis the columns. The altitude represents the occupancy level and is coloured to create a topological terrain map appearance.

4.2 Applying clustering to layout

Both the layout images mentioned above rely on the user to observe the blocks of data themselves. Al-

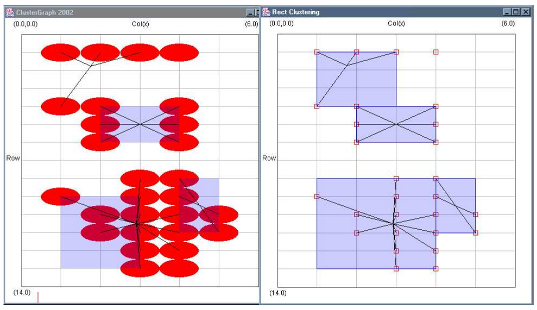


Figure 6: Two variations on rectangular clustering in a single worksheet.

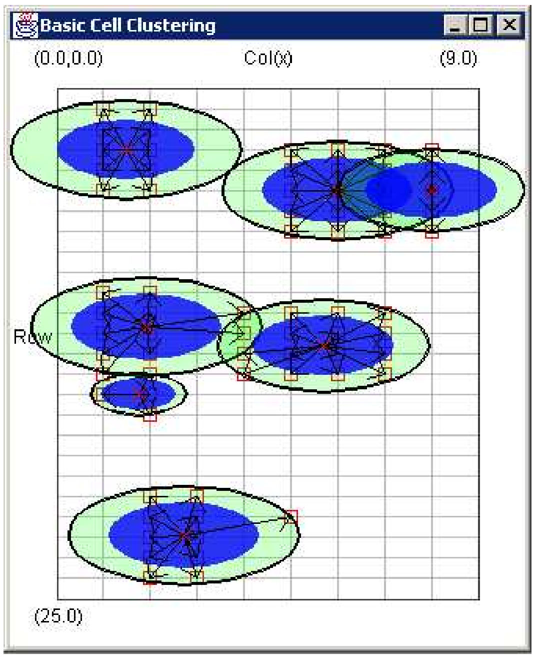


Figure 7: Circular clustering in a single worksheet.

though this technique is generally effective, it is also possible to highlight certain spatial relationships programmatically.

Clustering is a technique that partitions records into clusters (groups) that are similar according to two or more common attributes. Conversely, records in separate clusters are dissimilar according to the same attributes. To determine if an instance is part of a cluster a distance function is used to calculate similarity and a user defined cut-off value determines membership. Generally, as the cut-off value gets larger the clusters become bigger and engulf more instances.

When the clusters are graphed they provide a useful visual cue as to the relationship between several instances of some type, perhaps further simplifying the process of identifying blocks of data. Figures 6 and 7 are examples of the clustering visualisations that the toolkit can produce. When the clusters are displayed using rectangles only the minimal containing box is shown. Alternatively, with circular depictions the inner blue circle is the average instance radius for the cluster and the outer circle is the maximum radius.

The main use of the clustering algorithms to date has been looking at the spatial relationships between cells using the Euclidean distance between them as the similarity metric. Using alternative metrics in defining a new distance function would allow the technique to be extended to look for non-spatial clustering relationships.

4.3 Data Dependency Flow

In a similar fashion to Excel's built in auditing tools, it is possible to trace all inter-cell dependencies in a single diagram. Due to the magnitude of each vector such a diagram can become cluttered. To address this two simple alterations to the vectors displayed can reduce the volume of information displayed. Firstly, the magnitude of each vector could be altered to the equivalent unit vector. This removes the consideration of spatial distance and instead concentrates of flow direction. The second approach is to display just the average outgoing vector for each cell. These two techniques are combined to create a visualisation depicting the direction of dataflow. Figure 8 is an interesting example of this technique and shows a trend for the flow to curve back towards the origin. This particular visualisation is created using VisAd. As the inter-cell references can flow between spreadsheets the unit vectors have 3 dimensions. This can be made more apparent to the user by displaying all the vectors in 3D, as shown in figure 9.

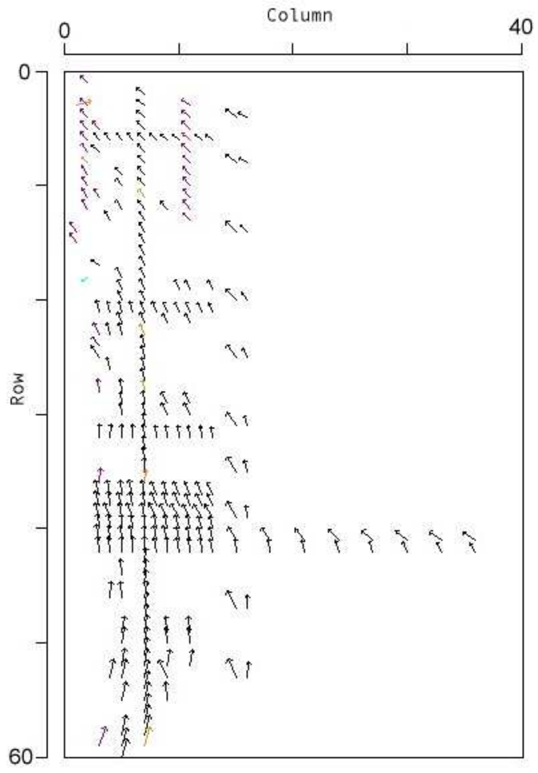


Figure 8: Data dependency flow using the average unit vectors in 2D.

4.4 Data Dependency Direction

While the data flow diagrams involve the spatial positions of cells in the information, it is also possible to concentrate purely on the directions on flow. This is done using a visualisation referred to as compass or radar view. This visualisation is applied to dependency direction data contained in a series of buckets. Each of the 36 buckets is created to store a count of the number of outgoing vectors that occur in the corresponding angle, e.g. 0 to 10 degrees for the first bucket.

Storing the bucket data in a comma separated value file allows the data to be read by Excel. Excel's graphing features can then be used to display the data as either a radar or line graph, as the example figure 10 demonstrates. It is clearly visible in these graphs that the rectangular grid layout of a spreadsheet encourages many of the inter-cell references to be either vertical or horizontal. After the four main axis the next significant measure occurs in the region between 300 to 360°.

4.5 Graph structure

It is possible to view the hidden graph structure for each spreadsheet independently of spatial considerations.

The first example of such a visualisation is a spring view, which uses a 2D layout algorithm based on the idea of spring forces to arrange the spatial positions

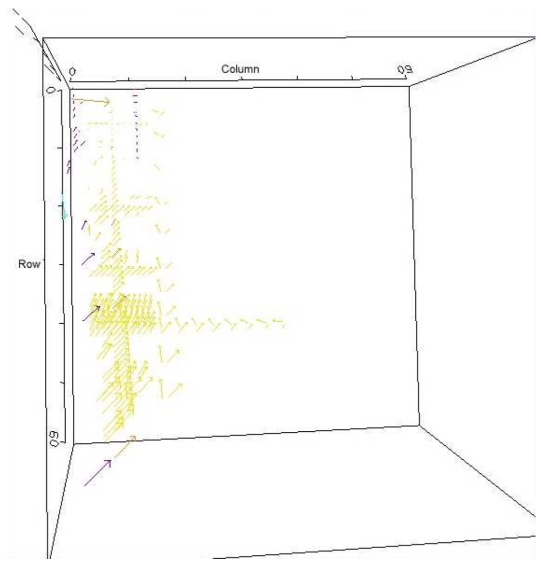


Figure 9: Data dependency flow using the average unit vectors in 3D.

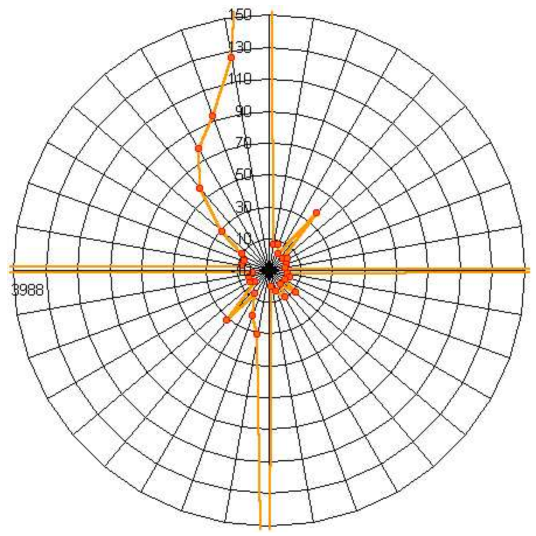


Figure 10: Radar graph of bucket data for 259 spreadsheets from the corpus. The upper bucket count is cropped at 150

of the cells, with an example given in figure 11. To do this, cells and the edges between them, which represent the inter-cell dependencies, are automatically arranged based on the internal structure of the graph. If two cells are connected via dependencies they are attracted to each other, otherwise they are pushed apart. If the algorithm is iterated a few times, the graph reaches a stable position and does not move anymore.

This has several benefits, such as the ability to untangle many complex dependency structures. When the structure has untangled it is of interest to observe if cells that were previously spatially related are still spatially related, or if instead they have drifted apart. If more than one tree structure is inserted into this visualisation they will often separate completely, as there are no attractive forces between them. Due to its nature, this visualisation is dynamically manipulable by the users, who can drag cells around to aid in the untangling process or fix them in position to enforce a particular structure (fixed cells are coloured red).

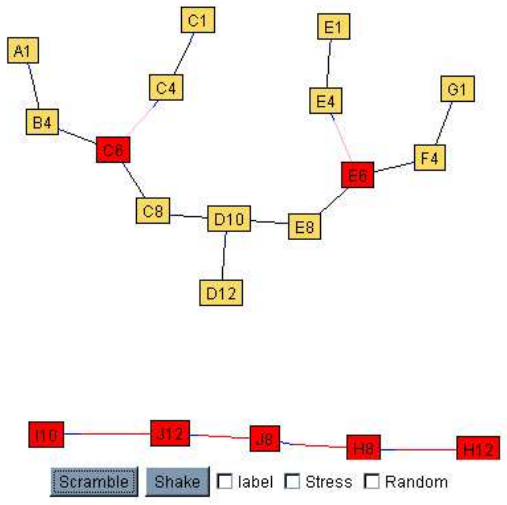


Figure 11: A spring view of the dependency structure between cells. By disregarding the spatial bounds usually enforced on cells, structures such as the chain between cells I10 and H12 become clearer.

4.5.1 Fisheye view

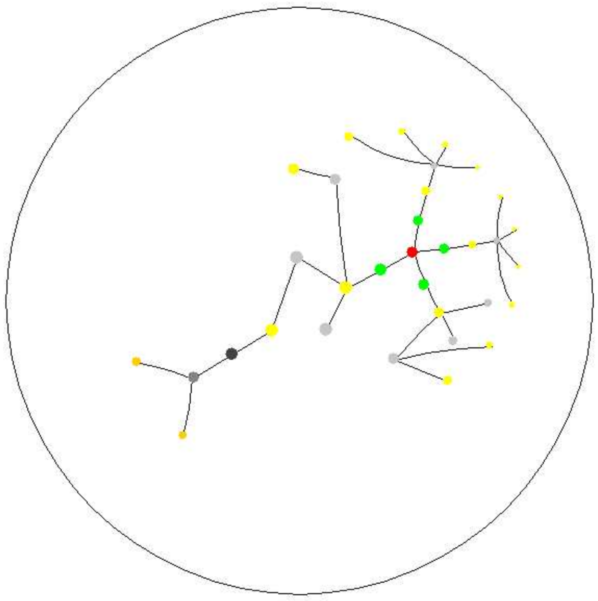


Figure 12: The fisheye view of 4 dependency trees.

The tree structures resulting from formula dependencies can span large numbers of cells over great spatial distances, making it difficult to view all the information and still derive useful patterns. The fisheye visualisation involves warping this tree over a hyperbolic lens that makes it possible to achieve both focus on an aspect of interest near the centre of the visualisation and the larger context for that aspect. Figure 12 is an example of 4 trees in a single worksheet being arranged around a red artificial root node. Alternative renderers are available for use with this visualisation to add additional information, such as the cell address.

4.6 Detailed inspection of formula

After examining the layout of the spreadsheet it is useful to then focus on the dataflow through the

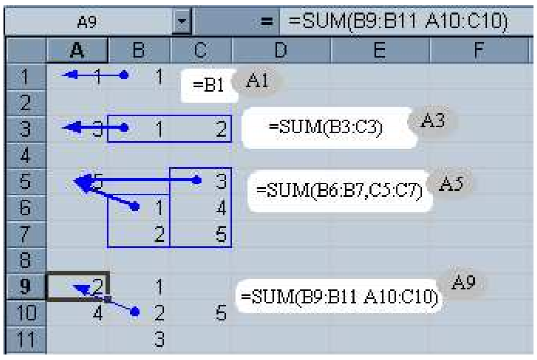


Figure 13: Excel's precedent trace auditing tool for sheet 1. Note that we have added by hand boxes to the right of each cell containing the relevant formula.

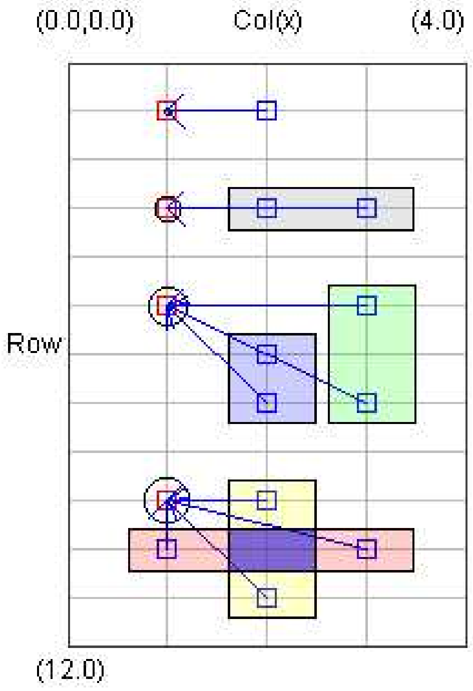


Figure 14: Formula extracted and displayed by the toolkit.

spreadsheets created by formula. As mentioned in the background section, Takeo Igarashi et al. observed that the process of dataflow discovery often involved clicking on individual cells and tracing the formula manually, putting an unreasonable strain on the user. Excel does provide some assistance via the range finder and auditing tools, but they were found to be lacking for complex and large spreadsheets. It is important that any process devised not put a heavy load on the user to manually trace the dependencies that exist.

The first stage in creating a visualisation for displaying these dataflow structures is to consider all the components individually. A formula can be considered to create both spatial and logical relationships (dependencies) between cells, both in two and three dimensions. Making these dependencies more accessible to the user is the primary aim of many of the following visualisations.

Figure 13 demonstrates the four main techniques that Excel allows a user to use in a formula to reference other cells and the precedent trace arrows added by Excel's built in auditing tools. Notice how Excel's

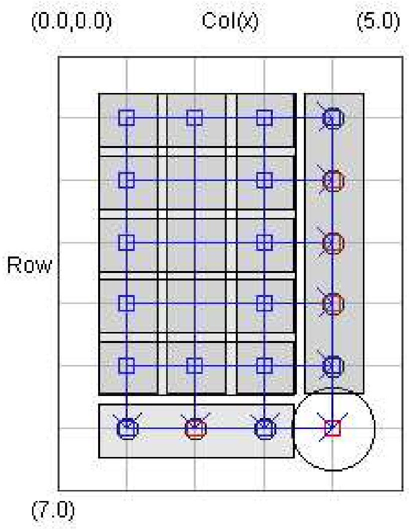


Figure 15: Using the summation operation on both the columns and rows in a table.

trace is somewhat deceptive in the case of an intersection, in this case looking more like an individual cell reference. In figure 14, created by the toolkit, a precedent cell dependency is represented by a blue arrow in a similar fashion to that presented by Excel's auditing tools. As with the real-estate visualisations, the layout is designed to mimic that of Excel. However, as the diagrams being presented here are unconcerned with actual cell values, they are omitted from the diagram, significantly reducing the amount of information that the user has to process. Any ranged references are depicted using a shaded box. A single range is shaded light grey while a union has the left and right sub-ranges coloured blue and green respectively. Intersections use yellow and red boxes for the left and right sub-ranges. The actual resulting intersection is shaded dark blue. The use of shading and transparency in these diagrams would not be as viable within Excel as they would obscure the cell values.

An extremely common spreadsheet operation is to sum the values in a range of cells. Figure 15 demonstrates such an operation with a more realistic style of worksheet where a series of columns are summed and then cross-checked with the sum of the rows. The circles in the diagram are indications of the complexity of the formula in those cells. Note how the bottom right cell is significantly more complex than those that just sum a single row or column. This style of image is similar to the static global view presented by Igarashi et al. (Igarashi et al. 1998).

A more specialised form of precedent tracing not supported by Excel's auditing tools involves examining the referencing syntax. Relative and absolute referencing between cells creates two different forms of dependency, those that change in a formula as the replication commands are applied and those that remain constant in one or two dimensions. Using the replication command in conjunction with combinations of relative and absolute references explicitly encourages the user to create regular patterns in the dependencies. Making these patterns more apparent through visualisation was one of the focuses of our project.

Figure 16 demonstrates the colouring of relative and absolute cell references used by the toolkit. The reference from B1, coloured blue, is a standard relative reference (Excel's default reference). B3 has a red absolute reference. A2 and C2 are both par-

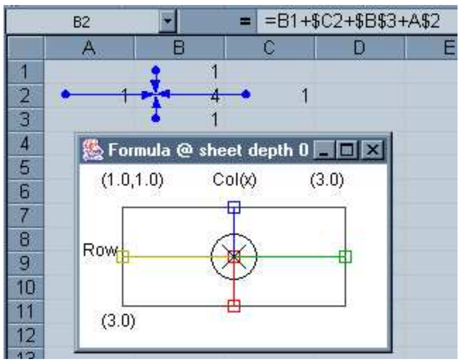
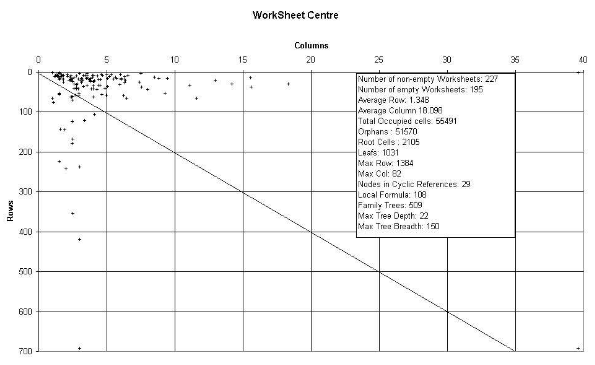


Figure 16: Both Excel's and the toolkit's trace of simple absolute and relative inter-cell dependencies.

tially absolute references, in that only one axis is fixed with the $ symbol, and are coloured yellow and green respectively. When these colourings are applied to complex real-world spreadsheets, the regular patterns used in cell referencing can become more apparent.

One hypothesis that we had from an early stage is that absolute cell references will be one of the main causes of long and angled dependency vectors. The first motivation for this hypothesis is that locality encourages regions where a large number of inter-cell dependencies exist to be spatially close to each other. This would imply that most cell references that refer to a distance cell would in fact be referring to a single parameter. The exception to this reasoning would be caused by presentation considerations, which may encourage the programmer to duplicate or refer to large ranges across some distance.

4.7 Corpus Analysis



Number of non-empty Worksheets: 227
Number of empty Worksheets: 195
Average Row: 1.348
Average Column 18.098
Total Occupied cells: 55491
Orphans : 51570
Root Cells : 2105
Leafs: 1031
Max Row: 1384
Max Col: 82
Nodes in Cyclic References: 29
Local Formula: 108
Family Trees: 509
Max Tree Depth: 22
Max Tree Breadth: 150

Figure 17: Worksheet centre for a corpus of 259 workbooks. Data is plotted in Excel using data exported from the toolkit.

In this section we briefly demonstrate visualisation of a corpus of spreadsheets. In figure 17 the spatial centre for each worksheet in a corpus of 259 workbooks is plotted. Some interesting observations include the trend towards a centre that has the column as the majority component. Also, the table containing data about the run reports a large number of orphan cells (no incoming or outgoing references).

After applying the toolkit's parser to the formula in each cell it is possible to count the functions utilised in each worksheet. Using this data and the Excel defined categorises for each function figure 18 is produced. This bar graph addresses the degree to which each function is utilised in the corpus. Indications

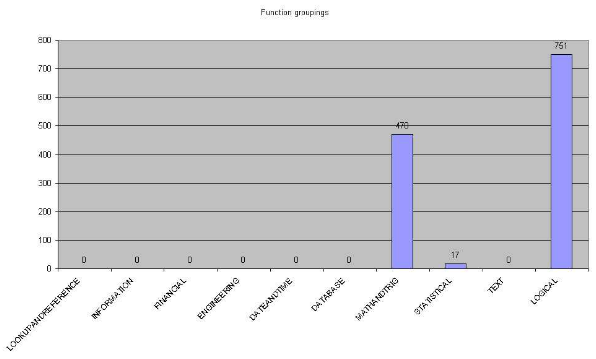


Figure 18: Function utilisation in a corpus of 259 spreadsheets. Data is plotted in Excel using data exported from the toolkit.

from this graph are that users only utilise a relatively small subset of commands. To draw reliable conclusions the size of the corpus used to generate this image will need to be greatly increased. Consideration would also have to be made as to the abilities of the extraction process to read certain functions and their associated source data.

This diagram is motivated by the hypothesis that large portions of users succeed in using spreadsheets while only utilising a very small subset of available functions, with the primitive operators and most basic functions, such as sum, average, and logical primitives, making up the majority of cases.

5 Conclusions

Spreadsheets are by far the most significant form of end-user programming. In this paper we have explored the subject of visualisation to assist end-users in better understanding the structure of spreadsheets. In particular, we addressed the issue of "hidden structure". We first reviewed the literature to investigate the implication of hidden structure on spreadsheet users. We then used a toolkit we developed and demonstrated the feasibility of externally accessing the spreadsheet structure, and generating visualisations helpful to end-user understanding. We believe this is a good foundation for future work on spreadsheet visualisation, including detail user studies, domain specific visualisation, and spreadsheet corpus analysis.

References

Ayalew, Y., Clermont, M. & Mittermeir, R. T. (2000), Detecting errors in spreadsheets, *in* 'EuSpRIG 2000 Symposium: Spreadsheet Risks, Audit and Development Methods'. University of Greenwich, London. `http://citeseer.nj.nec.com/485147.html`.

Ballinger, D., Biddle, R. & Noble, J. (2003), Spreadsheet strucuture inspection using low level access and visualisation, Vol. 4, Australasian User Interface Conference, Adelaide. `http://www.mcs.vuw.ac.nz/~db/publications/`.

Beckwith, Laura, Burnett, M. & Cook, C. (2002), Reasoning about many-to-many requirement relationships in spreadsheets, *in* 'IEEE Symposium on Human-Centric Computing Languages and Environments, Arlington, VA'. `ftp://ftp.cs.orst.edu/pub/burnett/hcc02.gridAssertions.pdf`.

Blackwood, J. (2002), 'Staroffice suite may be bitter pill for ms to swallow'. ZD-NET, CNET Networks, Inc. `http://techupdate.zdnet.com/techupdate/stories/main/0,14179,2865566,00.%html`.

Burnett, M., Sheretov, A., Ren, B. & Rothermel, G. (2002), Testing homogeneous spreadsheet grids with the "what you see is what you test" methodology, *in* 'IEEE Trans. Software Engineering'. 576-594. `ftp://ftp.cs.orst.edu/pub/burnett/TSE.gridTesting.preprint.pdf`.

Chadwick, D., Knight, B. & Rajalingham, K. (2002), 'Quality control in spreadsheets: A visual approach using color codings to reduce errors in formulae'. Information Integrity Research Centre, SCMS, University of Greenwich. `http://www.kamalasen.com/chadwick-00.pdf`.

Clermont, M., Hanin, C. & Mittermeir, R. (2002), A spreadsheet auditing tool evaluated in an industrial context, *in* 'EuSpRIG 2002 symposium'. `http://www.sysmod.com/eusprig02.htm`.

Gottfried, H. J. & Burnett, M. M. (1997), Graphical definitions: Making spreadsheets visual through direct manipulation and gestures, *in* 'Visual Languages', pp. 250–257. `http://citeseer.nj.nec.com/gottfried97graphical.html`.

Hibbard, B. (2002), 'Visad, java component library'. Space Science and Engineering Center - University of Wisconsin - Madison. `http://www.ssec.wisc.edu/~billh/visad.html`.

Igarashi, T., Mackinlay, J. D., Chang, B.-W. & Zellweger, P. (1998), Fluid visualization for spreadsheet structures, *in* 'Visual Languages', pp. 118–125. `http://citeseer.nj.nec.com/igarashi98fluid.html`.

Kay, A. (1984), *Computer Software*, Scientific American. 53-59.

Krazit, T. (2002), 'Staroffice set to challenge microsoft's office'. IDG News Service. `http://www.pcworld.com/news/article/0,aid,99643,00.asp`.

Liebowitz, S. (1999), *Rethinking the Network Economy*, American Management Association. Chapter 8 Major Markets-WordProcessors and Spreadsheets. `http://www.utdallas.edu/~liebowit/book/sheets/sheet.html`.

Myers, B. A. (1991), Graphical techniques in a spreadsheet for specifying user interfaces, *in* 'ACM CHI'91 Conference on Human Factors in Computing Systems', ACM Press, pp. 243–249.

Nardi, B. A. (1993), *A Small Matter of Programming: Perspectives on End User Computing*, MIT Press. `http://www.darrouzet-nardi.net/bonnie/ASmallMatter.html`.

Panko, R. R. (1997*a*), 'Human error website'. Honolulu, University of Hawaii. `http://www.cba.hawaii.edu/panko/papers/ss/humanerr.htm`.

Panko, R. R. (1997*b*), 'Spreadsheet research repository'. Honolulu, University of Hawaii. `http://panko.cba.hawaii.edu/ssr/`.

Panko, R. R. (1998), What we know about spreadsheet errors, *in* 'Journal of End User Computing.'. `http://panko.cba.hawaii.edu/ssr/Mypapers/whatknow.htm`.

Rothermel, K. J., Cook, C. R., Burnett, M. M., Schonfeld, J., Green, T. R. G. & Rothermel, G. (2000), WYSIWYT testing in the spreadsheet paradigm: an empirical evaluation, *in* 'International Conference on Software Engineering', pp. 230–239. `citeseer.nj.nec.com/rothermel99wysiwyt.html`.

Sajaniemi, J. (2000), 'Modeling spreadsheet audit: A rigorous approach to automatic visualization', *Journal of Visual Languages and Computing* **11**(1), 49–82. `citeseer.nj.nec.com/sajaniemi98modeling.html`.

Sun (2002), 'Java 3d(tm) api'. Sun Microsystems, Inc. `http://java.sun.com/products/java-media/3D/`.

Wilde, N. (1993), A WYSIWYC (What You See Is What You Compute) spreadsheet, *in* 'IEEE Symp. on Visual Languages, Bergen, Norway, Aug. 24-27, 1993, 72-76.'.

Yoder, A. G. & Cohn, D. L. (2002), 'Domain-specific and general-purpose aspects of spreadsheet languages'. Distributed Computing Research Lab, University of Notre Dame. `http://www-sal.cs.uiuc.edu/~kamin/dsl/papers/yoder.ps`.