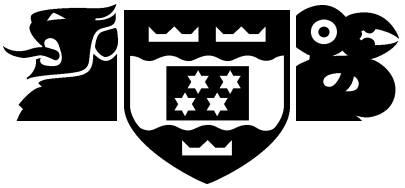
VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui



Computer Science

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@mcs.vuw.ac.nz

Visualising Spreadsheets

Daniel Ballinger

Supervisor: Robert Biddle and James Noble

Submitted in partial fulfilment of the requirements for
Bachelor of Science with Honours in Computer Science.

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. As a result there are limited facilities within the spreadsheet software user interfaces to create new visualisations, but the visualisation can also be supported externally. This report 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. Using corpus analysis techniques we can study aspects of how end-users program spreadsheets in practice.

Keywords: spreadsheets, visualisation, end user programming, corpus analysis

Contents

1 Introduction **2**
- 1.1 Motivation . . . 2
- 1.2 First Contact . . . 3
- 1.3 Approach to solving the problem . . . 3
- 1.4 End-user programming . . . 3

2 Background **5**
- 2.1 A Brief history of Spreadsheets . . . 5
  - 2.1.1 VisiCalc - A beginning . . . 5
  - 2.1.2 Evolution . . . 6
  - 2.1.3 Common components and operations . . . 7
  - 2.1.4 Spreadsheet Commands . . . 10
  - 2.1.5 Features of Interest . . . 10
- 2.2 Current support for visualisation . . . 13
- 2.3 Related Work . . . 14
  - 2.3.1 Spreadsheet Errors . . . 15
  - 2.3.2 Margaret Burnett's work on spreadsheet visualisation . . . 16
  - 2.3.3 Takeo Igarashi et al. work on fluid visualisations . . . 21
  - 2.3.4 Markus Clermont . . . 22
  - 2.3.5 Corpus Analysis . . . 23

3 The toolkit **27**
- 3.1 Gobbler — Searching for spreadsheets with Google . . . 27
  - 3.1.1 First Pass Design — Standard HTTP . . . 29
- 3.2 Refinement with the introduction of the Google web service . . . 30
  - 3.2.1 Corpus Sampling and Representativeness . . . 30
- 3.3 Fetcher — Downloading and storing files . . . 30
  - 3.3.1 Storage issues . . . 31
- 3.4 Extractor — Accessing the internals of spreadsheets . . . 31
  - 3.4.1 What needed to be extracted . . . 32
  - 3.4.2 File Format . . . 32
  - 3.4.3 Accessing the internal structure of a spreadsheet . . . 32
  - 3.4.4 Basic extraction options . . . 33
  - 3.4.5 Microsoft ODBC Excel and Sun JDBC-ODBC . . . 33
  - 3.4.6 XML . . . 33
  - 3.4.7 IBM alphaWorks Java Bean . . . 33
  - 3.4.8 JExcelAPI . . . 34
  - 3.4.9 Future opportunities . . . 34
- 3.5 Internal representation . . . 34
  - 3.5.1 Grammar . . . 35

3.6 Analyser — Calculation of corpus metrics . . . 38
3.7 Displaying the Information . . . 38
3.7.1 Available Visualisations . . . 38
3.7.2 Visualisation Tool Support . . . 39

4 Collection of visualisations **40**
4.1 Brief background on Information Visualisation . . . 40
4.2 The spreadsheet model . . . 41
4.2.1 Spreadsheet structure . . . 42
4.2.2 Spatial . . . 42
4.2.3 Logical . . . 43
4.2.4 Metrics . . . 43
4.3 Task model - A discovery process for the exploration of spreadsheets. . . . 45
4.4 Explorations - Areas of focus . . . 45
4.4.1 Real-estate diagrams - Single/Corpus . . . 45
4.4.2 Clustering - Single/Corpus . . . 50
4.4.3 Formulas at a Worksheet Level . . . 52
4.4.4 Precedent Tracing - Corpus/Single . . . 58
4.4.5 Dependency Trees . . . 60
4.4.6 Radar View - Corpus . . . 63
4.4.7 Future Visualisation Potential . . . 66

5 Conclusions **68**
5.1 Contributions . . . 68
5.1.1 Designed and implemented a toolkit for low level analysis and visualisation of spreadsheets . . . 68
5.1.2 Developed a principled model of spreadsheet structures . . . 68
5.1.3 Demonstrated sample spreadsheet visualisations . . . 68
5.1.4 Demonstrated sample corpus visualisations . . . 69
5.2 Related work . . . 69
5.3 Future Work . . . 70

List of Figures

2.1 The original VisiCalc in use 6
2.2 The trend towards Microsoft Excel market share dominance. Graph sourced from [32]. 7
2.3 Microsoft Excel running under Windows 8
2.4 Entering a basic formula in Excel with the aid of the mouse 9
2.5 Creating a histogram using the frequency function and an array of cells to perform bucketing. 11
2.6 Using Excel's Range Finder to examine a formula. 13
2.7 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. . . . 14
2.8 The Forms/3 visual programming language. The value in the Area cell is calculated using the value in the Abox cell. The red cell border indicates the cell has not been "tested". Image sourced from Margaret Burnett's guided tour of Forms/3 [12]. 17
2.9 Forms/3 visual support for testing. Here testing information is displayed after user validation. 18
2.10 The partial relation graph model for a clock spreadsheet 19
2.11 Static global view produced by Takeo Igarashi et al. to visualise the entire dataflow graph at once. 22

3.1 The general flow of information through the toolkit 28

4.1 Real-estate utilisation diagram in 2D 47
4.2 Real-estate utilisation diagram in 3D 48
4.3 The ThemeScape topographical map of word relationships between documents. 49
4.4 Clustering in a single worksheet 51
4.5 Clustering in a single worksheet 52
4.6 Excel's precedent trace auditing tool for sheet 1. Note that I have added by hand boxes to the right of each cell containing the relevant formula. 53
4.7 Formula extracted and displayed by the toolkit 54
4.8 Using the summation operation on both the columns and rows in a table. . . 55
4.9 Both Excel's and the toolkit's trace of simple absolute and relative inter-cell dependencies. 56
4.10 Jinsight's Reference Pattern View for exploring data structures and finding memory leaks. 57
4.11 Data dependency flow using the average unit vectors. 59
4.12 The flow of data between sheets in 3D. 59
4.13 A single dependency tree. 60
4.14 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. 61

4.15 The fisheye view of 4 dependency trees. 62
4.16 Radar and line graph views of bucket data for 259 spreadsheets from the corpus. The upper bucket count is cropped at 550. 63
4.17 The upper bucket count is cropped at 150 64
4.18 The upper bucket count is cropped at 40 64
4.19 Worksheet centre for a corpus of 259 workbooks. 65
4.20 Function utilisation in a corpus of 259 spreadsheets. 65

List of Tables

2.1 The cognitive dimensions devised by Thomas Green . [36](pg 9) 20

3.1 A BNF Grammar to describe Excel formula syntax 36
3.2 A BNF Grammar for parsing Excel formula components 37

Acknowledgments

I would like to thank Robert and James for their guidance and constructive criticism during the progression of this project. Special mention should also go to Robert again for coming up with the concept of this project. I would also like to thank my fellow graduate software engineering students: Dan, Rilla, Alex, Matt, Kirk, Pippin, Stuart, Angela, Mike, and Craig for the input they had into my research.

Chapter 1

Introduction

> Although end-user programming has received a growing amount of attention, there has been little research into aspects of end-user programming beyond the programming part per se. Programming is only one part of the development process, and focusing on other aspects is important for reliability of the programs end users create. In fact, reliability is an issue in end- user programming, as shown by statistics about spreadsheets, a widely used type of end-user programming language. [7](pg 1)

1.1 Motivation

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 analyses, and producing graphical displays. The versatility they provide allows the same programming interface to produce fiscal reports, help predict future stock market trends, record and analysis empirical data, and record student marks.

The notation used for such a powerful language is relatively easy to learn and use, most likely due to its similarities with the requirements and solution processes for the problems being modelled. The function and usage of the primitive operations is conceptually simple but they can be combined to perform reasonably complex tasks.

It is these similarities along with those that separate them from the majority of everyday programming paradigms that make the spreadsheet paradigm an interesting research topic.

Some of the properties that separate spreadsheets from most mainstream programming languages include that they are not compiled but rather updated in near real time as the user makes changes and interacts with individual worksheets. Also, spatial relationships replace time as the primary organising principle, and most sheets are constructed by end-users rather than trained programming professionals. Rather than immersing the programmer in low-level details of traditional programming, spreadsheets attempt to shield users from these details to instead free them to concentrate on solving the problem at hand.

The common theme between all uses of spreadsheets is that many larger problems result in the construction of monolithic programs that are hard to comprehend, even by those who programmed them. The logic of a spreadsheet model, with its many interrelated cells, can be arduous to follow, making it difficult to modify and debug a complex model. Comprehension issues are not just constrained to large, complex spreadsheets either. Relationships between

cells can span large regions of space or involve sizeable volumes of cells in a concise notation, adding to the cognitive load on the user.

1.2 First Contact

When first interacting with a spreadsheet the user can have 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 was commented on by Bonnie Nardi [38] where she mentions:

> 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.

1.3 Approach to solving the problem

A possible solution to this problem, and that investigated as part of this honours project, would be the creation of a set of images that could represent the contents at a more abstract level than possible with the spreadsheet software. The user would 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.

The spreadsheet programs themselves provide a limited ability to achieve such abstract views of the information they contain. This resulted in a design goal to have the ability to quickly implement new visualisations in a manner independent from the application they were created in and free from interaction with the applications developing company. 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.

1.4 End-user programming

In addition to providing new ways to view a spreadsheet there is also the aim of exposing the programming style and structure that occurs when domain experts rather than trained programmers code an application. So while one aspect of this research project involves examining individual spreadsheets there is also the objective of applying program visualisation techniques to a corpus of spreadsheets. The corpus will be used to do an empirical analysis of the dependency structures and general organisational layout patterns that typically occur in real-world problems. Of particular interest is how they model complex problems using the latitude granted by the spreadsheet model.

More specifically, the aim is to investigate styles and approaches used by people in programming visual spreadsheets. As Noam Chomsky observed about corpus linguistics [54], this is measuring user performance via naturally occurring data rather than actual user competence.

This can have interesting research benefits, as although users may be aware of advanced features like specialised functions or general rules of best practice, they may not apply them when dealing with real world problems. So although spreadsheets can grant users a flexible set of lower level operations, questions remain as to if they benefit from the full latitude granted by the spreadsheet language. It may be the case that users only ever use a small subset of the available operations in the majority of programming tasks. The remaining set of operations are then only rarely used for specialised problems. These claims are backed up in later chapters through the use of the corpus.

Spreadsheets are one of the most common forms of end-user programming currently in use, and Microsoft Excel is the most utilised example of a spreadsheet application. With a market share often topping more than 90%, the potential user base is vast. Even greater in magnitude is the number of spreadsheets that these users generate. This made Excel the obvious spreadsheet to concentrate research on.

A selection of all spreadsheets created will become part of a publicly accessible repository that is scattered around the Internet. This untapped resource offers the potential for research into the how end-users utilise the power and versatility spreadsheets provide.

Chapter 2

Background

Following their creation in the late seventies, spreadsheets rapidly developed into one of the most widely used software products during the 1980s and 1990s. Their design, which mimics that of an accountant's spreadsheet with of rows and columns, provides an end user programming environment that is accessible to a range of professions with different backgrounds and interests. This proven versatility has seen them used by accountants, managers, and a vast variety of other end users, as well as computer professionals.

This chapters serves to provide the reader with a background history of spreadsheets, what support they currently have for visualisation, the related research in the field, and then to finally introduce the concepts of corpus analysis.

2.1 A Brief history of Spreadsheets

Since its inception, the core spreadsheet paradigm hasn't evolved too far from its original founding concepts. Time has seen an evolution in the features present, and the performance has improved, but the basics of the paradigm, such as the grid layout and underlying formula language, remain remarkably similar. All spreadsheets can trace their roots back to a common ancestor, VisiCalc.

2.1.1 VisiCalc - A beginning

The spreadsheet began when Dan Bricklin and Bob Frankston created VisiCalc in 1979. The founding idea emanated from Bricklin wanting to create an effective way to utilise the computational power of microcomputers to solve small business school problems by "putting together the immediacy of word processing and the fluidity of the screen." [30] The core spreadsheet concept was patterned after a traditional blackboard production-planning layout. The resulting model used a set of cells arranged into rows and columns on a simple two-dimensional grid as an addressing mechanism for describing the spatial relationships between cells. This addressing system is constructed by identifying columns with letters and rows by positive integers. Individual cells are then referenced by a unique column and row pair. For example, E5 refers to the cell in column E of row 5.

Originally written in assembly language for a 32 KB Apple II, VisiCalc was a small spreadsheet with a terse single-line menu. Figure 2.1 is sourced from Dan Bricklin's website [11] and shows what the original interface looked like. The popularity and usefulness of this first spreadsheet led to the rapid development of numerous other spreadsheets, including early ports of VisiCalc to the Z80 and IBM PC (8086 or 8088 code).

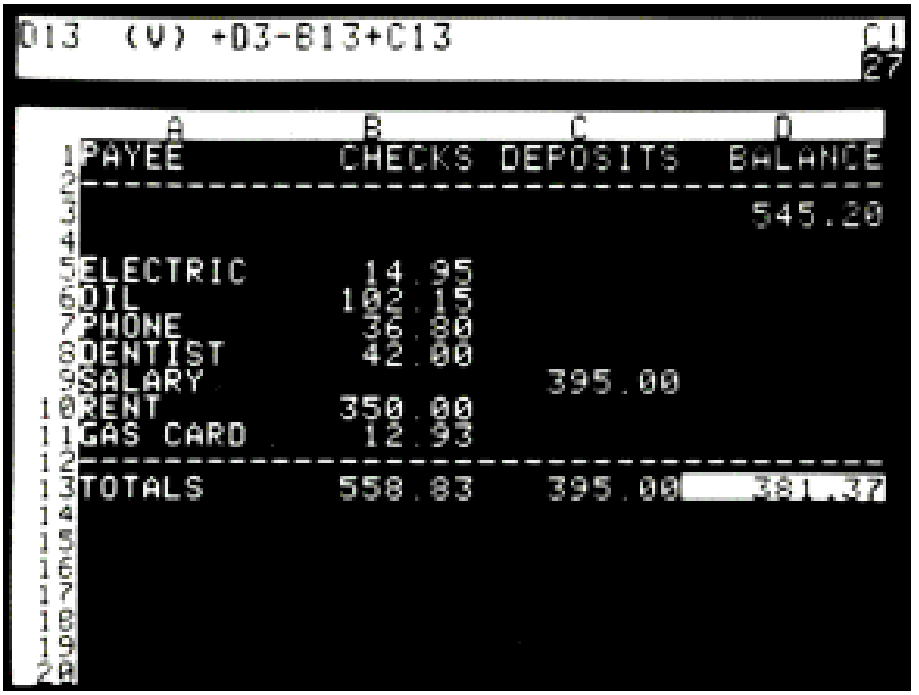


Figure 2.1: The original VisiCalc in use

2.1.2 Evolution

VisiCalc was soon replaced as the market leader in the increasingly competitive spreadsheet market. The first main rival to topple its dominance was SuperCalc in 1981. Developed for the Osborne computer, SuperCalc became the primary spreadsheet for 8-bit CP/M computers. SuperCalc was soon followed by Lotus 1-2-3 in 1983, which was created for 16-bit MS-DOS computers. The Lotus spreadsheet brought many new innovations and advanced features to the paradigm, including online help, sophisticated menus, graphic and database management capabilities, and macros. With its well thought out features it immediately became a best-selling software product for the time, and set standards for competitive spreadsheet products that followed [4].

With the spreadsheet user base growing and increased competition between vendors a race to add new features to meet varying user desires ensued. This led to the development of add- on features that have now been incorporated into the majority of spreadsheets.

One of the important phases in the expansion of the spreadsheet model was the addition of a third dimension, and in some cases a series of named grids, to allow for multiple-page spreadsheets to be bound into one larger workbook file. Some of the more advanced spreadsheet applications generalise this spatial representation by making it possible to perform inversion and rotation operations that can slice and project the data set in various ways.

More recently there has been the development of increasingly powerful spreadsheets with advanced features and presentation-quality graphics. It is seldom now that spreadsheets are created as standalone programs, instead forming fundamental components in larger integrated software suites that run under various operating systems.

One spreadsheet of particular note that attempted to introduce new concepts to the market was Lotus's Improv [56], which offered an innovative addressing mechanism as an alternative to the traditional A1 referencing notation. In a review of the release for Windows 3.1, Alan Zisman summarises the interface as follows:

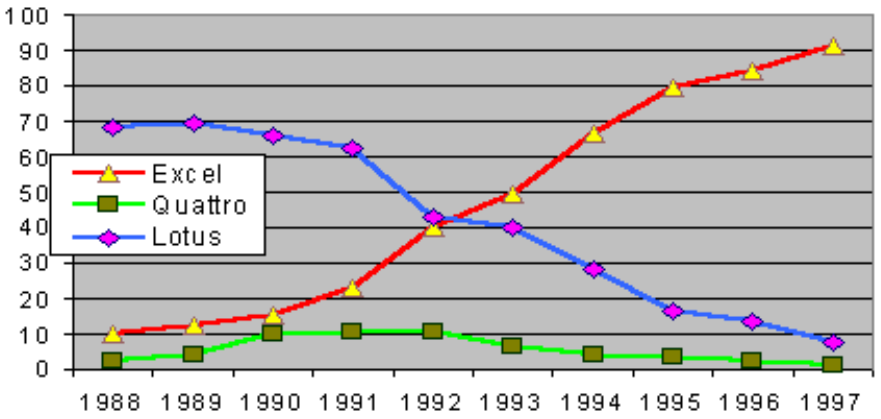


Figure 2.2: The trend towards Microsoft Excel market share dominance. Graph sourced from [32].

> Rows and columns become 'items' in Improv-speak. You start off with a single-cell worksheet, and two generic items. Give the items meaningful names: Income, Expenses, Profit, 1992, 1993, etc. Cells are referred to by the items that define them... 1992:Expenses, for example. [59]

Although considered to be a more flexible environment to perform spreadsheet style computations in, the differences were almost certainly its downfall, with the current market of users not willing to relearn controls to use the new interface.

In the early nineties the spreadsheet market was divided between three big players, Microsoft Excel, Lotus 1-2-3, and Quattro Pro [4], but more recently there as been a shift towards market dominance by a single product, Microsoft Excel. Figure 2.2 clearly shows Excel ousting 1-2-3 as the market leader in terms of revenue market share. According to data sourced from the analysts at Gartner Group, Microsoft currently enjoys a revenue market share of 93 percent in the category (office productivity software) [32, 9, 29].

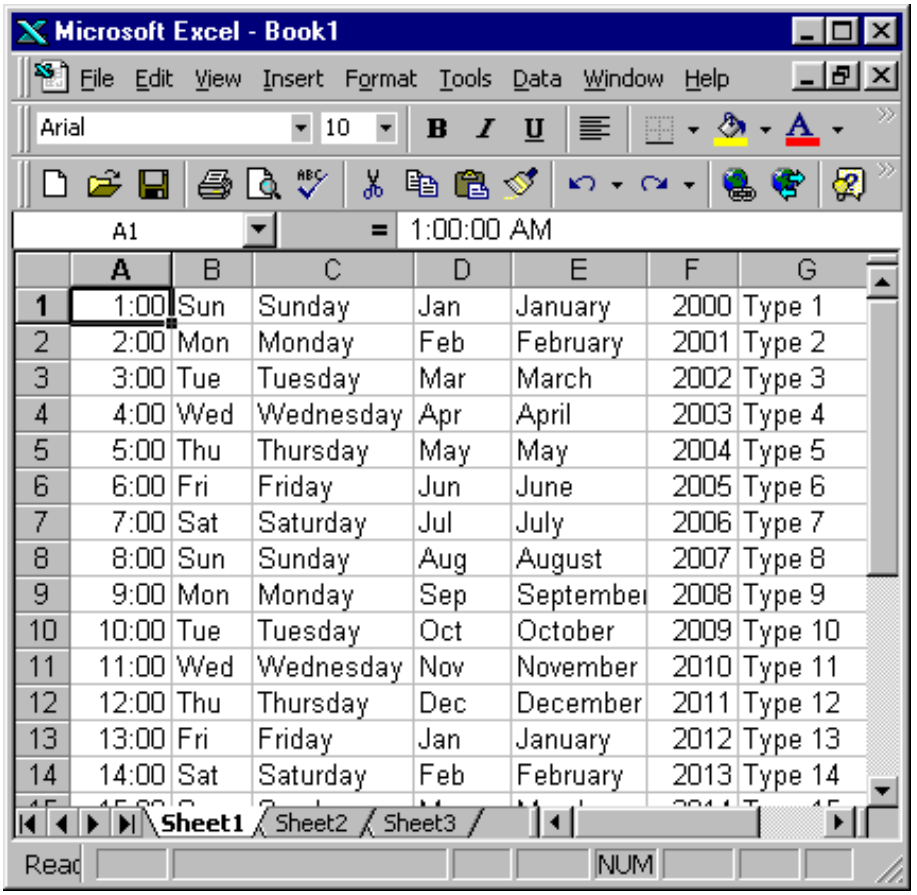
It is this strong dominance of market share that motivated our project to concentrate on studying Excel. The potential resources, both in the size of the user base and relevance with other research, were strong factors in the decision to concentrate on Excel. Figure 2.3 shows a typical user session with Excel.

2.1.3 Common components and operations

There are several important concepts that occur in the majority of spreadsheets on the market, and in particular Excel. Before further discussing the relevant areas of spreadsheet research it is important to have an understanding of them and their place in the underlying model.

Perhaps one of the most important mechanisms in a spreadsheet is the cell. Each cell has three core components: a value, an optional formula, and a coordinate defined by its position in the worksheet grid.

For any particular cell, a user can enter a primitive data element, such as a label (or string), a number, or a date. Alternatively, a cell's value may be imported from some other source file during the creation of the workbook, or be the result of computation on some other values at locations around the spreadsheet. Computations are achieved by defining a formula for the appropriate cell, which the interface allows for in much the same way as a value, except that an initial "=" symbol indicates that the following expression is a formula.



	A	B	C	D	E	F	G
1	1:00	Sun	Sunday	Jan	January	2000	Type 1
2	2:00	Mon	Monday	Feb	February	2001	Type 2
3	3:00	Tue	Tuesday	Mar	March	2002	Type 3
4	4:00	Wed	Wednesday	Apr	April	2003	Type 4
5	5:00	Thu	Thursday	May	May	2004	Type 5
6	6:00	Fri	Friday	Jun	June	2005	Type 6
7	7:00	Sat	Saturday	Jul	July	2006	Type 7
8	8:00	Sun	Sunday	Aug	August	2007	Type 8
9	9:00	Mon	Monday	Sep	Septembe	2008	Type 9
10	10:00	Tue	Tuesday	Oct	October	2009	Type 10
11	11:00	Wed	Wednesday	Nov	November	2010	Type 11
12	12:00	Thu	Thursday	Dec	December	2011	Type 12
13	13:00	Fri	Friday	Jan	January	2012	Type 13
14	14:00	Sat	Saturday	Feb	February	2013	Type 14

Figure 2.3: Microsoft Excel running under Windows

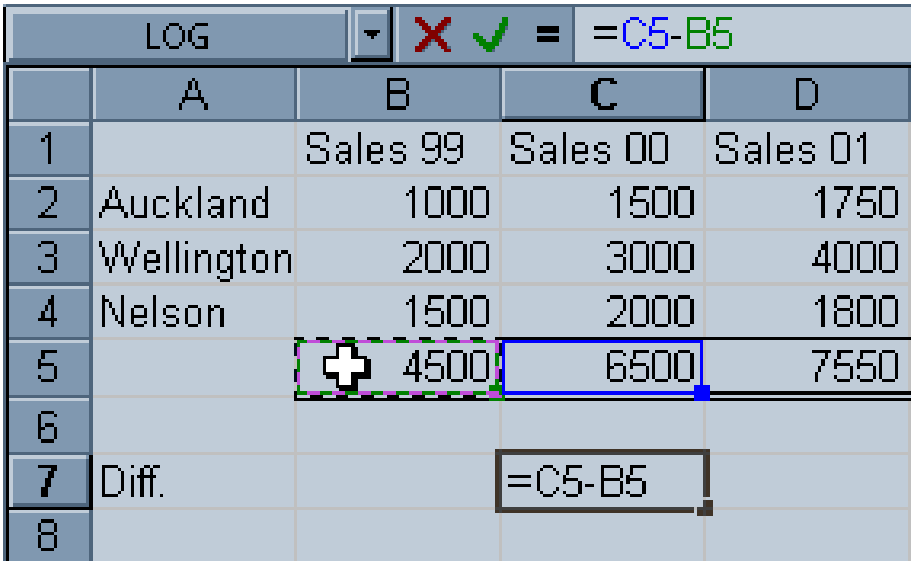


Figure 2.4: Entering a basic formula in Excel with the aid of the mouse

Although such a formula may be entered by typing in the symbols directly, a conceptually simpler approach for many users is to enter the formula's cell locations by "pointing to them", i,e. by clicking on the referenced cells while entering the formula as demonstrated in figure 2.4. The spreadsheet calculates the resulting value of a formula in a cell by using the current values of any cells that it refers to and any operators/functions it contains. It then stores the result as the cells new value that is displayed to the user on the screen. Generally, the calculation of a spreadsheet is performed in an order that first evaluates any of the cells referenced in another cell, although other options are available.

The unbounded referencing ability of formulas provides a linking mechanism between cells that permits the construction of modular and multidimensional models. The user is free to arrange data as they see fit, without considerations of accessibility between cells.

Notation for referencing cells is enhanced further due to the ability to reference a rectangular sub-grid through a range, allowing users to concisely refer to large blocks of cells to be used as operands in aggregate functions. Unlike traditional programming arrays, there is no limitation that prevents a cell belonging to more than one range.

Given this versatility it is often common practice to arrange the information in a logical fashion, particularly for a spreadsheet that is used repeatedly. Often such an arrangement will be related to the semantics of the sheet [35]. Markus Clermont observes that for a large sheet, business logic may dictate "arrangements where data-entry cells, cells immediately dependent on these data entries used for preparatory operations, and cells performing the final modelling or analysis are allotted to distinct, well identified, locations ... or laid out in a regular pattern." [35] These arrangements often put the values into logical sets with other values that conceptually play the same role, such as forming base data for use in a summation [34]. It is possible for a cell to belong to more than one conceptual set.

These layouts can provide cues about the underlying dataflow structure of the spreadsheet, such as in an example given by Igarashi where the summation formula "is typically placed at the end of the corresponding row or column." [25] Other benefits of a well organised spreadsheet include offering spatial clues for data location that enables users familiar with layout to quickly find data, even in very large spreadsheets [39](pg 6).

Absolute and relative referencing

One of the most popular features of the spreadsheet paradigm are the copy and fill (also known as replication) commands. David Reed accurately summarises the function of these commands as allowing the "construction of iterative calculations in a natural way by adjusting the formula cell references either to follow 'relatively' to the displacement of the formula or 'absolutely' to define a global parameter that affected all cells." [11] Excel, along with several other spreadsheets, uses the $ symbol to determine how a cell or range location identifier is interpreted by a copy command. Each location identifier can have a $ symbol associated with each dimensional component to indicate that it is to be treated as an absolute, or constant, reference. Conversely, without the $ symbol the reference is treated as relative. A cell reference such as C1 is unchanged in copying, while $A3 varies from row to row, but always comes from column A. When the relative cell reference, such as D3, is copied as part of a formula to the cell in the next row down it is changed to D4, a translation equal in distance to that of the copy command.

Copy and fill can also be applied to range references using the same mechanism, providing a consistency in the interface that David Reed observes as allowing for "very interesting and powerful recurrence relations to be represented naturally and obviously in the programming-by-example metaphor." [11].

The power of this addressing notation can have consequences, such as the ability to create a high degree of complexity within spreadsheets. This can cause difficulties, such as those observed by Clermont [35](pg 2) where spreadsheet users are generally not aware of this potential complexity. The result is that "mistakes, that have been made anywhere in the underlying model, will be propagated."

2.1.4 Spreadsheet Commands

Spreadsheets contain many additional features, including numerical, string, table, and logic library functions from mathematics, finance, engineering, statistics, and computing. There are also more sophisticated functions and commands for more advanced mathematics, including linear and multiple regression, matrix operations, random number generation, and linear programming.

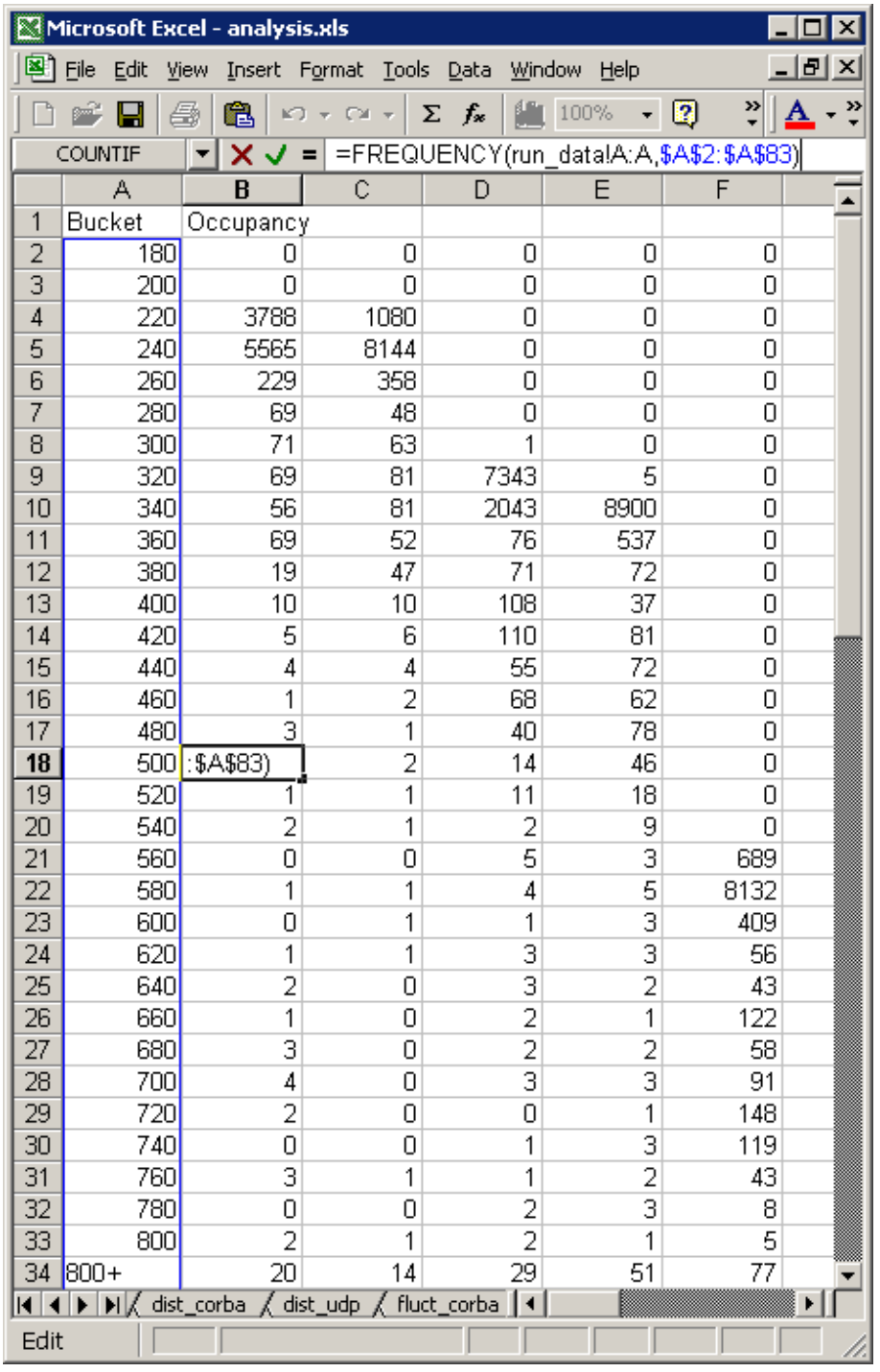
Several of these more advanced functions use the less well-known array syntax. One example of this array syntax is used is for separating ranges of data into buckets for producing histograms, as demonstrated in figure 2.5.

Along with many other commercial spreadsheets, Excel provides a powerful set of graphing commands to help users with data visualisations. The graphs available range from simple bargraphs to more complex surface maps. Of particular use with graphs is the dynamic update that allows users to observe changes in source data affecting the graphs.

2.1.5 Features of Interest

One of the most interesting characteristics of the spreadsheet paradigm is that even people with no traditional programming experience generally find spreadsheets to be intuitive, natural, and usable tools for business and mathematical modelling, decision making, simulation, and problem solving. While originally regarded simply as applications programs, below their relatively simple interfaces spreadsheets are in fact effective instruments for nonprocedural programming in general.

In a large portion of general computation programs imperative sequential control order is used to order the computational steps and provide the primary organisation mechanism. A well- defined entry point is used to determine the first instruction to be executed, with



	A	B	C	D	E	F
1	Bucket	Occupancy				
2	180	0	0	0	0	0
3	200	0	0	0	0	0
4	220	3788	1080	0	0	0
5	240	5565	8144	0	0	0
6	260	229	358	0	0	0
7	280	69	48	0	0	0
8	300	71	63	1	0	0
9	320	69	81	7343	5	0
10	340	56	81	2043	8900	0
11	360	69	52	76	537	0
12	380	19	47	71	72	0
13	400	10	10	108	37	0
14	420	5	6	110	81	0
15	440	4	4	55	72	0
16	460	1	2	68	62	0
17	480	3	1	40	78	0
18	500	:A83)	2	14	46	0
19	520	1	1	11	18	0
20	540	2	1	2	9	0
21	560	0	0	5	3	689
22	580	1	1	4	5	8132
23	600	0	1	1	3	409
24	620	1	1	3	3	56
25	640	2	0	3	2	43
26	660	1	0	2	1	122
27	680	3	0	2	2	58
28	700	4	0	3	3	91
29	720	2	0	0	1	148
30	740	0	0	1	3	119
31	760	3	1	1	2	43
32	780	0	0	2	3	8
33	800	2	1	2	1	5
34	800+	20	14	29	51	77

Figure 2.5: Creating a histogram using the frequency function and an array of cells to perform bucketing.

other instructions being reachable via a prior sequence of instructions. In contrast, the non-procedural spreadsheet environment replaces time with spatial relationships as the primary organising principle (at least from the perspective of the user)[57].

For performance reasons, the majority of programming languages based on time for organisation are compiled to a lower level representation, such as machine code or Java byte code. Spreadsheets instead offer direct real-time manipulation of an operating program, without the need for user interaction with an interpreter or compiler. From the users perspective a spreadsheet doesn't appear to run, but rather update to reflect changes in the model.

This typical spreadsheet interface is often referred to as having a WYSIWYG ("What You See Is What You Get") functionality via the ability for instant, and automatic, value recalculation based on formulas and the references they create between cells. This leads to another useful property of spreadsheets, the ability for rapid prototyping of programmes with the capability for trying "What if...?" scenarios. Exploratory programming allows users to experiment with the effects of a certain cell by varying its parameters or data and observing the immediate effects of these manipulations. A typical example of such a scenario is in financial modelling where it is possible to examine the compound effects of changes of such interrelated components as projected sales, prices, production costs, interest rates, and profit. Allowing the uses to "tinker" with the model in this way can aid in understanding and debugging complex data structures. When this ability is combined with undo features it is considered to satisfy Shneiderman's third principle of direct manipulation: "rapid incremental reversible operations whose effect on the object of interest is immediately visible" [49].

When first introduced, spreadsheets were used primarily in low-level decision-making, often by single individuals, to model smaller problems. Within a decade the use of spreadsheets expanded to become a valuable management tool that is now used extensively as a medium for implementing increasingly larger models [4]. These complex models are utilised for doing significant high-level business decision analysis where the resulting decision can have major consequences and repercussions. So with large spreadsheets being used increasingly for the analysis of critical decisions it is becoming crucial to ensure the correctness of the spreadsheet models, and current research in the field is reflecting this need.

When a user sits down to implement a model with a spreadsheet, they are typically concerned with two underlying problems, computation and presentation [39]. The model that a spreadsheet presents addresses both of these issues in an integrated way. The grid-based structure provides a presentation medium, storage for persistent data, and an addressing mechanism for information within formulas. The code that a user writes is dual purpose, being used both for performing the required calculations, and secondly displaying the results back in an intelligible form. This is one reason why formulas are traditionally transparent to users in the standard interface view. As cells are the only available storage mechanism for variables, a spreadsheet programmer must either store all information that is required for a formula in separate cells, or recalculate the value in every formula. This lack of temporary variables often creates a conflict between presentation and calculation requirements for a spreadsheet. A user who is conscious of needlessly recalculating values or formula complexity can find a solution by doing the calculation in a cell that is out of the visible screen area (scope) of the data to be presented. However, related information is no longer spatially grouped, which can make later debugging a more difficult task, especially for a user unfamiliar with the sheet. Takeo Igarashi et al. comment: "But in some cases, a user must put formulas referring to distant cells in irregular positions, which makes it difficult to understand the structure." [25]

As one would expect, a spreadsheet model will only contain those logical dependencies between cells that the programmer understood during implementation. A more complex task for this project could be to find those relationships that are only implicit in the model

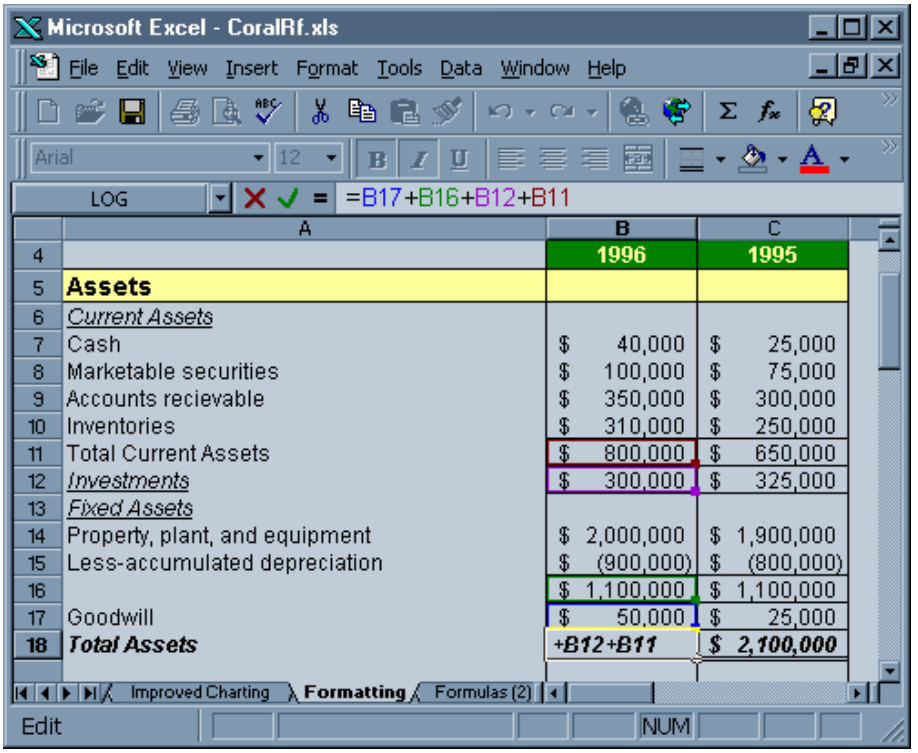


Figure 2.6: Using Excel's Range Finder to examine a formula.

and make them more apparent to the user. A good example of an implicit relationship is a constant value that is defined in multiple formulas around the spreadsheet. In many situations it may be more appropriate, disregarding the problem mentioned above, to define this value once and use absolute syntax to refer to it.

2.2 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 2.6. 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 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.

The second built in auditing tool has the ability to display interconnections between cells by tracing cell precedents and dependants using arrows. An example is given in firgue 2.7 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

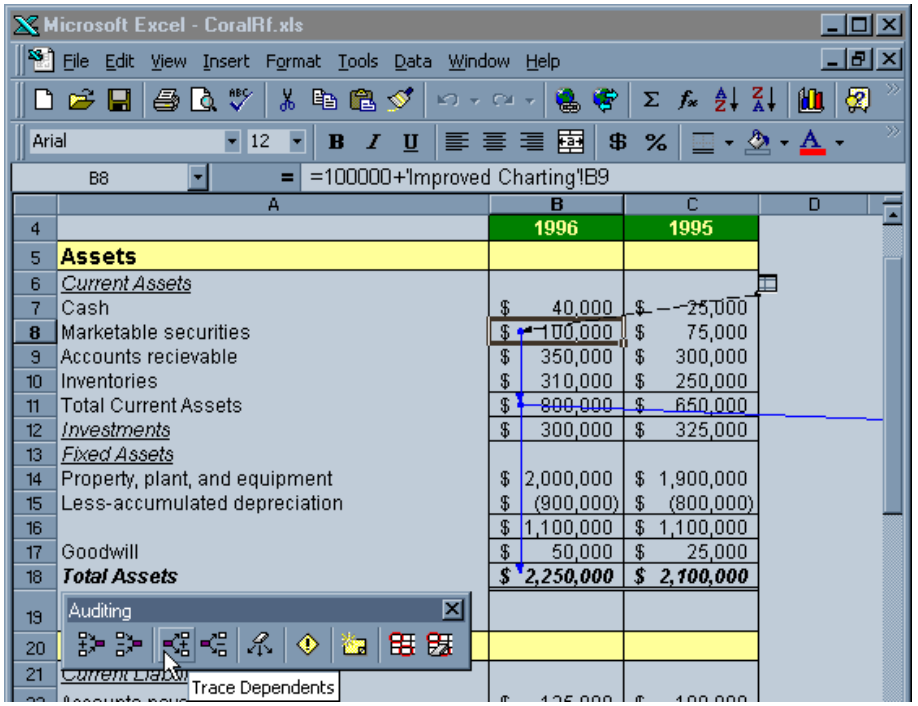


Figure 2.7: Excel's built-in auditing tools. Note how the reference to a cell is not fully visible and a sheet icon is the only indication of the link between sheets.

by Takeo Igarashi et al. [25] 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 [1, 15], 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.

2.3 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 [47]. Wilde's work on the WYSIWYC spreadsheet [58] aims to improve traditional spreadsheet programming by making cell formulas visible and by making the visible structure of the spreadsheet match its computational structure. Brad Myers created C32 [37], 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." [20] (pg 2)

Creating and applying specialised visualisations for visual programs has achieved less attention than many of the other fields of research. Work done that has been influential on our project includes James Noble's and Robert Biddle's visualisation of the visual programming language for the Nord modular synthesizer [41] and the later paper on visualising a corpus of the Nord modular patch language programs [40]. In the latter paper they make the following comments about the current state of visual program research and corpus analysis:

> Probably because most visual languages do not have a large user base, the practice of the visual languages community has been to adopt empirical usability evaluations to understand how languages are used, or to measure the effectiveness of individual small details of language designs [...], or researchers may participate in programming communities to evaluate their use of languages [...].
>
> Probably closest in spirit to our work is the empirical analysis of spreadsheet programs, where accountants or auditors work through a corpus to identify features of programs, such as cell error rates [...]. In the mainstream textual language community, analyses of programs are carried out mainly to improve implementations: analyses and critiques of programming style are generally based on single examples, drawing on literary criticism [...] or the patterns movement [...] for models.

Returning to spreadsheets, much of the emphasis in spreadsheet research at the moment appears to be the admirable goal of detecting and correcting errors. The research focus of the majority of published papers on spreadsheets all have errors as a common consideration. Proposed methods for locating errors range from detailed inspection of individual spreadsheets as part of an auditing process to automatic technologies.

2.3.1 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 [43].

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." [43]

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 [43]. 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 [42], 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" [42].

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 [42] (pg 14). More still may dismiss errors, because many syntactic errors are automatically detected and brought to the users attention.

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 [42] (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.

2.3.2 Margaret Burnett's work on 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, shown in figure 2.8, 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,

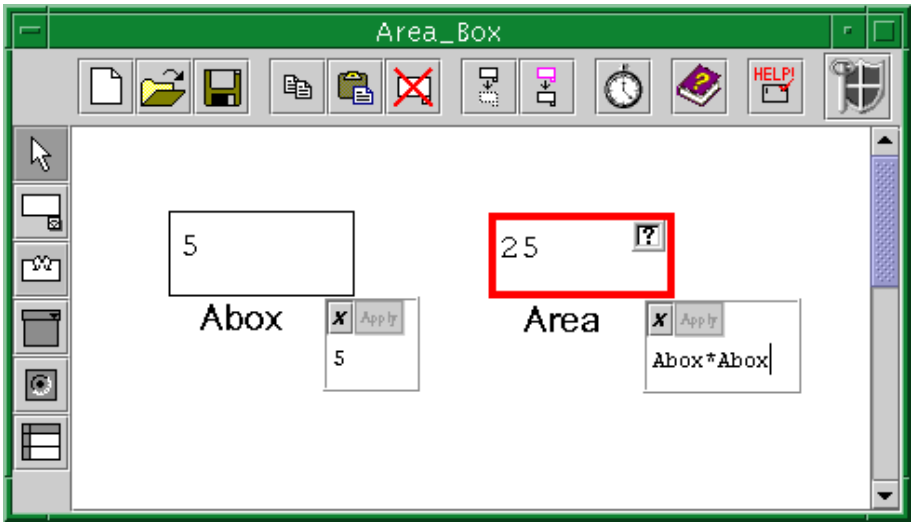


Figure 2.8: The Forms/3 visual programming language. The value in the Area cell is calculated using the value in the Abox cell. The red cell border indicates the cell has not been "tested". Image sourced from Margaret Burnett's guided tour of Forms/3 [12].

one-way constraint paradigm of spreadsheets, emphasizing that it should follow the value rule for spreadsheets" [20](pg 1). The definition she gives for the value rule is taken from Kay [26] 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 hypotheses 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. [7](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 [46]. An example of the Forms/3 support for testing is shown in figure 2.9. 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 subexpressions 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 [33](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

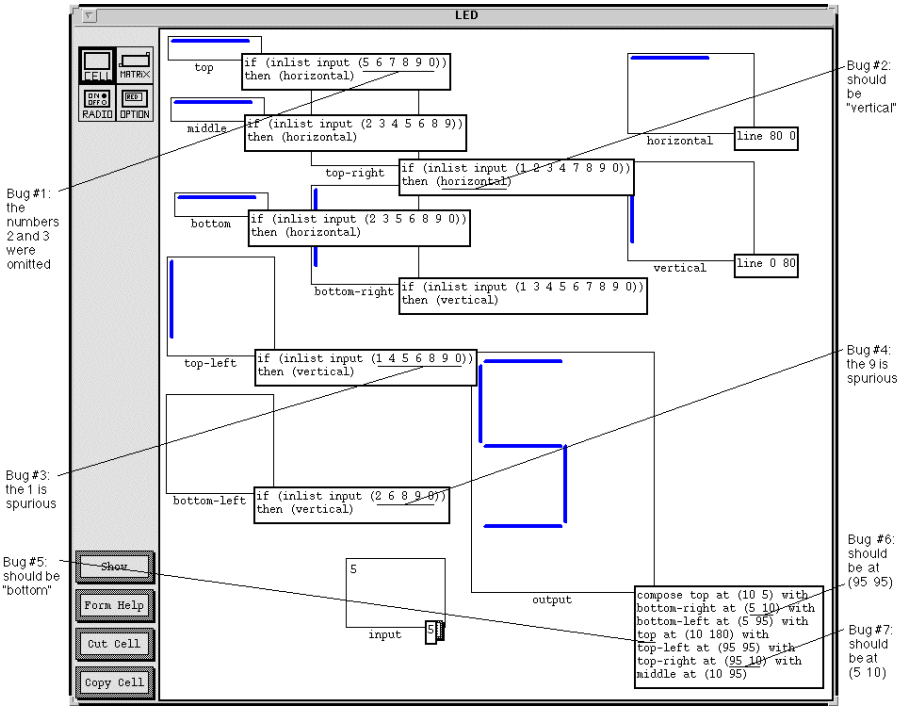


Figure 2.9: Forms/3 visual support for testing. Here testing information is displayed after user validation.

studies conducted on testing, debugging, and maintenance tasks with the help of WYSIWYT [7](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" [33] (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." [33]

A research effort relevant to our project was the creation of a cell relation graph in earlier testing work [33](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. Figure 2.10 show an example relation graph.

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 it) [33](pg 8).

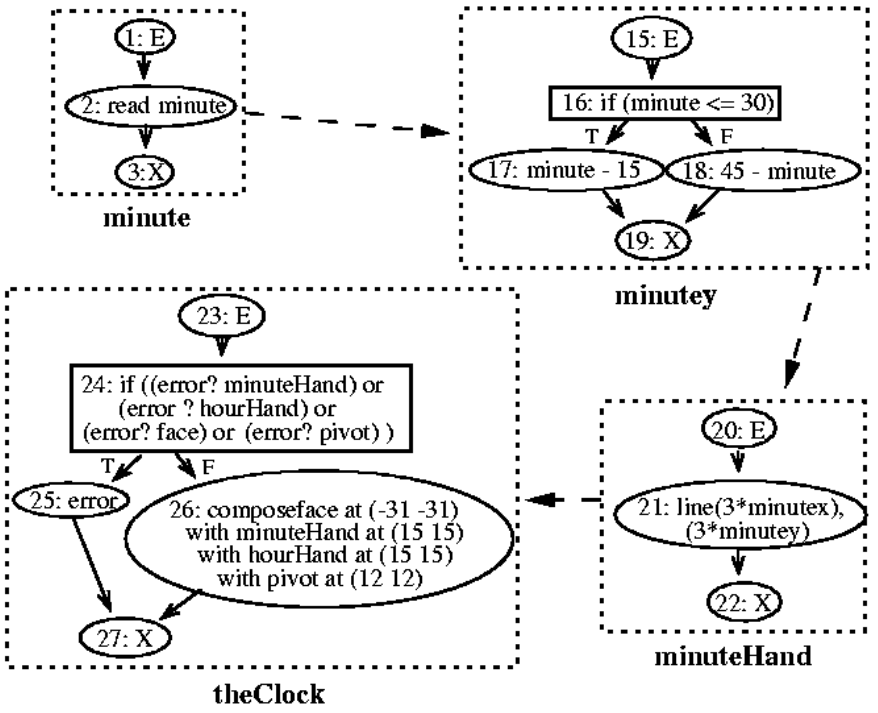


Figure 2.10: The partial relation graph model for a clock spreadsheet

Cognitive Dimensions

Visual programming languages have strong connections with cognitive theory as they attempt to improve a human's ability to program effectively, making it important to understand the cognitive issues that are relevant to programming. Burnett observes this importance with the aim of understanding the cognitive issues that are relevant to visual programmers to help them to use their full potential [36](pg 9). Her main research into this field is through the work of the psychologist Thomas Green and his colleagues on "cognitive dimensions" [22], which are "a set of terms describing the structure of a programming language's components as they relate to cognitive issues in programming". When assessing usability at design time, the cognitive dimensions framework does not provide a strict set of rules, but instead are used as heuristics and to provide vocabulary with which to talk about important design factors.

Table 2.1 reproduces a list of the dimensions, along with a thumb-nail description of each, that Burnett extracted from Green's work as part of a visual programming paper [36]. Each of these dimensions are related to a number of empirical studies and psychological principles detailed in Green's work [22], but it is also carefully pointed out that there are gaps in this body of underlying evidence. This is summarised by Green and the other authors as follows, "The framework of cognitive dimensions consists of a small number of terms which have been chosen to be easy for non-specialists to comprehend, while yet capturing a significant amount of the psychology and HCI of programming."

While several of these dimensions are not applicable to most of the images planned for this project, they are all significant as they communicate many of the important principles of gaps in Excel that I hope to address.

This work is by no means the only to address programming from the cognitive perspective. Other terms such as directness and immediate visual feedback have crossovers with Green's dimensions. Spreadsheet directness, in the context of direct manipulation, is described as "the

Abstraction gradient What are the minimum and maximum levels of abstraction? Can fragments be encapsulated?

Closeness of mapping What 'programming games' need to be learned?

Consistency When some of the language has been learnt, how much of the rest can be inferred?

Diffuseness How many symbols or graphic entities are required to express a meaning?

Error-proneness Does the design of the notation induce 'careless mistakes'?

Hard mental operations Are there places where the user needs to resort to fingers or penciled annotation to keep track of what's happening?

Hidden dependencies Is every dependency overtly indicated in both directions? Is the indication perceptual or only symbolic?

Premature commitment Do programmers have to make decisions before they have the information they need?

Progressive evaluation Can a partially-complete program be executed to obtain feedback on "How am I doing"?

Role-expressiveness Can the reader see how each component of a program relates to the whole?

Secondary notation Can programmers use layout, color, or other cues to convey extra meaning, above and beyond the 'official' semantics of the language?

Viscosity How much effort is required to perform a single change?

Visibility Is every part of the code simultaneously visible (assuming a large enough display), or is it at least possible to compare any two parts side-by-side at will? If the code is dispersed, is it at least possible to know in what order to read it?

Table 2.1: The cognitive dimensions devised by Thomas Green . [36](pg 9)

feeling that one is directly manipulating the object" [48]. From the cognitive perspective, Burnett and others describe directness in computing as having a "small distance between a goal and the actions required of the user to achieve the goal" [36](pg 3). The example Burnett gives of directness, given concreteness in a visual programming language, is allowing the "programmer to manipulate a specific object or value directly to specify semantics rather than describing these semantics textually."

Immediate Visual Feedback in visual programming refers to the language automatically displaying the effects of program edits. Visual languages often exhibit a degree of immediate visual feedback, which can be categorized using on of the four levels of liveness. Liveness is a term coined by Tanimoto to categorise the immediacy of semantic feedback that is automatically provided during the process of editing a program [52]. At level 1 there is no semantics implied to the computer, and at level 2 semantic feedback about a portion of a program is available, but it is not provided automatically. Level 3 is the point of interest as it is the level provided by most spreadsheets through automatic recalculation. At this level "incremental semantic feedback is automatically provided whenever the programmer performs an incremental program edit, and all affected onscreen values are automatically redisplayed" [36](pg 3). Using this technique it is safe to assume the consistency of the display state and system state, unless systems state changes can be triggered by events other than programmer editing. Level 4 address these additional triggers by including possible events such as system clock ticks and mouse clicks over time, "ensuring that all data on display accurately reflects the current state of the system as computations continue to evolve"[36](pg 3)

Liveness exhibited by spreadsheets strongly relies on the underlying dependency graph (or relation graph model). The importance of this structure is one reason we are exploring ways to visualise it as part of our project.

2.3.3 Takeo Igarashi et al. work on fluid visualisations

Takeo Igarashi et al. describe spreadsheets as augmenting "a visible tabular layout with invisible formulas" [25]. 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 [25] 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 2.11.

The invisible formulas can affect the 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." [25]

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

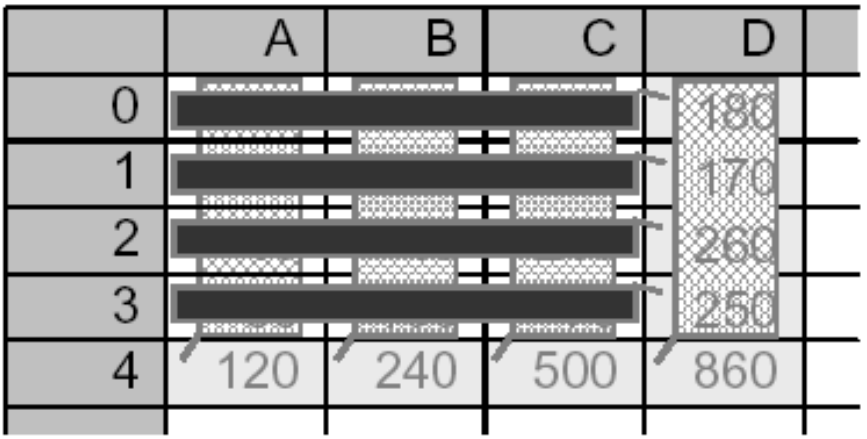


Figure 2.11: Static global view produced by Takeo Igarashi et al. to visualise the entire dataflow graph at once.

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 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. [25]

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.

2.3.4 Markus Clermont

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 [35, 34].

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!" [34]. 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." [35](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 [34]. 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 [35]. As they have presented this system, when comparing two formulas they find either no-equivalence or:

- Copy-Equivalence, which exists if "the formula 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 formula 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 this project when looking for patterns through visualisation. Different types of equivalence can be used to conceptually group sets of formula 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 aggregate 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" [35]. 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.

2.3.5 Corpus Analysis

Defined in the loosest sense a corpus is any body of text, more commonly it is a body of machine-readable texts, and in the strictest sense it is a finite collection of machine-readable text, sampled to be maximally representative of a language or variety [54]. The important point is that the texts should be naturally occurring and that, as a whole, the corpus should "characterise a state or variety of a language" [10].

Corpus analysis has long been the domain of linguists, who utilise a corpus as a starting point for "linguistic description or as a means of verifying hypotheses about a language." [10] The methodology they use for this process is often referred to as corpus linguistics [27].

When linguists undertake research they are often not interested in an individual text or author, but a whole variety of language [54]. In such cases there are two options for data collection:

- They could analyse every single utterance in that variety. However, in all but a few cases this option is impracticable. As analysing every utterance would be an unending and impossible task, due both to the share volumes of text and the speed at which they could process them. An example of an exceptional case would be a dead language that only has a few known texts in existence.

- They could construct a smaller sample of that variety. This is a more plausible and realistic option.

It is the latter option that corpus analysis concentrates on. However, the use of a sample rather than the entire population impacts both the collection methods used and how the results are interpreted.

One of the most influential people in the field of linguistics to comment on the use of corpora was Noam Chomsky. In a remarkably short space of time (the late fifties and early sixties) he changed the direction of linguistics away from empiricism and towards rationalism. Only in more recent times has mainstream attention returned to corpus analysis as a source of empirical data for use in research, but not without first re-evaluating and modifying the practices in light of his observations. An empiricist approach is dominated by the observation of naturally occurring data, typically through the medium of the corpus. An alternative to empiricist analysis suggested by Chomsky is to use a rationalist theory, which is a theory based on artificial behavioural data, and conscious introspective judgements.

Chomsky's main criticism of the corpus approach for linguistics was that language is infinite and therefore, a corpus would be skewed. Tony McEnery and Andrew Wilson summarise this skew as follows:

> In other words, some utterances would be excluded because they are rare, others which are much more common might be excluded by chance, and alternatively, extremely rare utterances might also be included several times.[54]

They follow these comments with the observation that while modern computers allow the collection of corpora that are much larger than Chomsky had envisioned, his criticisms still must be taken seriously. Doing this does not imply that corpus analysis should be abandoned completely, but instead that it is important to establish techniques in which a much less biased and representative corpus may be constructed. In linguistics this is achieved by using a broad range of authors and genres which, when taken together, can be considered to "average out" and provide a reasonably accurate picture of the entire population of interest. Naturally, these same implications will apply to a corpus constructed for research into the spreadsheet population.

There are other issues that can bias the sampling technique used for constructing the corpus. One of these is the size and content of the corpus with respect to time. The two general approaches are to either have static information or allow the corpus to grow dynamically over time. The type of a corpus can be classified as being either static in size and content or growing dynamically with time. Those that do grow are referred to as longitudinal and involve a large sample study. The changes that occur over time affect the basis of the sampling technique in a longitudinal study.

In the same way that it is infeasible to study every existing literary document, studying the entire population of spreadsheets is also infeasible. Much of the research into corpus linguistics is applicable to this project, but there are fundamental differences in the source data used to construct the corpus.

There are strong grammatical rules and dictionaries in language and part of what is done in corpus linguistics involves looking for possibly new rules or words and proof of usage for existing rules and words. A spreadsheet, being a computational entity is (in theory) strongly defined and constrained by the environment it is programmed in. That is, the programmers decide what the language can and cannot do. However, almost all software has bugs, undocumented features, backdoors and hacks (especially for larger programs like Excel) that allow them to do things not envisaged at the time they were designed and programmed. The conclusion from these properties is that language has a more flexible form of evolution than software. People are free to invent new words or change the meaning of older words, where as programs usually require structured and planned change. Changes in software will also mainly occur in bursts as new versions (and patches) are released.

With spreadsheets being constructed from a set of well-defined syntactic rules it would be plausible to attempt to generate every possible spreadsheet in existence. However, studying every possible construction of a spreadsheet wouldn't tell you anything about how the language is used in practice. Rather, it would only show you the full scope of its abilities. The important concept here is that while it is in theory possible to generate spreadsheets from the well-defined rules, the share magnitude of possible spreadsheets would make the task implausible.

With corpus linguistics, the fact that the language is infinitely extendible effects the construction of the corpus. For the purposes of our project there are a sufficient number of possible spreadsheets that the population could be considered infinite. That is, it would not be possible to generate all the possible spreadsheets using the syntax rules and a systematic program. An example that serves to justify this conclusion involves considering all the possible positions for a single occupied cell. Given that there are 256 columns and 65536 rows, there are almost 17 million possible locations for the single occupied cell.

Corpus analysis, usability testing and interviews

The motivation for using a corpus approach is summarised by James Noble and Robert Biddle in considering the alternative methods of gathering information:

> Because they are time consuming, usability evaluations and participant observation are generally limited to tens of subjects working on tens of programs. Surveys can provide information from many more subjects, but surveys cannot engage with actual programs, only programmers' opinions and beliefs about their programs [...]. [40]

The main benefit of corpus analysis in comparison to usability evaluation or participant observation is that while it requires a sample of several hundred or more programs, it does not require a detailed analysis of how each of these programs were written. James Noble and Robert Biddle observe that this makes the corpus analysis approach "best suited for investigating the parole [...] of a language — the way it is used in practice — while other techniques can provide more specific information about the design of languages themselves." [40]

When acquiring information directly from a test subject there is a risk that the very act of testing itself could bias the results. When a user is being observed or is aware that their work will later be analysed, they can alter their normal behaviour and work practices.

For example, a user may be aware that a better function exists to perform a computation, but in normal every day work would perform the same computation using several smaller functions, simply because it was easier. Under examination the user may opt to use the more complex function in an attempt to appear more competent with the language. A similar problem applies to interviews, where the user may say one thing and then practice something completely different. This is the important point about a corpus based approach, as the use of naturally occurring data that is representative of the population. That's not to say that the data derived from usability testing or interviews in not valuable, just that it can be open to bias.

One possible research technique would be to combine all the above processes into a more comprehensive and effective methodology. Firstly, corpus analysis would be applied to indicate characteristics of interest that are worth further investigation. The information derived from the corpus could be used to devise and then perform more targeted in depth research via usability testing and interviews.

A summary of the arguments against the uses of a corpus by XYZ serves as a good reminder for the implications of using one:

> First, the corpus encourages us to model the wrong thing - we try to model performance rather than competence. Chomsky argued that the goals of linguistics are not the enumeration and description of performance phenomena, but rather they are introspection and explanation of linguistic competence. Second, even if we accept enumeration and description as a goal for linguistics, it seems an unattainable one, as natural languages are not finite. As a consequence, the enumeration of sentences can never possibly yield as adequate description of language. How can a partial corpus be the sole explicandum of an infinite language? Finally, we must not eschew introspection entirely. If we do, detecting ungrammatical structures and ambiguous structures becomes difficult, and indeed may be impossible. [54]

Chapter 3

The toolkit

A large portion of this project involved the implementation of 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 purpose of collecting spreadsheets from around the Internet is to allow the rapid collection of large number spreadsheets that can then be used to perform corpus analysis. To reduce the amount of code production required, many of these applications make use of libraries and services from third parties. Discounting the code from outside sources, the toolkit is mainly composed of 20,000 lines of Java code distributed across 6 main 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.

Figure 3.1 shows the general flow of information between the packages. The process starts with user-supplied keywords that are submitted to Google. The resulting URLs are stored and then feed through to Fetcher, which downloads and stores all the Excel files. The next phase involves file format conversion using the Extractor package that outputs each Excel file in the toolkit's internal representation. Following this translation, Analyser iterates over the corpus of files collecting and aggregating the information of interest. Once completed, the data is passed to the visualisation tools for presentation using various techniques.

3.1 Gobbler — Searching for spreadsheets with Google

Collecting the large volumes of spreadsheets to populate the corpus required automated tools to firstly find the location of the spreadsheets on the Internet and secondly to download and store them persistently.

The Gobbler application was created to satisfy the former part of this acquisition phase. It provides automated corpus identification via keywords and standard Internet search engines. The main focus is the formation of a collection of URLs for spreadsheets located around the Internet with public accessibility. Google [18] was chosen for the search engine due to its ability to narrow results to Excel files by simply appending the argument "filetype:xls" to the search terms. Also, Google indexes hundreds of thousands of Excel spreadsheets, giving it a sufficient magnitude for a reasonably sizeable corpus without the need to build a separate crawler.

Search keywords were generally chosen to acquire spreadsheets that have content in common or with interesting attributes, such as particular functions or dependency structures. Common examples of keywords used include "student marks", "fiscal year", and "profit and loss". An unanticipated benefit of this search technique is that other Google features can be used to further refine searches. An example of such a refinement that was used on several

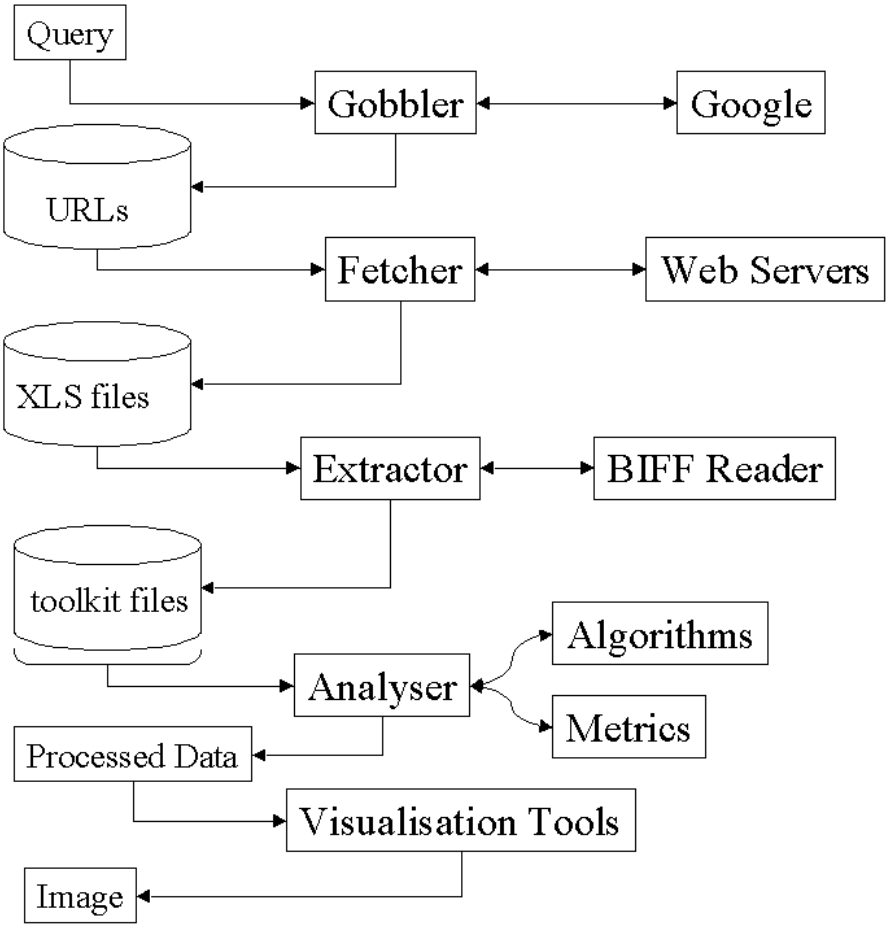


Figure 3.1: The general flow of information through the toolkit

occasions is the ability to target the search for spreadsheets from individual domains, such as just within New Zealand or within a particular organisation.

3.1.1 First Pass Design — Standard HTTP

The main chain of events undertaken to complete the Gobbler process are modelled on the sequences that would be followed if it were undertaken manually. The search terms and filetype:xls argument are passed to Google. The URLs are extracted from the resulting page and appended to any previous results. The process continues until the target number of results is reached or Google reports that no more results are available. The last step undertaken by the Gobbler application is to then store all the resulting URLs in a file with the search terms used as the filename.

The process of sending a page request to Google initially involved submitting a query via the standard HTTP 1.0 GET syntax [53], which is similar to how a standard Internet browser would send search term data from a webpage form to Google by encoding it in the URL. With this particular technique it is possible to speed up the search process by appending "&num=100" to the search terms to request 100 results with each query. The resulting page is then read back and parsed using regular expressions to remove all the URLs from the page. Prior to the first parse for any search the estimated number of results is extracted and used to modify the target number of URLs if necessary. Interestingly enough, during the coding of a parser it was revealed that Google doesn't use standard valid HTML syntax for URLs in the link tags as they are not enclosed in parenthesises, i.e. `*<a`href=link¿label¡/a¿* instead of `*<a`href="link"¿label¡/a¿*.

A GNU regular expression package [13] is used to provide the regular expression support for Java rather than the regular expression package of Sun's Java version 1.4, mainly as version 1.4 was not currently available on the university's Unix computing environment. Filtering was required on the parsed URLs as many of them were to resources other than the files of interest, such as links to cached pages, other Google services, and advertising. In most cases this was simply a case of matching known strings against the URLs, such as "/search?q=cache:" to remove any cache links.

The storage of results was a relatively trivial task of writing to a file using the search terms as the filename. In the case of an existing file with the same name, the results are appended to those from prior searches with duplicates removed prior to downloading.

Problems

Using standard HTTP requests to the Google server proved effective for most of the toolkit's searching requirements, but a few issues detracted from its overall effectiveness.

Firstly, this technique may be in violation of the terms of usage for Google as it could be defined as "sending automated queries" or performing meta-searching, a process where several search engines are queried through one interface. We initially thought this primarily addressed using Google's data by other search engine providers, but then realised it might also relate to our work.

Secondly, for any particular search, the maximum number of results returned is capped at 1000. This is generally not a major issue but it does create an upper bound for the number of URLs available for any keyword in the corpus if this is the only search technique available.

3.2 Refinement with the introduction of the Google web service

In mid-April this year Google released their beta web API (Application Programmer's Interface) [19] that communicates with their web service search engine via SOAP (previously an acronym for the "Simple Object Access Protocol" [55]), an XML-based mechanism for exchanging typed information. This allows standard query access to over 2 billion Web pages that Google indexes. With some modification to the HTTP version of Gobbler, a second search method was added supporting this more legitimate query method. In addition to the benefits of conforming to Google's terms of usage, the API also presents the URLs in a form that can be directly appended to the result list without parsing, and hence slightly improves performance.

Perhaps one of the greatest benefits of this search method over using standard HTTP GET is in addressing the danger of overusing Google, as documented by Andreas Eberhart [16] while creating a similar tool to collect RDF documents. He mentions, "that running the automated query repeatedly might cause the requesting IP to be banned by Google." This could be potentially disastrous for the University, if not for myself when the cause was traced. The exact query volumes required to trigger blocking are not publicly documented, so all searching was generally limited per day as to become insignificant with respect to normal university traffic. This can be roughly gauged by examining the web-cache logs that indicate in a busy week there can be between forty and fifty thousand accesses to Google (and sometimes up to two hundred megabytes transferred). The API also enforces a limit of up to 100 queries per day, each returning a maximum of 10 results, through the use of a unique identification key. This was considered to be sufficient for the requirements of this honours project.

To make the Google Web API's Java interface compatible with the university computing environment it needed to have authentication ability with the schools web proxy added. This involved using a patch by Patrick Chanezon [14], as Google's beta release software did not have support for this.

3.2.1 Corpus Sampling and Representativeness

An important property of a corpus is that the sample it contains be representative of the population. The use of Internet search engines greatly accelerates the collection process but also skews the corpus. The causes of this skew vary from the keywords chosen when searching for the corpus, to the nature of spreadsheets available on the Internet.

Those spreadsheets that are publicly available on the Internet exhibit characteristics that are not representative of the population. One reasons for this difference is that many spreadsheets are altered prior to being made publicly available, such as spreadsheets containing student marks. This can include the removal or sensitive source data and formula. Another form of spreadsheets in the population that will never see the Internet are the "sketchpads" [42], those spreadsheets that only existing for short periods of time to address a small problem.

3.3 Fetcher — Downloading and storing files

The Fetcher application provides the relatively simple but important second phase task of downloading and storing the corpus from the lists of URLs created by Gobbler. Content is read from the Internet through a standard Java URLConnection as an array of bytes. Once the files are acquired they are stored in a directory structure derived from the keywords used to locate them so that later analysis can easily be constrained to particular search terms.

While downloading Excel files it was quite common to find extremely slow web servers. To help address this issue and improve performance when a high bandwidth Internet connection is present, a multithreaded design was used. When combined with a multiprocessor computer the performance improvements were considerable.

As with the Gobbler application, one of the biggest hurdles for this application was traversing the schools web proxy from within the Java runtime. Exact documentation on what was required was sparse, with several sources providing conflicting information. When resolved, the process involved setting Java environment properties and base64 encoding usernames and passwords that were to be submitted to the proxy when requested.

To provide reasonable performance for batch processing jobs, the user name and password are encrypted to disk so the program can automatically retrieve this data. This later caused issues with the Sun cryptography libraries when trying to bootstrap the Java classpath to modify the URLConnection classes with timeouts.

Maintaining a complete list of source files and their locations is achieved by never removing URLs from Gobblers output files (with the exception of duplicates). This introduces the possibility of downloading a file twice and wasting Internet connection bandwidth. The issue is addressed by comparing the HTTP Content-Length header field with the size of any file stored on disk having the correct path and name. If the length differs, the file is downloaded again.

To prevent a broken link overwriting an existing file with a sever generated HTML 404 message, the MIME type is first checked to indicate Excel Data Content (application/vnd.ms-excel, application/x-msexcel, or application/ms-excel). This technique proved effective over a dialup connection but there may be an issue when using a web-cache that attempts to finish downloading the file before the application reads the data, effectively defeating one reason for the check.

3.3.1 Storage issues

Due to the large volume of files that were collected to create the corpus it was important to make special consideration of how to store them. Asides from the obvious requirement for persistent storage there was also the need to be able to perform an iterative retrieval of records that may have been compressed using standard compression technologies. As mentioned below, not only is the source Excel file stored but also the toolkit's internal representation.

The simplest solution, and that current employed, is to store all the files in the standard file system. As the 8000 spreadsheet occupy around 2 gigabytes of disk space it has not been necessary to compress the files, although zip compression has been added to the toolkit to address the issue if and when the corpus grows. Tests on Excel files between 100 and 500 kilobytes show compression levels of between 50 and 75%.

In addition to compression another future option would be store all the corpus data in a database. This would allow for iterating over spreadsheets with selection based on more advanced queries.

3.4 Extractor — Accessing the internals of spreadsheets

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 aim is to present information from the corpora in a format that can be easily processed and analysed by the rest of the toolkit.

3.4.1 What needed to be extracted

The artefacts required from each Excel file are derived from the low-level structure of the spreadsheets. The basic unit of interest is the cell, the atomic building block of the spreadsheet model. Each occupied cell in the spreadsheet had to be iterated over to extract all the data in a spreadsheet workbook. Occupied cells have a value that is of some type, such as text or a number. Cells that are to perform a calculation and assign the result to their value will also contain a formula. The formula extracted from the binary file requires conversion from the stored Reverse-Polish-Notation (RPN) format to the standard formula notation presented to the user when interacting directly with Excel. Ideally this process will be an accurate conversion, and not missing any important elements such as absolute reference identifiers. This requirement stems from the need to later construct the full set of dependencies between cells using the toolkit's formula parser, which understands standard Excel formula notation, in order to create an internal representation of all the spreadsheets elements. Accordingly extraction converts formulas from RPN to standard formula notation.

All cells in the spreadsheet have positioning data that must be maintained for later computation. This information includes the worksheet the cell exists in, its column and row coordinates, and any incoming or outgoing references created by formulas. Additional data is stored for each worksheet, such as the sheet's name and position in the stack of sheets, that is of interest for use in later analysis.

All information extracted from the spreadsheets is converted into the toolkit's internal representation, which results in a strong degree of integration between the components of the toolkit, partially for efficiency reasons and also to enable independent development of the components.

3.4.2 File Format

Excel files are formatted and stored using one of several versions of the BIFF (Binary Interchange File Format), depending on the version of Excel used to save the file. There has already been significant work on documenting this file format (outside Microsoft), including that done by Daniel Rentz as part of the OpenOffice project [45].

Developing an application to process the raw binary of an Excel file from scratch is a complex project in itself and the time required too undertake such a design and programming task was not available within the constraints of this project. Instead, direct interaction with the underlying file format is achieved through the use of third party APIs.

3.4.3 Accessing the internal structure of a spreadsheet

Available packages for reading and converting the Microsoft Excel file format range between being commercial, freeware and open source. The API needed to be freeware or opensource, as the costs of commercial packages for reading Excel are beyond the resources available for this project.

To keep the programming consistent, the target format had to be readable by Java, and preferably the package would be operated via a Java interface. In the ideal case the process would be pure Java, implying platform independence and the ability to migrate comfortably with the rest of the toolkit.

As no individual third party Excel file reader proved to have all the desired attributes, the internal structure of the Extractor application was designed to allow for different reading methods to be utilised. A wrapper design provides transparency through a common interface to the rest of the toolkit about which reading method was being used. Hence, different methods can be used depending on the required characteristics.

3.4.4 Basic extraction options

When looking for a suitable method of converting BIFF encoded files to a representation using Java object structures, several technologies and processes were investigated. Each of these brought certain benefits, implications, and limitations. All techniques were assessed on the ability to extract the required information, minimal user interaction, platform independence and fault tolerance. Fault tolerance becomes an issue as some extraction methods encounter difficulty with certain Excel structures and it is important that they handle them in a manner that allows the toolkit to continue functioning with minimal missing information.

3.4.5 Microsoft ODBC Excel and Sun JDBC-ODBC

The article "It's Excel-lent" [50] by Tony Sintes on the JavaWorld website showed how a single Excel file could be treated as a table in a database using Microsoft's Open Database Connectivity (ODBC). ODBC treats the worksheet name as the database name and the first row in a spreadsheet as the column names. Java Database Connectivity (JDBC) then allows information to be imported into the Java runtime environment. This method requires a large degree of user interaction and an alternative way to migrate the column names into the extraction process. Other drawbacks include the lack of support for formulas and, through the use of a Microsoft specific Excel ODBC driver, binding any implementation to the Windows platform.

3.4.6 XML

In many aspects, XML provides an attractive intermediate encoding format by providing a universally understood common denominator format for Excel data. The spreadsheet could be saved as XML and read into Java via an XML interface (possibly using JBDC). This has many potential advantages, including being a non-proprietary, language-independent, encoding format that is also human readable.

Unfortunately, few techniques showed promise in this area as they would often need commercial software to perform the conversion, require a high degree of user interaction to manually open and then save the Excel file as XML, or lack support for formulas.

Another issue with XML is the size of the encoded files caused by the trade-off between universality and performance in terms of speed. The ASCII encoding and XML tags all affect performance by increasing the size of files and requiring more system resources to parse the files. In a comparison between serialised Java binary and XML encoded with UTF-8 [21]XML is found to be around 4 times larger in byte size. This is a cause for concern, as the number of spreadsheets present in the corpus magnifies the implications of this larger encoding size.

3.4.7 IBM alphaWorks Java Bean

IBM alphaWorks produced the ExcelAccessor Java Bean suite [2] to access and modify the contents of worksheets using a Windows Dynamic Link Library (DLL). This option was trailed first as it had easier configuration options than the other available methods.

The advantage of the ExcelAccessor Bean is that it extracts information directly from Excel, rather than individual files. All data is reproduced accurately and formatted in the same form that the user interacts with. The Bean usually works well, but the use of native code and requirement of having Excel installed limits its overall portability as it requires the installation of a DLL in a directory that resides in the Windows path environment variable. The Windows native code in the DLL then interacts directly with an Excel process to perform the required operations. Also, it is prone to irregular crashes during larger processing jobs

involving multiple spreadsheets. When it does crash it often leaves an Excel process open on Windows 9x with subsequent runs having problems due to the still active process. This is attributed to the use of the native code in the DLL, which can also be left unintentionally running if not closed down correctly. Other unusual behaviour results from only being able to create and dispose of the bean once per session.

A typical usage of ExcelAccessor on a spreadsheet will extract two components for each worksheet. When first extracted, the data for both components is represented in a two-dimensional structure created by a Java Vector of Vectors. The first component will contain all the values in the spreadsheet and the second all the formulas. The Bean converts most value types to an equivalent Java Object, such as a Double, Boolean, String, or Date. Only a String value is required for the toolkit's current representation, which is created by casting these values. Formulas are equally simple to extract as Java Strings because they are already in a format that matches that seen by the user.

3.4.8 JExcelAPI

An alternate extraction process found uses JExcelApi by Andy Khan [28]. Being pure Java it integrates well with the rest of the toolkit and doesn't require native code libraries. The reading also tends to be faster as there is no communication with a DLL or Excel process.

Unlike the ExcelAccessor Bean, the data is extracted on a cell-by-cell basis. As each cell is iterated over, its content type can be queried to determine the information that is to be extracted. Any non-empty cell containing valid information will create a corresponding toolkit cell with the value and, if appropriate, the formula.

Some element types produce problems in the current 2.2.8 version, such as array functions, intersection area references, absolute references becoming relative in formulas, and files using a BIFF from early versions of Excel and the more recent XP version.

Another issue involved with this API was the requirement for additional memory to be allocated to the Java runtime than the default provides when working with larger files. This issue can usually easily be addressed using command line arguments to allocate a larger upper bound for the heap size.

3.4.9 Future opportunities

The capability to extract values and formulas in the user observed format from all occupied cells of the current range of freeware Java Excel readers is improving constantly, as most are still actively in development. An example is the Apache POI (Poor Obfuscation Implementation) project [3], where formula support is still being added. Tests that were run using what support that was currently available indicated impressive performance results would be possible using POI.

3.5 Internal representation

Working directly with Excel files stored in raw BIFF through a third party library can be difficult and have an impact on performance. Early on in the project a decision had to be made about the suitable format for the internal representation of the spreadsheets to use for both in memory and persistent storage. Such a format had to be flexible enough to provide source information for all analysis and visualisations while still supporting space efficient encoding to disk and good in-memory performance.

Using Java object structures stood out as the natural choice, as it would integrate well with the rest of the Java based toolkit, serialise to disk for easy storage, maintain platform

independence, and provide good performance when interacting with other Java objects. To ensure that serialisation to disk is possible, all content in the structure had to be serialisable or tagged transient while still being derivable from other encoded data. The majority of encoded information that the structures contain can be represented using combinations of Strings, integers, arrays, Vectors, and inter-object references constrained to the same set of objects.

As the analysis and visualisation tools may make many passes through a particular spreadsheet, while collecting data and constructing metrics, it is important that a large amount of information is recorded in the internal structure, rather than derived every time the workbook is opened. Much of this information will be redundant as it is already encoded in the workbook. A particular example of extra information that is stored for performance reasons is the dependencies between cells. Each cell has two structures, one for containing references to all cells that refer to it, and a second for the cells it refers to. This information is invaluable during later analysis and prevents the toolkit making many passes through all the cells to reconstruct this information.

The overall data structure will be a hierarchy that most spreadsheet users would be familiar with. At the highest level in any spreadsheet is the workbook. The workbook is a collection of worksheets and other layers, such as graphs, that make up the contents. Each worksheet is then composed of a grid of cells with a maximum dimension of 256 columns by 65536 rows. As mentioned in the limitations section, merged cells, embedded graphs and other similar elements will be ignored, as will any changes to cell width of height. Rows and columns are not explicitly represented but rather exist implicitly as part of the grid structure stored for every worksheet.

The toolkit's cells are conceptually very similar to those present in Excel. They provide a container for a value, and a formula if one is defined. As mentioned above, they also provide structures to link to all antecedent and descendent cells. To improve performance for later analysis, we convert each cells A1 notation style coordinate to its corresponding numerical column and row values.

Several limitations where deliberately imposed on the internal representation to simplify aspects of our project. These mainly included elements that did not integrate well with the grid based structure, such as floating elements (graphs and images) and other formatting like merged cells, and cell dimensions (width and height). Any imbedded programming like macros or Visual Basic for Applications that allow Excel to go beyond normal spreadsheet programming were also excluded from the internal model. Currently not supported, but with the possibilities for future expansion, include label references for cells and ranges in formulae and arrays of cells.

3.5.1 Grammar

The design requirement of using varying methods to read Excel files meant that in many cases there was a need to parse the string representation of a formula to extract the structures and dependencies of interest and reconstruct them in the toolkit's internal format. Constructing this parser first required a grammar to describe the syntax of standard Excel formula. My research efforts failed to find a publicly available grammar from Microsoft. I did however find several other researchers who were proceeding along similar lines.

Markus Clermont had assembled the BNF grammar shown in table 3.1 as part of a compiler-construction course (the version I converted it from was originally written in German, and several non-terminal labels have been altered for clarity). It provides a syntax very similar to that used by Excel with a few exceptions. Cells are addressed using the R1C1 notation and semicolons are used in many places where Excel would normally use a comma.

```
Spreadsheet ::= Row {col Row} col
Row ::=         (Formula | Number | string) {";"Row}
Number ::=      ["-"] digit {digit} ["," digit {digit}]
Formula ::=     "="FormulaExpression
FormulaExpression ::= Expression {("+"|"-"|"*"|"/"|"^") Expression}
Expression ::=  ("("FormulaExpression")")| Number | CellAddress |FunctionCall
CellAddress ::= "R"("\$" AbsCord)|RelCord "C"("\$" AbsCord)|RelCord
RelCord ::=     ["-"]digit{digit}
AbsCord ::=     digit {digit}
LogOp ::=       ">" | "<" | "=" | "!=" | ">=" | "<="
LogicalExpression ::= FormulaExpression [LogOp FormulaExpression]
LogArgList ::=  LogicalExpression ";" LogicalExpression
FunctionCall ::= ("IF(" LogicalExpression ";" FormulaExpression,
                 [";" FormulaExpression] ")" |
                 (GroupFct"(" RargList ")") |
                 ("NOT(" LogicalExpression ")") |
                 (Function "(" ArgList ")") |
                 (LogFunc"(" LogArgList ")") |
                 ("SVerweis(" FormulaExpression ";"
                  CellAddress [":" CellAddress] ";"
                  FormulaExpression ";" FormulaExpression ")")
GroupFct ::=    "SUM"|"MAX"|"MIN"|...
Function ::=    ...
LogFunc ::=     "Or"|"And"|...
ArgList ::=     FormulaExpression {";" FormulaExpression}
RArgList ::=    RArgument [";" RArgument]
RArgument ::=   RangeReference | FormulaExpression
RangeReference ::= CellAddress ":" CellAddress
```

Table 3.1: A BNF Grammar to describe Excel formula syntax

I have also slightly altered the original syntax for clarity reasons.

Using Markus Clermont's work as a starting point I created a formula parser that used a modified version of this grammar presented in the Table 3.2. This grammar is mainly focused on extracting all the referencing components. It is also possible to recognise all the basic operators and function tokens along with their nested arguments. The syntax is not purely correct, for example it does not enforce the correct arguments for functions, but it does greatly simplify the treatment of functions for parsing. Additional constraints are enforced as part of the parser, such as limiting the row references to between 1 and 65536 (inclusive) and creating a separate expression for the possible sheet names as defined in the workbook.

Parts of the parser design are improved by using regular expressions, again from the GNU package [13], to accelerate the process. While advancing through the input string each regular expression is trailed until a match for the head of the string is found. The matched string is then used to construct the corresponding object with appropriate attributes. For structures such as cells this process will often involve looking up currently stored cells from the grid structure to prevent duplicate information being created.

```
Formula ::=      "=" FormulaExpression
FormulaExpression ::= Expression {Operator Expression}
Expression ::=   ( "(" FormulaExpression ")" ) | ReferencePrefix | FunctionCall |
                  number | string
Operator ::=     "\&" | MathOp | LogOp
MathOp ::=       "+" | "-" | "*" | "/" | "^"
LogOp ::=        ">" | "<" | "=" | "!=" | ">=" | "<="
ReferencePrefix ::= WorkbookRef | SheetRef | Reference
WorkbookRef ::=  "[" workbook_path "/" filename.xls "]" SheetRef
SheetRef ::=     Sheetname"!" Reference
Sheetname ::=    worksheet_names
Reference ::=    AreaReference | Cell
AreaReference ::= Range | Intersection | Union
Cell ::=         ["$"]Column["$"]<Row>
Column ::=       Alpha | ("A" | ... | "H") Alpha | "I" Alpha
Alpha ::=        "A" |  | "Z"
Row ::=          digit [digit] [digit] [digit] [digit]
Range ::=        Cell ":" Cell | Vector
Vector ::=       ColumnRange | RowRange
ColumnRange ::=  Column ":" Column
RowRange ::=     Row ":" Row
Intersection ::= Cell " " Cell
Union ::=        Cell "," Cell
FunctionCall ::= Function "(" ArgList ")"
Funciton ::=     "IF" | "NOT" | ... | "SUM" | "MAX" | "MIN" | ...
ArgList ::=      FormulaExpression ["," FormulaExpression]
```

Table 3.2: A BNF Grammar for parsing Excel formula components

3.6 Analyser — Calculation of corpus metrics

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 to be undertaken. Within corpus linguistics such a tool is referred to as a concordance program.

With the Excel files stored in an 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.

Analyser incorporates all the different styles of corpus analysis that can currently do on collections of spreadsheets. Most often this involves iterating over the toolkit's internal representation of workbooks that Extractor creates. The workbooks are usually passed to specialised analysis methods that are then targeted at the style of analysis desired. Due to memory requirements for processing large individual spreadsheets, only one file is in memory at any one time.

In support of Analyser are the MathVector and Grid classes.

MathVector is a representation of mathematical style vectors, in 3D space, with a starting point and magnitude. It provides simple methods to find average vectors from a collection, unit vectors, and for pairs of vectors the angles between them as well as the cross and dot products. These abilities are particularly useful when doing analysis with inter-cell dependencies, such as finding the vector angles from the vertical axis.

As the name suggests, the Grid class provides storage for arbitrary objects in a dynamically scaleable array. The main focus is on removing the need for the programmer to track the current dimensions when inserting new values, instead expanding as required when a new value is inserted. Grid objects have the ability to output the data they contain into a format suitable for use by the visualisation tools (like comma-separated values or an array of doubles). This class is also used for worksheets in the internal representation to store all the cells.

3.7 Displaying the Information

The final phase in the process is to utilise the metrics and structures constructed by the toolkit to generate visualisations that address issues with spreadsheets and attempt to reveal interesting end-user programming aspects.

A collection of several programs satisfies the production of the large variety of visualisations created by the toolkit. The input data for each visualisation tool is highly dependent on the information to be displayed.

3.7.1 Available Visualisations

We have created 10 main forms of visualisation available for the data, although there is a degree of overlap between several of the fields. The general list of available techniques can be classified under one of the following headings: 2D spatial, 3D spatial, 2D logical, focus + context, animated spatial, and animated logical.

3.7.2 Visualisation Tool Support

Two main third party tools are used to convert analysed data in to useful visualisations. The first is VisAD (an acronym for Visualization for Algorithm Development) [23], which is a Java component library for interactive analysis and visualisation of numerical data. VisAD is a powerful (and sometimes complex) visualisation tool with the ability to display data in a range of forms. It makes use of the Java3D environment to generate three-dimensional images that can be manipulated in real time.

VisAD has a powerful utility called a spreadsheet that provides an environment for creating visualisations from data files. Like an Excel spreadsheet, it has columns and rows of cells. Each cell can contain a visualisation and, via formulas, combinations of other cells can create new visualisations. Data can be loaded in from a large range of formats including comma- separated values (CSV). The main uses of VisAD have been for creating surface maps, heat maps, and basic vector plots. A typical usage involves converting the input data from a 2D double array into a flat field, if the 2D array has X columns a Y rows the flat field will contain the same data in a single column with a length equal to the product of X and Y. This information can then be passed directly to VisAD classes to create either 2D or 3D visualisations.

The second tool used to generate visualisations is Excel itself. The data is imported via comma-separated values and used to create classical style diagrams such as bar graphs and radar charts. In some cases, additional computations are performed to further aggregate the imported data.

All remaining diagrams are produced directly from Java using the Swing libraries. This method offered the greatest flexibility to create novel images, but also required the largest degree of programming on my part.

Chapter 4

Collection of visualisations

One of the main focuses of this project was the creation of a series of visualisations that aid in revealing interesting properties of spreadsheets, whether this is just exposing the underlying structure to someone who is unfamiliar with an existing complex spreadsheet, or examining a corpus for larger patterns. An important feature of many of the visualisations is that they are based on structural characteristics, which are found through inspection of the low level structures in individual spreadsheets and aggregated analysis of the corpus.

4.1 Brief background on Information Visualisation

Research in the development of information visualisation is an active field that holds much promise; primarily due to the success of many visual tools that indicate the graphical medium has the powerful potential to unlock the human visual system. The practice of information visualisation is an expanding field, with origins in cartography and astronomy. As time progresses, visualisation techniques are becoming more mature and involve degrees of abstraction to convey new information using spatial relationships (distance, size, shape and orientation), colour (brightness, transparency), temporal encoding (animation), and interaction (malleability and events) to name but a few approaches. In most modern visualisations these abstractions are combined to convey important information to the user in a way that doesn't overwhelm the senses.

Many consider that documentation on the practice of information visualisation started with the classical work of Jacques Bertin in 1967 with publication of the Semiology of Graphics [8]. This work formed a large portion of the origins of information visualisation and covers the material in a rigorous, yet understandable way. Topics covered include the foundations of the analysis of information, the basic variables, the rules of graphic systems, including creation and legibility rules, and its application to diagrams networks and cartography.

Price et al. use the one of the definitions given in the Oxford English Dictionary for "visualisation" as "the power or process of forming a mental picture or vision of something not actually present to the sight" [6]. The usual interpretation of this form of visualisation is associated with the concept of gaining information through the human eye and resulting in the conveyance of a mental image to the interpreter. This interpretation can, however, be expanded to include other senses, such as sound, touch and smell, that all could contribute to the formation of a mental image. As this project will be constrained to the resources available on a standard computer system, and many of the results will be presented on a static print media, we concentrate of those senses that work well with these 2D media. In some cases this restriction is loosened to allow the creation of dynamic visualisations that can convey information about a sequence of events in a clearer manner.

Although the above definition of visualisation is correct, an alternative meaning is used in this paper to instead refer to the actual information under observation. That is, the technique of making a visible presentation of numerical data, particularly via a graphical medium. Using definitions given by Sharon Ellershaw and Micheal Oudshoorn [17], the visualisations presented in this paper will be a mix of program visualisation and scientific visualisation. They define program visualisation to be "the application of graphical transformations to an executing program to enhance the reader's understanding of that program." While our project does not directly interact with an executing spreadsheet, it does aim to enhance the reader's understanding. They go on to define scientific visualisation as dealing "solely with the graphical representation of scientific data", with the motivation of helping users deal with the large volumes of data present, which they would otherwise be unable to process all at once. Visualisations generated by our project will use the presentation methods Ellershaw et al. describe for scientific visualisation: "Scientific visualisation attempts to convert this deluge of data into color images, in order to convey the information produced to the user in a manner that can be easily assimilated."

In the paper "A Principled Taxonomy of Software Visualisation" [6] Price et al. provide a summary of Brad Myers work spanning 1986 to 1990. Of particular interest in this review are the observations about the top level of Myers visualisation taxonomy having two axes: "their level of abstraction (from showing code or data to showing algorithms) and the level of animation in their displays (whether static or dynamic)." The visualisations presented in our project all have a relatively low level of abstraction over the information extracted, particularly the vector based diagrams. Also, the majority of the diagrams don't have any animation, although some can be dynamically manipulated and screenshots captured to create animation. When the visualisations are expanded to larger, more complex spreadsheets, there will be the need to adopt higher levels of abstraction to manage the resulting visual complexity.

There are considerable benefits to be gained by using visual information to communicate data created by the toolkit. As a communication medium, visualisation is a powerful device with a high bandwidth capacity due to the nature of our visual system. As humans perceive the outside world through their senses, there are strong links between their visual system and cognitive abilities, such as the extension of memory, the grouping of related information through visual aspects, and the finding of patterns in complex data. These links allow the interpreter to quickly absorb large amounts of information from a single image, and then derive patterns and relationships well before a computer could. Michele Lanza observes that program visualisation is "often applied because good visual displays allow the human brain to study multiple aspects of complex problems in parallel." [31]

4.2 The spreadsheet model

To aid in reasoning about spreadsheets, and the visualisations derived from them, it was useful to construct a taxonomy describing the structures present in the spreadsheet paradigm. This taxonomy provides a common naming scheme for communicating about ideas and discoveries in spreadsheet structures, describes a base theory for reasoning purposes, and motivates many of the visualisations. The longer term motivation for the construction of a taxonomy is summarised by Price et al. where they write: "a well founded taxonomy can further serious investigation in any field of study." [6]

Prior to defining the taxonomy within the context of spreadsheet visualisation, it is appropriate to investigate the definition of this term in other fields of study. Rajalingham summarises taxonomy from the biology perspective as:

> the establishment of a hierarchical system of categories on the basis of presumed natural relationships among organisms. The goal of classifying is to place an organism into an already existing group or to create a new group for it, based on its resemblances to and differences from known forms. To this end, a hierarchy of categories is recognised. [44]

Taxonomies already exist for the classification of areas that are strongly related to our project. Rajalingham's work on creating the taxonomy for spreadsheet errors [44] provided a good starting point for the construction of our taxonomy for structure. Of particular interest is the branch on semantics that considers structural and temporal details. Equally relevant from the visualisation perspective is Price et al.'s principled taxonomy of software visualisation [6]. Both provide good examples of the structure required for assembling the taxonomy.

The creation of our taxonomy was based on prior experiences with spreadsheets and research into related work. To some degree the taxonomy created for describing the structures present is ad-hoc in nature, while being loosely based on the underlying nature of spreadsheets and in particular the traits of Excel. This does not however invalidate its worth, as the classification taxonomy can be generalised to include a larger body of the spreadsheet paradigm and adapted as new research suggests the need for alterations. Ideally, this taxonomy will grow towards providing a classification of the spreadsheet semantics, and would be aimed at helping to understand how users interact with the spreadsheet.

4.2.1 Spreadsheet structure

The spreadsheet paradigm exhibits two main characteristics: the spatial arrangement of cells on a table and the logical relationships between them created by formulas. These characteristics are not entirely disjoint, with the spatial relationship between cells often having a strong correspondence to the logical dependencies between them. A third, temporal, characteristic is exhibited by a subset of spreadsheets that go beyond a sketchpad existence, and are long-lived and contain a large number of cells with complex dependencies. Markus Clermont et al. observes that due to the nature of these long lived sheets, they can go through similar evolutionary steps as conventional software [35].

These three characteristics form the highest level of the taxonomy, and are positioned immediately below the structural root node. While the temporal dimension has great research potential, the main focus of our project will be the spatial and logical dimensions due to the nature of corpus analysis.

4.2.2 Spatial

The spatial properties of cells are important for several reasons, most of which were covered in the background chapter. These include the creation of a presentation medium for the end user of the spreadsheet, an addressing mechanism that corresponds with the use of cells as variables in the program, and the concept of scope.

An important part of the spatial relationship between of cells is the lack of locality exhibited. It is generally considered to be good programming practice to avoid unconstrained jumping and only nest if statements to several levels deep, otherwise a program can become difficult to follow, or worse still, develop into spaghetti code. In contrast to a trained computer programmer, a novice spreadsheet user would think nothing of chaining together many cells via formula to construct what is in essence a large and possibly complex formula. These complex formulas can have their code distributed across large, spatially disjoint, regions. Effectively scattering the information that is required to understand their workings. This is

characteristic of the spreadsheet paradigm. Clermont comments on this problem from the perspective of error prevention:

> The principle of locality, an important concept for reducing the complexity of software, is not part of the spreadsheet model, i.e. any other cell anywhere on the spreadsheet can freely access the result value of a certain cell. Hence, the effects of an error in an arbitrary cell will potentially influence one or more results of the spreadsheet irrespective of their "distance" to the erroneous cell. Worse, the effect of an error might show at a different place than the error itself, thus further increasing the complexity of identifying faults. [35]

Scope also has an effect on the spatial relationships that the users create. Several factors affect the scope of a reference in Excel. A single reference could be within the visible screen area, between worksheets, or between separate files. The effects of screen size are also variable, as one user may only have a visible working area that is highly dependent on their monitor size and screen resolution. Experienced spreadsheet users who realise the benefit of locality will usually attempt to find a spatial arrangement of all logically related data within the minimal scope plausible. The use of cell values as a presentation medium can be viewed as a force that influences the degree of locality achieved, as certain data may need to be visible within a separate scope to satisfy appearance requirements.

More experienced spreadsheet programmers may attempt to enforce modularity through spatial arrangement, but the spreadsheet itself enforces no such restrictions. A programming concept that is strongly related to modularity is encapsulation. Again, Excel does not enforce this due to the unconstrained referencing power given to the user.

4.2.3 Logical

As hinted at in the spatial section above, there is a strong binding between spatial and logical components resulting from the visual environment. Typically, cells that have a high degree of logical coupling will be spatially close due to the user attempting to enforce locality. In many respects the connection between logical and spatial relationships is not fully symmetric, as the spatial aspects of a spreadsheet are provided to create an addressing scheme for describing the logical dependencies. With presentation issues set aside, the logical dimension of a spreadsheet becomes more important than the superficial spatial relationships. The spreadsheet interface melds both the logical and spatial aspects from the perspective of the user, but it would be possible for the computations to function with unbounded cells, as it is the articulation — the joints between components — between them that is of greatest importance.

The formula present in each cell is a potential source of complexity in the logical domain. The length of the formula, the number of operators used, and the number of functions used (both total numbers and different functions). All contribute to making the individual formula more complex. These complexities provide the motivation behind the construction of a set of metrics.

4.2.4 Metrics

The following metrics are mainly derived from the logical relationships between cells that are derived from each cells formula. Certain metrics also contain a spatial relationship component. These metrics can then be used to show supplementary information in the visualisations.

Sample Metrics for a formula:

- Length when viewed as string — a crude but often effective measure of complexity. Quite simply, the longer the formula is the harder it can be to understand. In traditional programming, it is common practice to split a complex piece of code into multiple parts, leading to modularity and encapsulation of code for easier comprehension. Formulas can create a conflict with this concept, as there are no local variables, and to use the replication function the user may prefer to have the entire calculation represented in a single, often monolithic, formula.

- Degree of bracket nesting present — like the nesting of if-statements in traditional imperative programming, large degrees of nesting can be difficult to follow.

- Number of (distinct) cells referenced and number of (distinct) ranges referenced — a measure of the dependency complexity that this cell has on others. The larger this metric is, the more complex the relationship this cell has with other cells around the spreadsheet. Often the user must be aware of the implications of changing any one of those remote cells. Clearly, the range syntax allows a potentially larger number of cells to be referenced in a single statement than a single cell reference. This effect could also be used by dividing the number of cells referenced by the number of referencing operators.

- Number of functions used — more functions can often result in a large degree of nesting and a confusing numbers of brackets.

- Number of basic operators (+, -, *, /) used

- Position of formulas and raw data in a spatial and clustering sense — this has layout implications and also effects how a user will go about learning the structure.

- Cohesion of data within clusters. Density of data — What is the difference between the average radius and maximum radius or number of nodes? A cluster with a dense centre will tend to reach out and pull other elements in due to its dense average radius.

- Branching factor information — at what rate is information dispersed through formula?

- Euclidean distance between cells in dependency trees — further reaching references are harder to follow. Locality and scope issues, are related to this.

- Coupling between sheets and workbooks. — measuring the flow of information and dependency between sheets in the workbook.

- Number of non-orphan cells with values. Orphan cells provide no computational function and must therefore only be present for visual presentation reasons. An orphan cell is a cell with no incoming or out going references — it is essentially disconnected from the rest of the sheet. It is possible to have a cell with a formula that doesn't reference any other cell, such as "=sin(60)".

- Size of the worksheet, width and height dimensions — How much of the available space is utilised? Which parts of the available real estate are people interested in using? Are there hidden tables off in the corner?

- Comparing formulas for varying types of equivalence — As discussed in the background chapter using work by Markus Clermont. A good method of detecting repeated patterns.

4.3 Task model - A discovery process for the exploration of spreadsheets.

When a user is faced with an unfamiliar spreadsheet, it can be a difficult task to proceed in building a mental model of the information present, and even harder to find only the relevant information. As complicated as this task can be, it is still important, as the implications of incorrect alterations often result in serious errors.

The visualisations created for individual spreadsheets are aimed at addressing certain phases a user will go through when becoming familiar with a spreadsheet, and to provide the information at a level of abstraction that the user is currently interested in. For a user who is planning to modify an existing spreadsheet it is important that the visualisations provide only that information which is directly relevant to what they are trying to accomplish at that stage, such as, which sections of the spreadsheet they would need to exercise more care in due to a higher level of complexity. While this is one of the main focuses of the single spreadsheet visualisations, it is not their only use, as they may be utilised at any time to provide additional information.

As the first phase in the discovery process, a user will need to know where the data is located. At the highest level this involves the number of worksheets present and their overall dimensions. Following that, a more detailed analysis at the granularity of individual sheets and then cells will help reveal information that is located away from the core data.

The next important information to address is related to the dataflow structures. This can involve locating the position of formula, source data for the formula, and cells that a conceptually grouped through ranges. The logical relationships in the dataflow structure can be used to create dependency trees that show the flow of information from source data to final output. While initially it can be useful to observe this data using the same spatial presentation that Excel uses, later visualisations can present this information in alternate forms that are designed to highlight features that may not be apparent in more traditional views.

4.4 Explorations - Areas of focus

Various different diagrams are created to address various analysed data produced by the toolkit. The main aim of the visualisations was that they should be well founded and understandable by anybody with an interest in spreadsheets or end-user programming. To aid the understandability of these diagrams they are often directly traceable back to individual spreadsheets, thus providing context to the interpreter. These visualisations are principled by and flow naturally from the taxonomy of spreadsheets created as part of our project.

4.4.1 Real-estate diagrams - Single/Corpus

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.

A map is one of the oldest and most common forms of visualisation for addressing issues of location. The easiest way to begin constructing a map for a spreadsheet is to commence with an empty grid structure and then to add details of interest as they become available. It is seldom that the full 256 columns by 65,536 rows available in each worksheet are used in any one problem, which is why the dimensions of the grid are only expanded to accommodate new elements when needed. Also, if the reverse process were undertaken, resources would be wasted during processing and the grid would have to be cropped prior to visualisation.

Collecting the source data for these visualisations proved to be a simple task of iterating over all the cells and incrementing a count for the corresponding coordinate in a grid for any occupied cells. The resulting data structure is a grid with a count at each coordinate for the number of times it is occupied in all the observed input worksheets.

It is possible to generalise this information further by only considering the utilisation of worksheet area, rather than individual cell coordinates on the grid. This is achieved by only counting the minimal area surrounding all the cells in the worksheet. For a particular worksheet the minimal area is defined to be the smallest possible range that will contain all occupied cells in the worksheet.

The real-estate visualisation tools produced for the toolkit take a very abstract view of occupancy and population information for worksheets and are capable of working with either individual workbooks or an entire corpus. The main purpose of these diagrams is to understand basic positioning information about the workbook. Using these diagrams it is possible to quickly get an impression about which sections of the spreadsheet are occupied and to what extent these sections are utilised. These diagrams also make finding sections of data that the designer has purposely tried to obscure easier to find. These obscured sections will often be positioned in a far corner of the sheet and provide functionality that does not need to be seen when using the core of the sheet, such as lookup tables.

Two alternative methods are available for viewing this information.

The simplest is a 2D approach where the occupancy data is displayed over a grid, using a coordinate scheme that is similar to how Excel displays cells. This results in a spatial organisation that mimics Excel's layout by positioning the origin at the top left of the screen and then addressing the column dimension on the horizontal axis flowing right and the row dimension on the vertical axis flowing down. The advantages of this approach include matching the users current expectation for cell position, and that relationships understood in one view can more easily be carried over to the other.

An example of the 2D real-estate diagrams produced by the toolkit are shown in figure 4.1. 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 to create a heatmap 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 will those with lower counts will be coloured towards the violet/blue end.

To create an alternative view for this information, the data was projected into 3D to create a surface map. I believe this transformation from discrete data to continuous surface benefits the viewer by smoothing out the effects on 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 [51] and VisAD [23], a visualisation tool

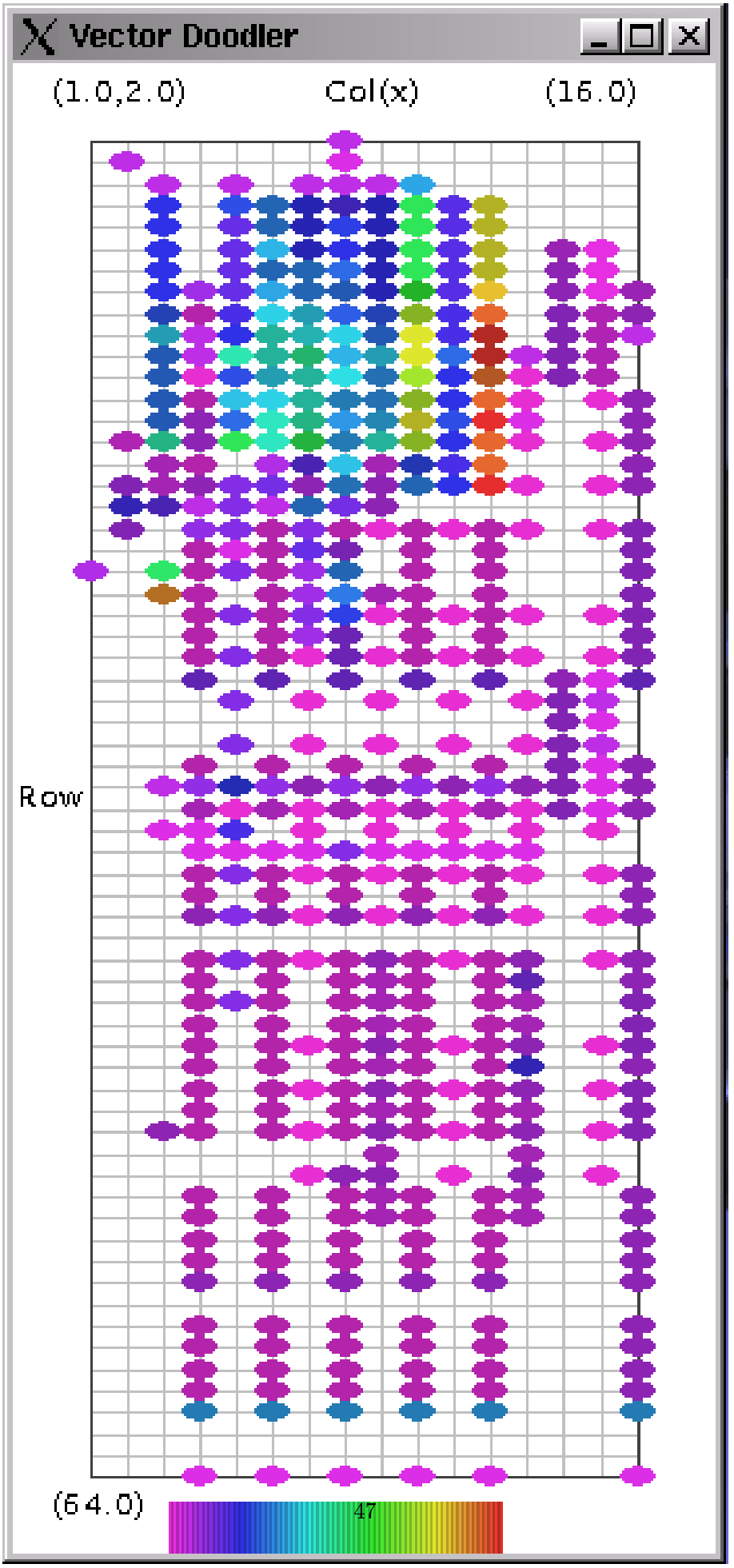


Figure 4.1: Real-estate utilisation diagram in 2D

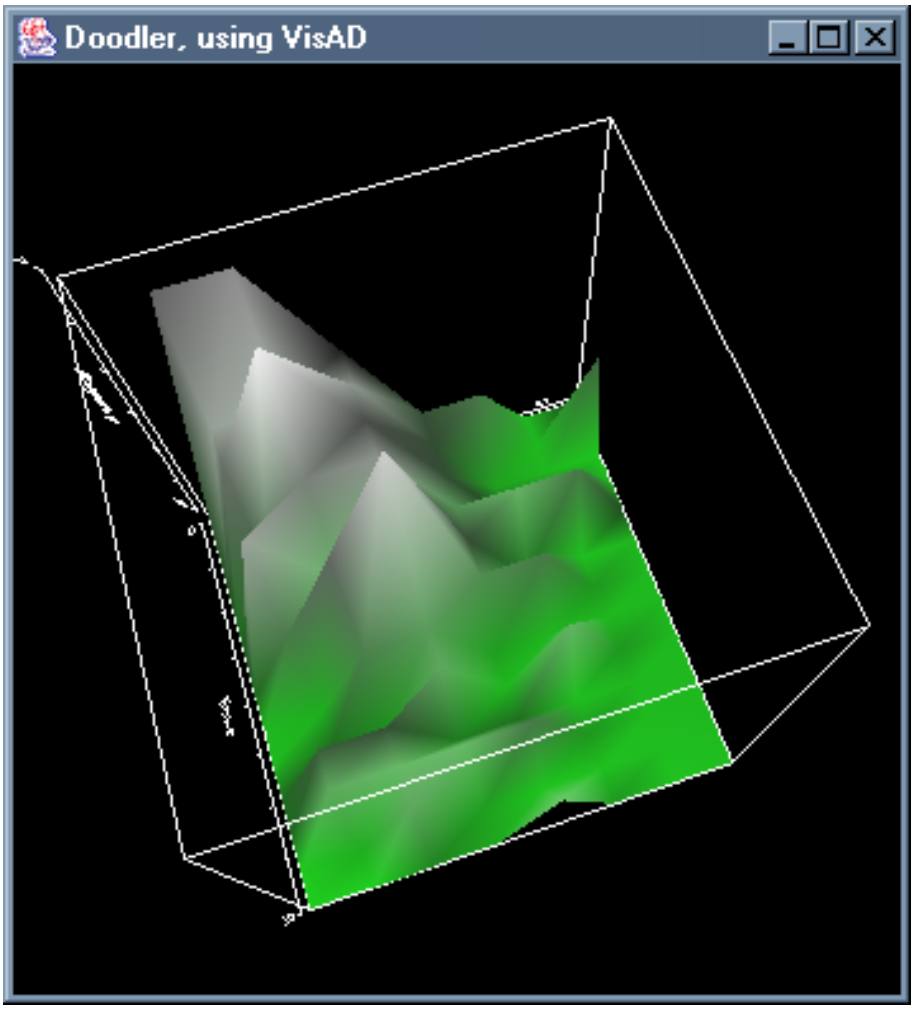


Figure 4.2: Real-estate utilisation diagram in 3D

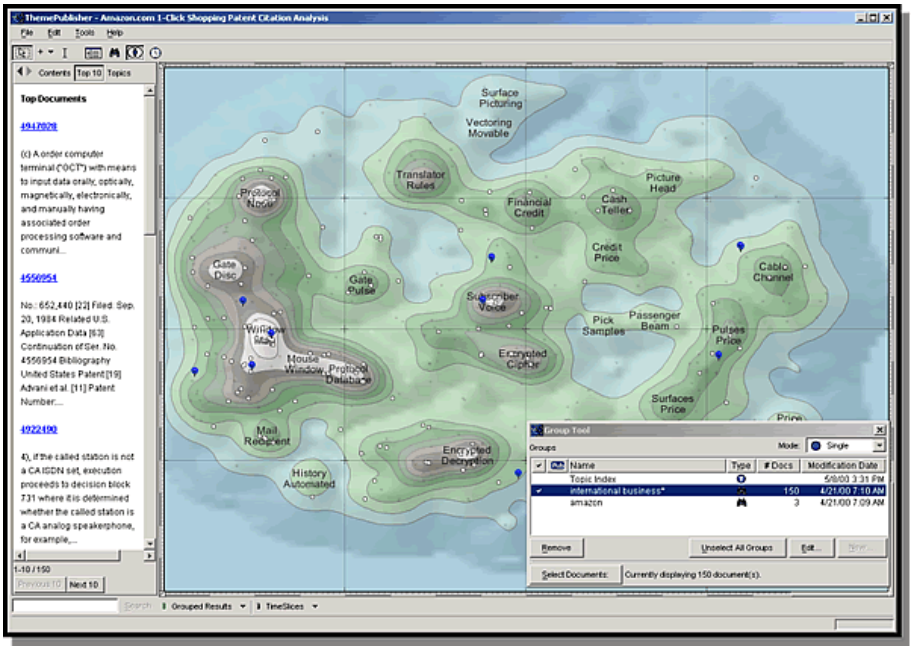


Figure 4.3: The ThemeScape topographical map of word relationships between documents.

for numerical data. These tools had the benefit of allowing the user to interact with the image via the mouse and keyboard to rotate and zoom the image. 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 4.2. 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 terrain type appearance.

Colouring occupancy levels in real-estate diagrams in the same fashion as topographical maps can utilise the interpreter's previous understanding when examining new diagrams. An example of this approach is the ThemeScape surface map visualisation from the Aureka client/server system [5]. ThemeScape, shown in figure 4.3, is a text analysis tool that visualises data sets from a 30,000-foot perspective topographical map, which is based on word relationships between a large collection of patents and other non-patent documents. The diagrams it produces bare a strong resemblance to traditional maps, particularly in contour colouring. This allows the user to carry over their prior knowledge of such diagrams in interpreting the new image.

Although this topographical colouring can be effective in helping the user gain a better understanding of what the image is conveying, other useful information can also be associated with the image through colouring. For example, the colourmap could be used for conveying information other than altitude, such as the average formula complexity metric for that cell.

The information presented in the real-estate images can be further abstracted from cell occupancy to sheet occupancy. Using this process, individual cells are ignored and instead the occupied dimensions for each worksheet are used (the minimal and maximal columns that form a range around all the cells contained in that sheet). This new visualisation is better suited for studying a corpus of spreadsheets, although it may be used for individual workbooks. When used across the corpus new details can be observed, such as the tendency for a larger sheet to use many more rows than columns (as would be expected due to the coordinate system provided by Excel).

There is potential to further refine and utilise these real-estate images to observe more

interesting spreadsheet properties. Rather than abstracting from individual cells to the granularity of worksheet dimensions as the units of interest, ranges of cells could be used. This would make the location of large input data blocks and replicated functions more apparent. A hypothesis I've had from an early stage in the project is that many spreadsheets will be destined for a print media and as such the arrangement of cells will be oriented towards the dimensions of the paper they will be printed on. In many cases this would be standard A4 sheets and so units with the equivalent cell dimensions (with Excel's default cell width and height) could also be of interest. I suspect that because many spreadsheets would be arranged to facilitate printing that an A4 (paper size) ratio pattern would emerge.

4.4.2 Clustering - Single/Corpus

All the real-estate images mentioned above rely on the user to observe the blocks of data themselves. Although 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.

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 relationship.

In the case of these diagrams such as figure 4.4, spatial similarities between rows and columns of occupied cells are investigated using rectangular or circular depictions. Of particular interest are locations where the clusters clump together, indicating islands of data in the worksheet. For these pockets of data it may be sensible to expand the cut-off value to form larger clusters. These diagrams are interesting for observing the conceptual grouping of data the user has created for the particular model. In many cases there could be a degree of fragmentation, or zoning, as users separate different areas of functionality. Much like the real- estate diagrams, clustering can help with locating hidden tables present in a worksheet.

Visualisation support in the toolkit is supplied by the ClusterGraph application. A ClusterGraph can be created for a grid of cells present at a certain sheet depth or for an entire workbook. When considering clustering for a corpus of spreadsheets, the visualisation currently doesn't scale well due to the high density of information present, making basic clustering diagrams somewhat irrelevant. Several options exist to address this problem. After the initial run of the clustering algorithm, clusters could replace individual cells as the unit of interest, providing a higher level of abstraction over the cell. With the source data at the granularity of clusters the algorithm could then be run recursively to do clustering on the clusters (after increasing the cluster radius). Another alternative is to use the coordinate occupancy levels to add another dimension in the distance functions. This could cause the clustering algorithm to produce contouring information in addition to the spatial clustering.

Formula used and example graph. ¡datamining26¿ ¡datamining27¿ ¡datamining28¿

The cluster finding algorithm used was derived from notes on data mining multidimensional data in a data-warehousing course, which in turn was derived from work on artificial

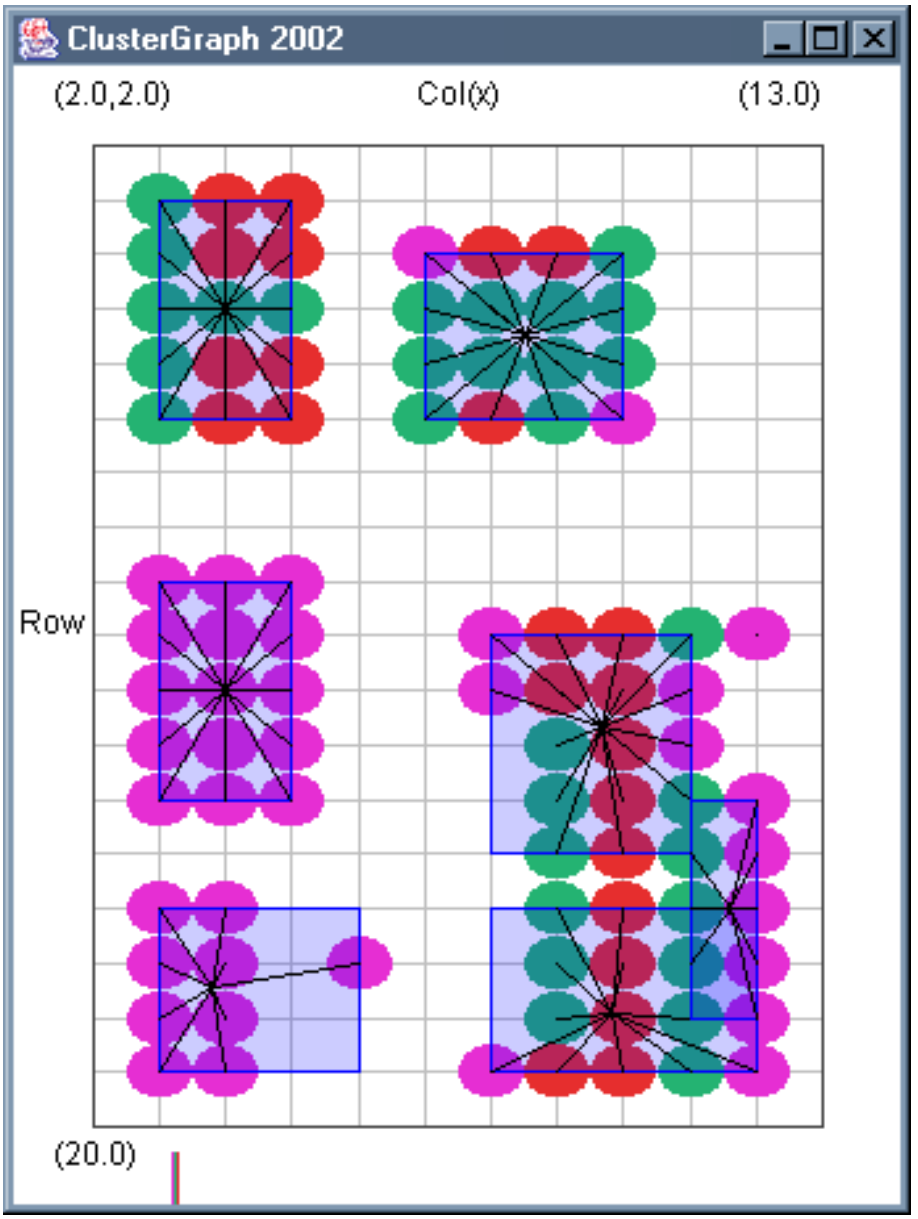


Figure 4.4: Clustering in a single worksheet

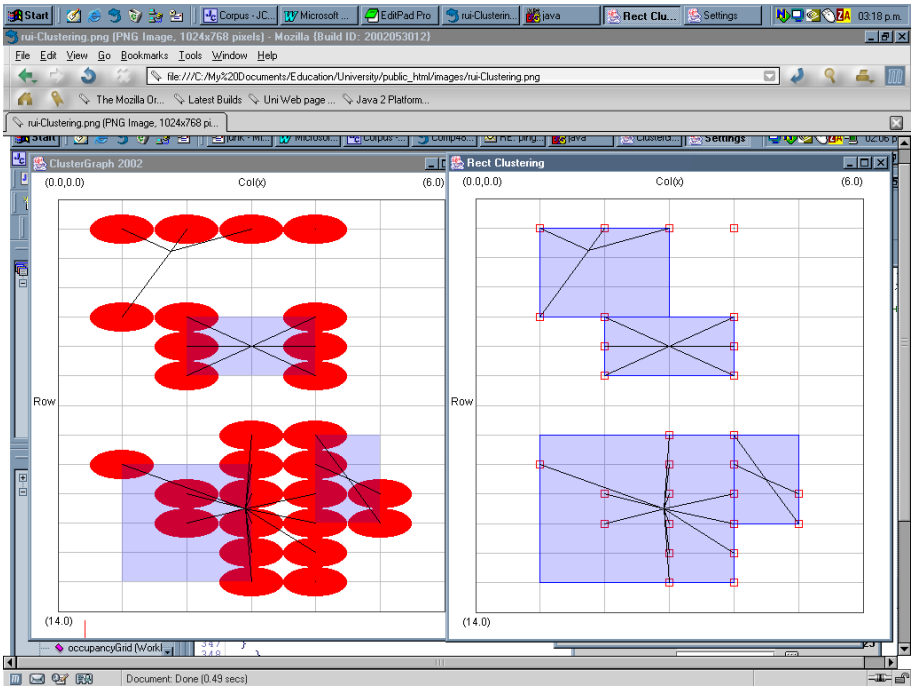


Figure 4.5: Clustering in a single worksheet

intelligence. Although this algorithm was simple to implement within the toolkit, other clustering algorithms may be better tuned to the grid like spatial structure of spreadsheets. One cluster search strategy considered involved scanning each worksheet on a row-by-row basis looking for blocks of data. When a occupied cell is encountered that is not part of a previous block, all the cells reachable in the immediately adjacent blocks would form a larger block unit. This block forming technique would expand out recursively to encompass all cells that are spatially connected.

This algorithm has the potential to find all the spatial clusters in a worksheet, and with alterations could instead search for logical clustering. Logical clustering would involve using the same recursive search across adjacent cells, with the additional requirement of Clermont et al.'s formula equivalence, as document in the background chapter.

4.4.3 Formulas at a Worksheet Level

After examining the layout of the spreadsheet it is useful to then focus on the dataflow through the 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, or 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 4.6 demonstrates the four main techniques that Excel allows a user can use in

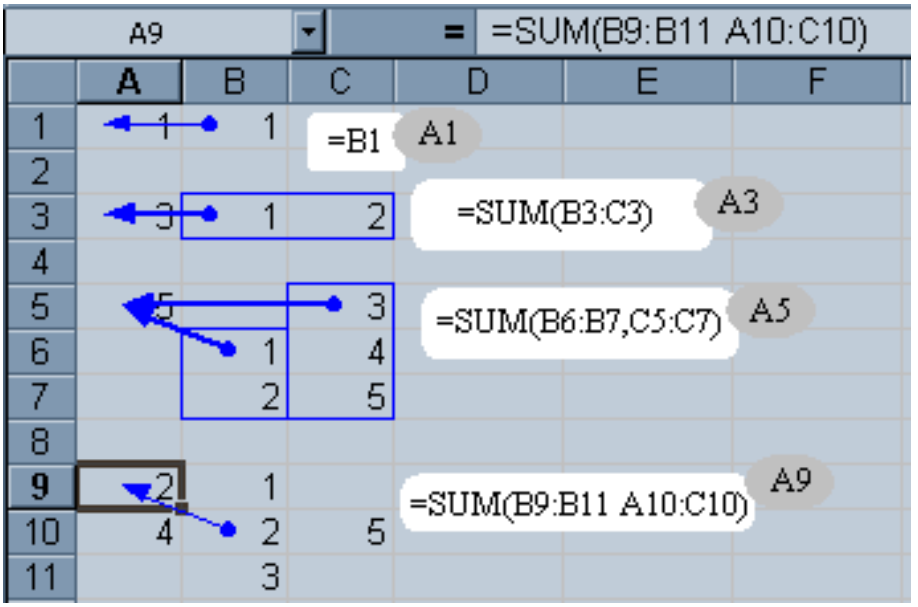


Figure 4.6: Excel's precedent trace auditing tool for sheet 1. Note that I have added by hand boxes to the right of each cell containing the relevant formula.

a formula to reference other cells and the precedent trace arrows added by Excel's built in auditing tools. Notice how Excel's trace is somewhat deceptive in the case of an intersection, in this case looking more like an individual cell reference. In figure 4.7, 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 4.8 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. [25].

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 focuses of our project.

Figure 4.9 demonstrates the colouring of relative and absolute cell references used by

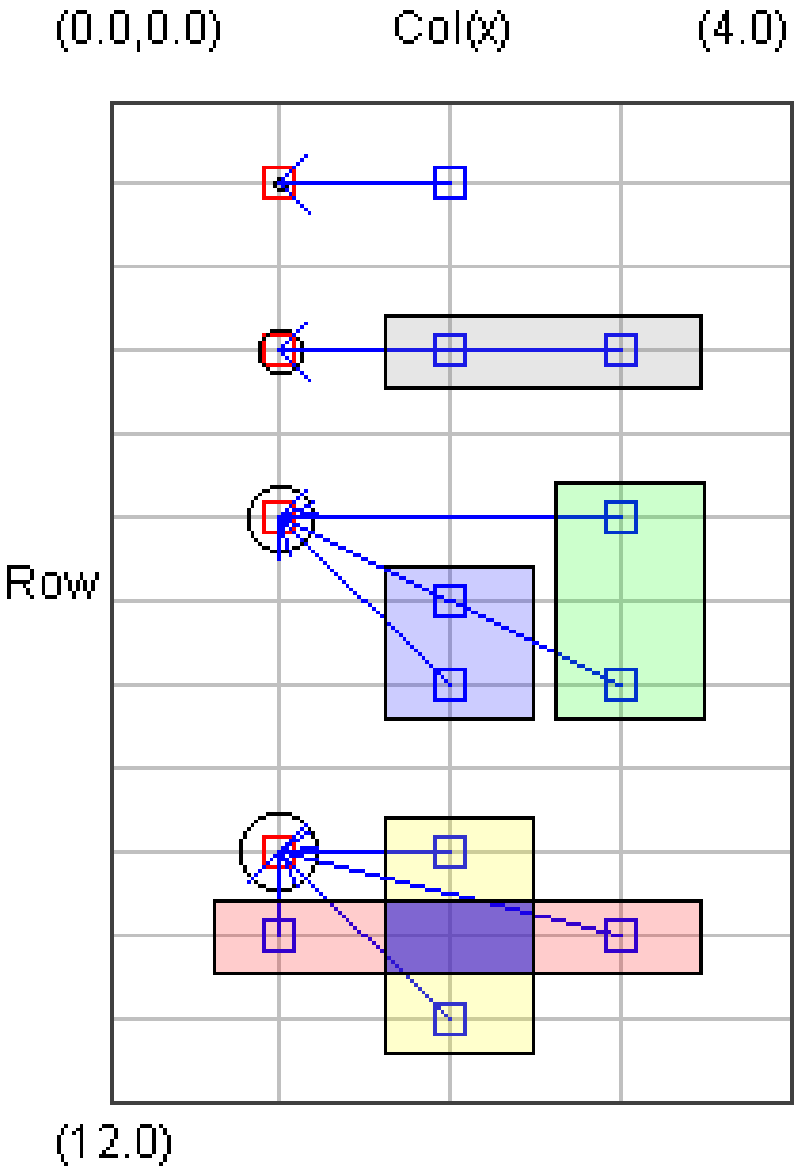


Figure 4.7: Formula extracted and displayed by the toolkit

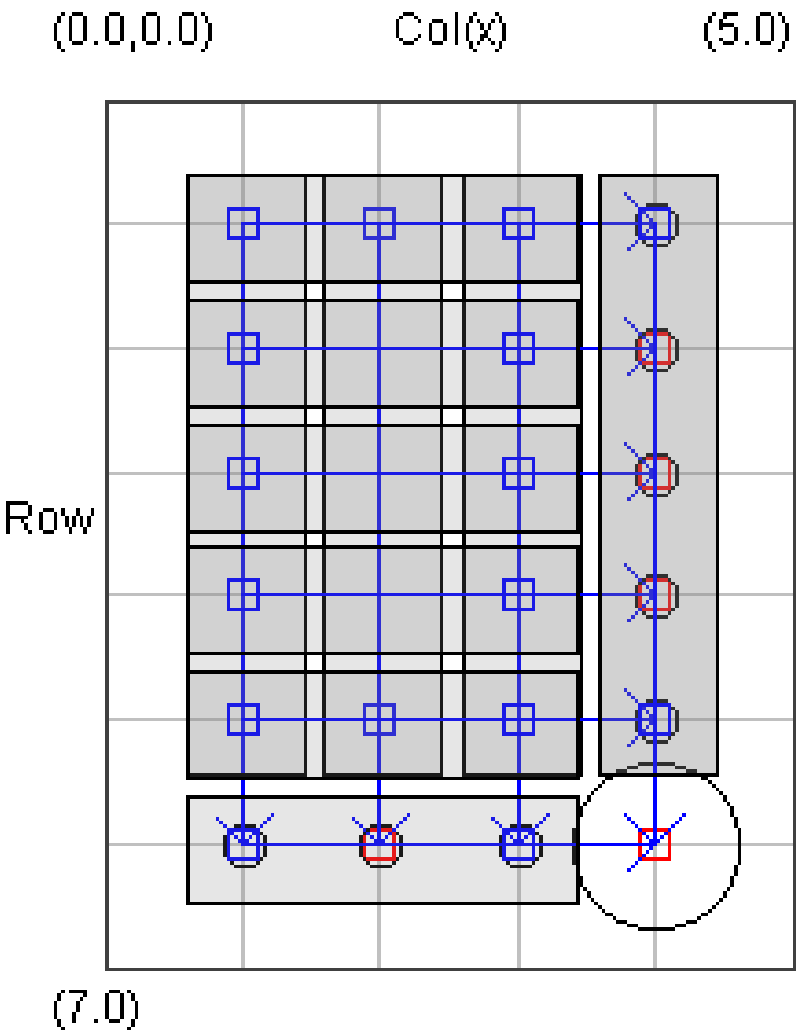


Figure 4.8: Using the summation operation on both the columns and rows in a table.

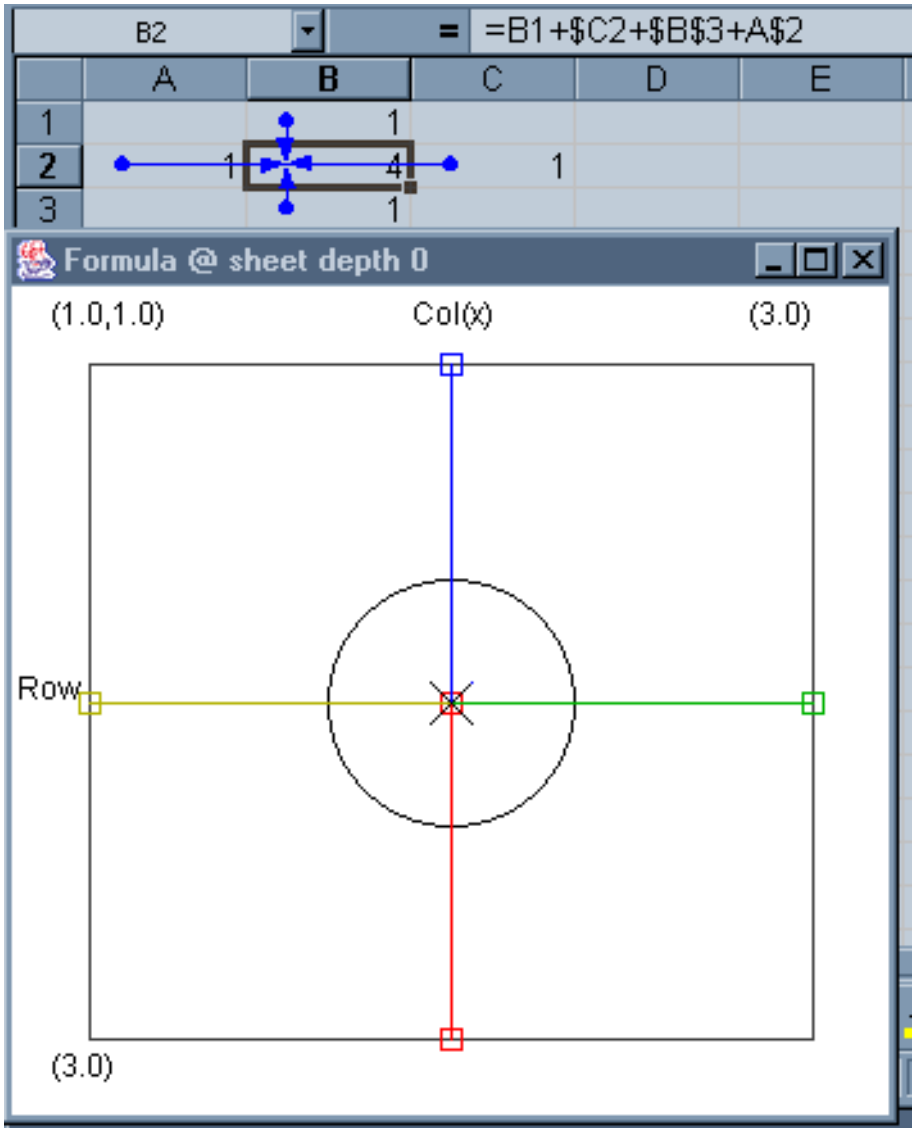


Figure 4.9: Both Excel's and the toolkit's trace of simple absolute and relative inter-cell dependencies.

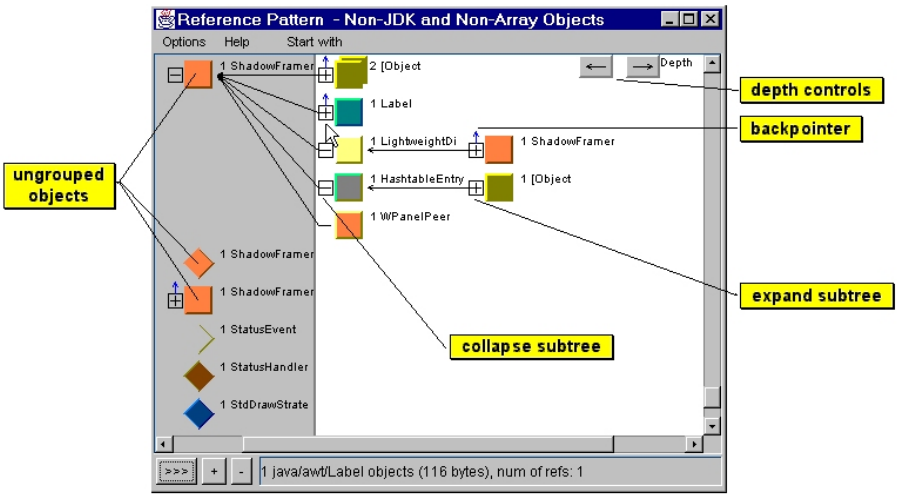


Figure 4.10: Jinsight's Reference Pattern View for exploring data structures and finding memory leaks.

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 partially 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.

Related work to this search for patterns has been done by IBM alphaWorks on the Jinsight project [24]. The Jinsight project covers many visualisations, with the relevant reference pattern view shown in figure 4.10 being useful for exploring data structures and finding memory leaks.

The absolute and relative referencing capabilities of the toolkit are currently only applicable for references to individual cells. There is the future opportunity to expand the colouring scheme to include range references. One possible approach would be to colour range boxes and arrows as single references are. This would require simplification of the depiction of ranges, perhaps to show only the minimal containing range for intersections.

It is possible to substitute the source data used for the generation of the 2D real-estate visualisations the any data in the same format. This allows the creation of an alternative view for measuring formula complexity to be created using the same visualisation application. In the range and summation example shown in figure 4.6, a metric is used to define the size of a circle indicating the complexity of a formula. Either this string length metric, or another appropriate metric from the spreadsheet model section, can be used to create the source data that is displayed by either a circle with dimensions specified by the metric or a colour map based on the range of complexity values observed.

4.4.4 Precedent Tracing - Corpus/Single

Techniques presented for formulas in the prior section can be applied on a much larger scale to visualise the entire dataflow structure present in single worksheets or across a larger corpus.

Each dependency created by logical references between cells in formulas can be converted to a pair of mathematical vectors. From the toolkit's perspective, each dependency has two component vectors. The first is a displacement vector that is transparent when viewed and alters the starting coordinates for the second vector to the position of the referencing cell. The second vector has a magnitude equal to the distance between the two cells and an orientation that causes it to end at the coordinates of the referenced cell. Plotting all second vectors can show the general flow direction created by the dependency graph and help make the patterns in the spreadsheets more apparent.

When viewed for a single worksheet, the vectors serve to graphically represent the dependencies created by the formula in each cell, helping users visually understand the dataflow structure without any tedious interaction with cells or formula. Cells with a complex dependency on many other cells become more apparent due to the large out degree of arrows.

With a single large worksheet, viewing all the dependency arrows in one visualisation can still convey useful information to the user, partially as most larger problem models have a high degree of homogeneity [33]. As the number of vectors grows, there can be the need to perform abstractions to reduce the amount of information presented to the user. Although this is seen as an important scalability issue for the visualisations, techniques to resolve it have only been considered in theory.

One possible solution to address the clutter that can be created by having a large number of vectors displayed in a small space is to make the image interactive. All vectors by default will be semi-transparent and when the mouse is over a certain cell its vectors will become opaque. This interface could be further enhanced by allowing the user to fix and then toggle the opacity by clicking on a cell that vectors originate from. However, care must be taken to not recreate the problem of requiring a high degree of user interaction to explore the dataflow structure.

When the precedent trace exists for a data sourced from a larger number of worksheets, two simple alterations to the vectors displayed can reduce the volume of information displayed. Firstly, the magnitude of each vector could be altered to create unit vectors. 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 usually combined to create a visualisation depicting the direction of dataflow. Figure 4.11 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, with the columns assigned to the horizontal axis and rows to the vertical.

Using the same clustering technique outlined above it would be possible to change the unit of visualisation from a single cell to a cluster. The outgoing vector for all the cells present in the cluster could be the average of all their outgoing vectors. When combined with the spatial aggregation that clustering already provides this technique could present patterns in a larger homogenous model clearer.

Another solution considered involves abstracting over the dataflow structure. In many cases this involves aggregation over sets of logically related vectors. The initial aggregation considered involves summing all the vectors between pairs of root nodes and leaf nodes in the dataflow graph, which will in essence create a new vector from the root cell to the leaf cell. The presence of range references complicates this matter due to the potential for great breadth in the dependency tree.

All the dependency visualisations considered above have a two-dimensional perspective

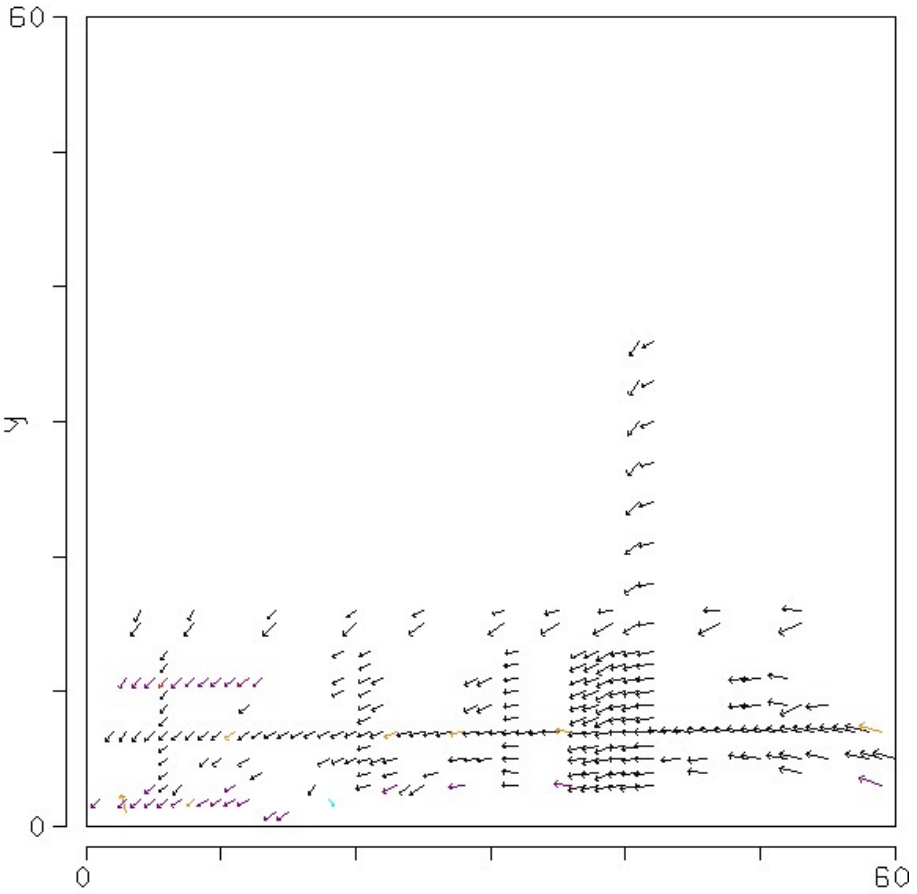


Figure 4.11: Data dependency flow using the average unit vectors.

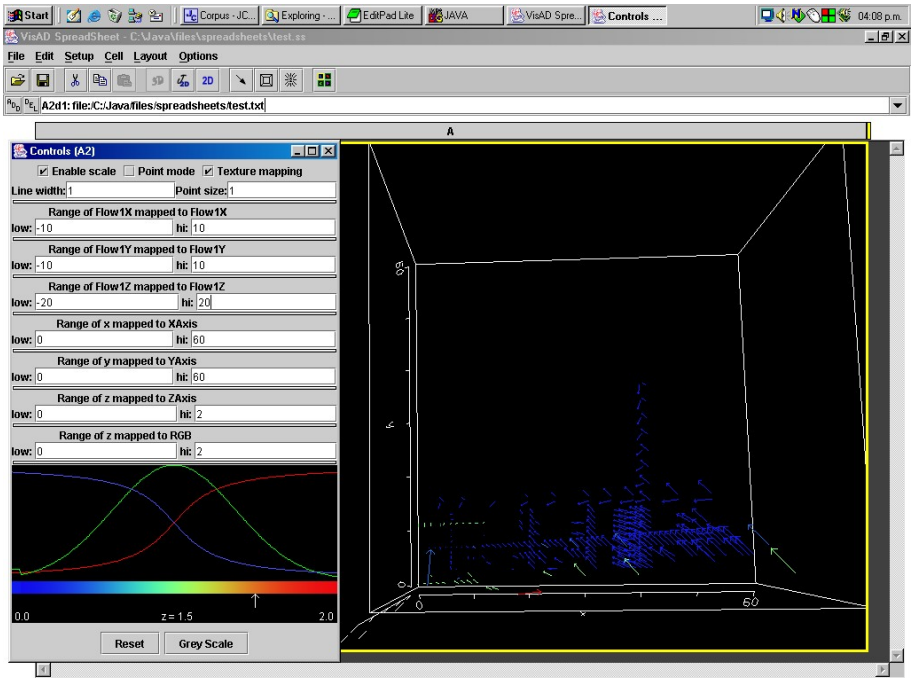


Figure 4.12: The flow of data between sheets in 3D.

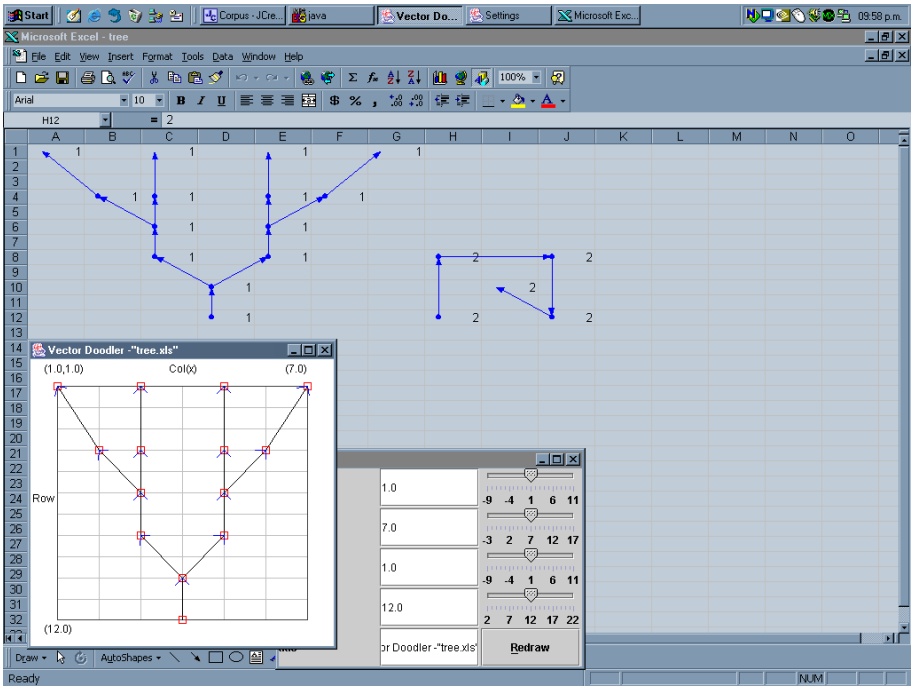


Figure 4.13: A single dependency tree.

on the dataflow structure. The worksheet stack that Excel provides can also create a three-dimensional flow of data as formulas reference data in sheets that are above or below their worksheet depth. We suspected that most of these three-dimensional dependencies will flow from the higher sheets towards the lower sheets; with the highest sheet in the stack considered to be the first worksheet the user sees when they open the spreadsheet. Some visualisations have focused on this area, mainly using 3D images with dependencies represented as vectors. Figure 4.12 is an example showing this flow. It should be noted that this particular visualisation relies on the user being able to manipulate it to observe the full structure.

4.4.5 Dependency Trees

For any particular model the entire dataflow graph can be large and complex, featuring a range of layout structures for solving varying problems. In many situations the user may instead be interested only in a single part of this larger structure, whether it be a trace from a root cell to all the leaf cells that depend on it, or a trace for a leaf cell to all the root cells that it depends on.

To help the user focus on a single tree the toolkit is capable of identifying all the root and leaf cells in a worksheet, and then for any particular cell, tracing either all the cells that depend on it or those cells that is depends on. The resulting set of cells can then be stored in a new grid structure and used to produce visualisations focused on the single tree.

Many of the prior visualisations are still appropriate for a single dependency tree, with the particular advantage of a reduction in the volume of information. An example of a simple tree trace in given in figure 4.13.

In addition to utilising the prior visualisations, new images can are created to investigate characteristics that would not have been possible with the larger structures present in the entire dataflow model.

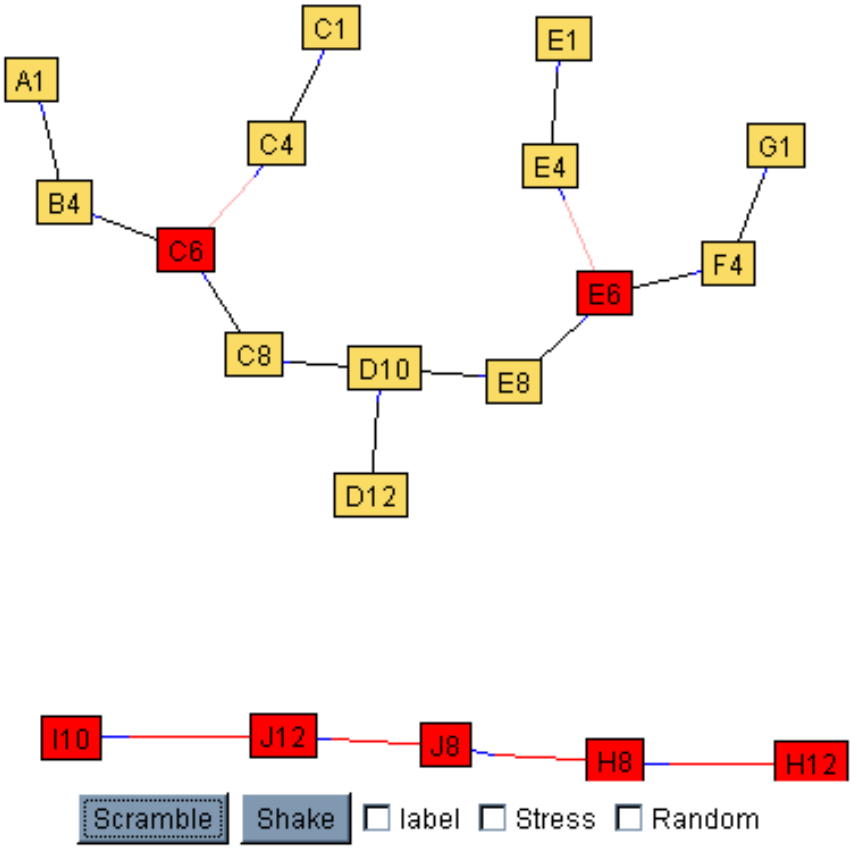


Figure 4.14: A spring view of the dependency structure between cells. By disregarding the spatial bounds usually enforced on cells, structures such as the chain between cells I10 and H12 become clearer.

Spring View

The first of these tree specific visualisations uses a 2D layout algorithm based on the idea of spring forces to arrange the spatial positions of the cells, with an example given in figure 4.14. To do this, cells and the edges between them 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.

Fisheye view

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

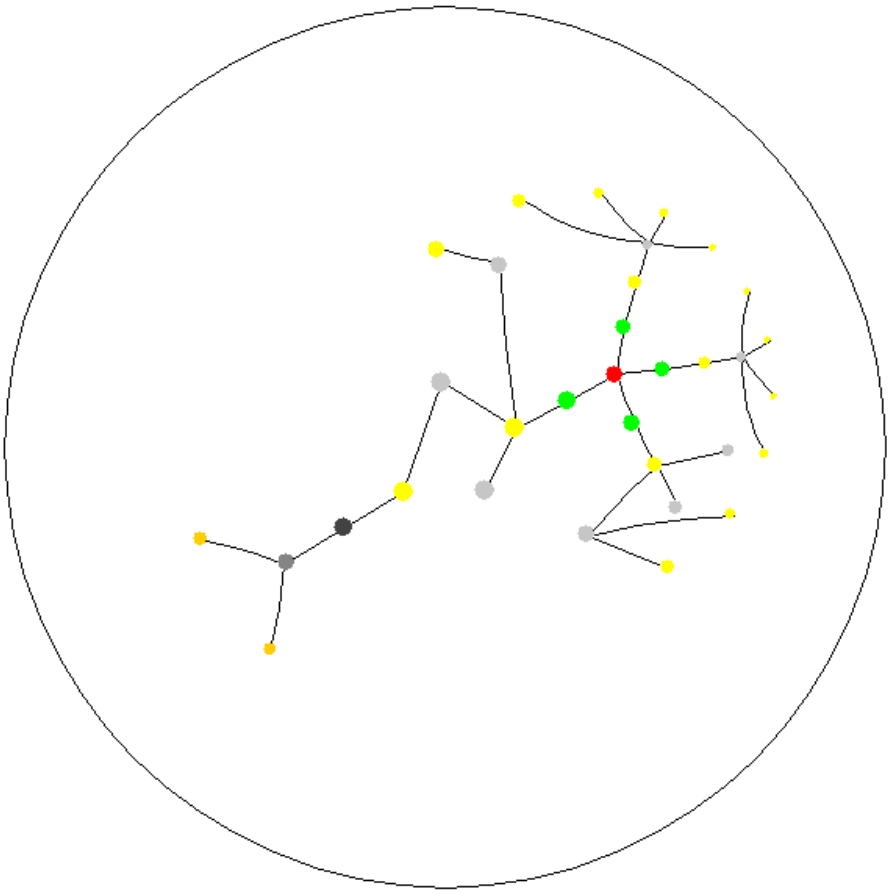


Figure 4.15: The fisheye view of 4 dependency trees.

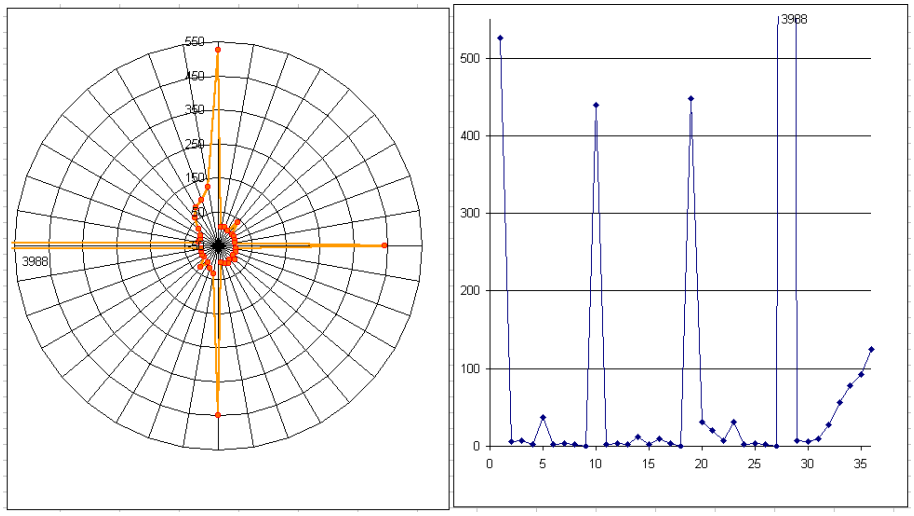


Figure 4.16: Radar and line graph views of bucket data for 259 spreadsheets from the corpus. The upper bucket count is cropped at 550.

patterns. The fisheye visualisation involves warping this tree over a hyperbolic lens that makes it is possible to achieve both focus on an aspect of interest near the centre of the visualisation and the larger context for that aspect. Figure 4.15 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.

4.4.6 Radar View - Corpus

The direction of dependency vectors leaving a cell is investigated using the visualisation referred to as the compass view. This visualisation is applied to dependency direction data contained in a series of buckets extracted from a corpus of spreadsheets. 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.

Storing the bucket data using comma separated value files allows the data to be read by Excel. Excel's graphing features can then be used to display the data as either a radar of a line graph, as the example figure 4.16 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.

Worksheet Centre - Corpus

In figure 4.19 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).

Function Utilisation - Corpus

After applying the toolkit's parser to the formula in each cell in is possible to count the functions utilised in each worksheet. Using this data and the Excel defined categorises for each function figure 4.20 is produced. This bar graph addresses the degree to which each

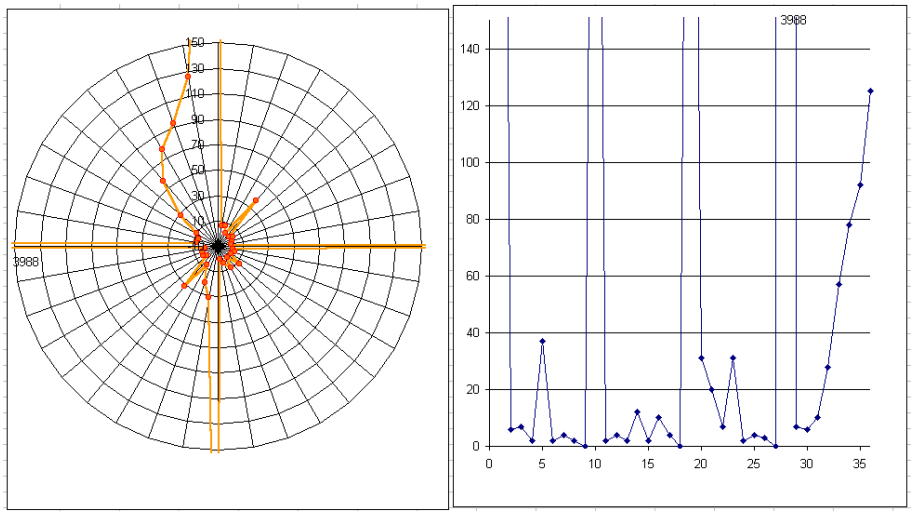


Figure 4.17: The upper bucket count is cropped at 150

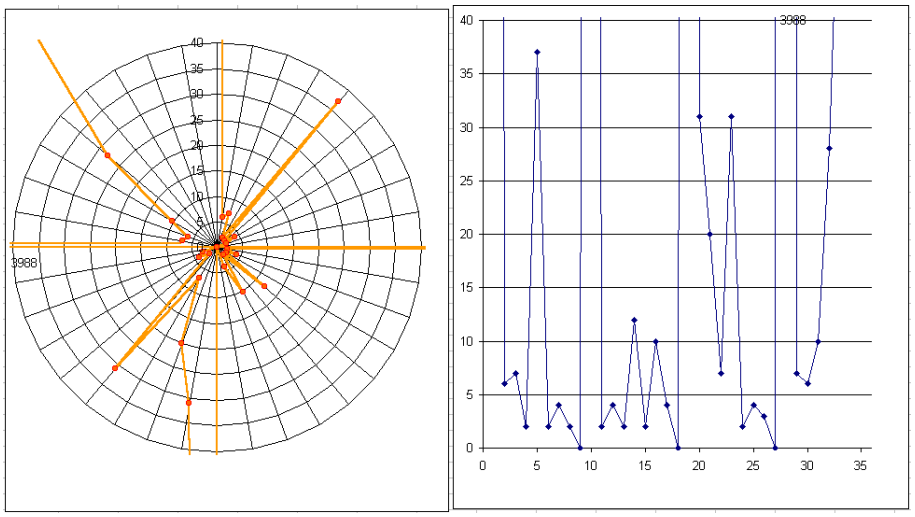


Figure 4.18: The upper bucket count is cropped at 40

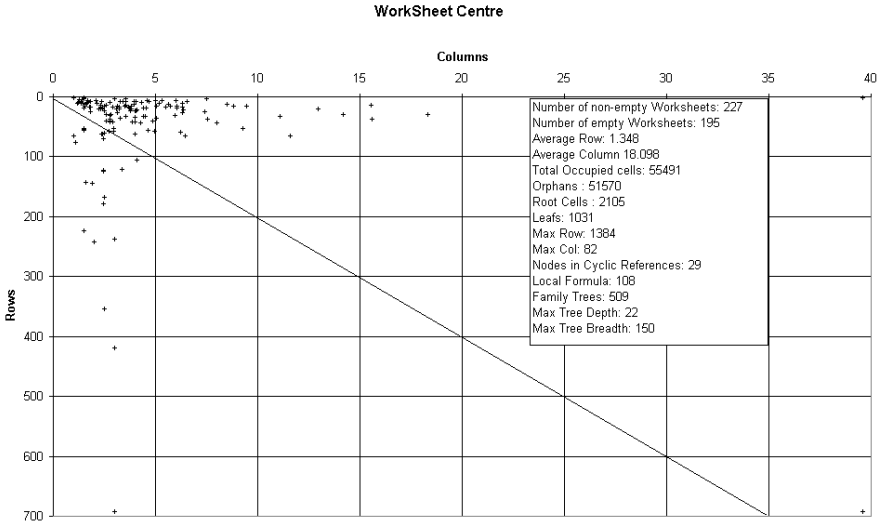


Figure 4.19: Worksheet centre for a corpus of 259 workbooks.

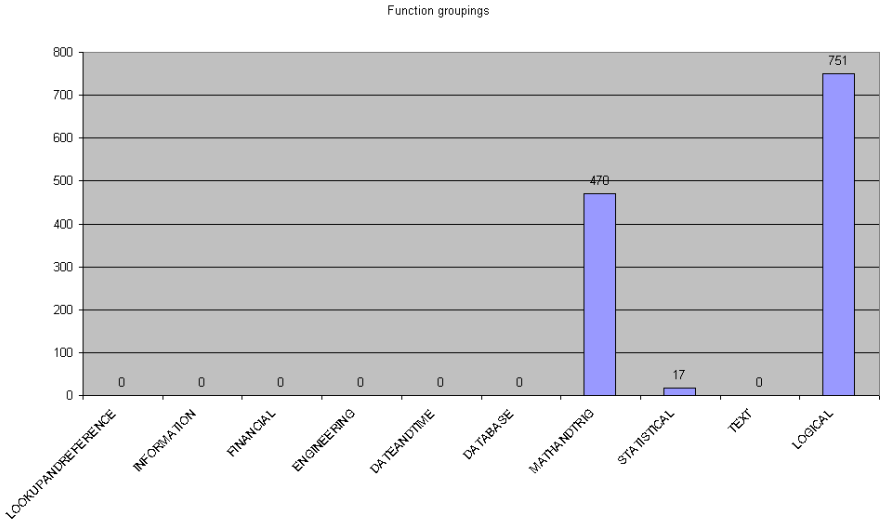


Figure 4.20: Function utilisation in a corpus of 259 spreadsheets.

function is utilised in the corpus. Indications 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.

4.4.7 Future Visualisation Potential

While the above visualisations can provide insight into the low-level structure of spreadsheets and how they are utilised by end-users, they only begin to address the full potential for visualisation in this research area. Much of the data structures for the following visualisations is already provided by the toolkit, and a considerable amount more could be provided with only minor modifications. Outlined below are ideas for visualisations that could be produced in the future.

An interesting visualisation would be a combination of both the real estate and formula information for an individual worksheet. This new visualisation would give a broad overview of each cell's function in the dataflow graph relative to its spatial position. Cells would be classified and coloured depending on their purpose in this graph. Example graph functions could include: ? Root cells, which are referenced by other cells but reference no cells themselves. ? Intermediate cells, which both reference other cells and are referenced. ? Leaf cells, which reference other cells but are not referenced. ? Orphan cells, which have no incoming or outgoing references. ? Pure calculation cells, which contain a formula but make no reference to other cells. ? Translation or mirroring cells, which contain only a single reference to a single cell, i.e. "=B3".

Mirroring cells seem to serve no real computational benefit, but merely transfer a value to a more convenient location for the programmer. When a larger group of cells performs such a transfer of data it is mirroring larger ranges. Other then for presentation purposes, this may be done to bring variables or constants within the scope of the visible screen area.

Another possible visualisation would create a transient (animated) global view. In creating this view several steps would be undertaken. Firstly, all the root/source cells are found. Then a sequence of frames is then generated showing the addition of dependencies at singe level steps. All prior arrows could be included in subsequent frames to build the entire tree, or alternatively removed to give a moving view of the flow.

Several existing visualisations address areas of the spatial relationships between cells. These could be expanded and new visualisations created to consider in more depth the distance of dependencies. In particular, it is important to understand how far these dependencies reach, if there is a relationship between the distance of the dependency and the number of cells referenced, and if there is a significant difference between the horizontal and vertical components. Investigating these characteristics relates to considering the spatial and logical relationships between cells in a combined view.

Another area could include looking for a relationship between a cell's value and its position (either as the spatial coordinate or displacement from the origin). These visualisations could also be expanded to consider data from the corpus, such as aggregating the numerical values in cell coordinates to investigate where the values of highest magnitude are in a spreadsheet.

The total number of logical dependencies created by the formula for a single cell can be classified as referring to one, few, or many other cells. Range references present in the

formula of a single cell allow it to reference a significant number of cells via the lightweight syntax, resulting in dependency trees with the potential for great breadth. When combined with common functions such as sum, there is the potential for large clusters of data to be referenced by very few cells. In addition to considering the breadth of the dependency trees it is also sensible to investigate the tree depth.

One interesting result of the logical dependencies created by formula is that they are typically acyclic, resulting in a tree structure. When a user does create a circular reference, Excel will alert them to the fact using the built in auditing tools to create trace arrows. It is possible to switch Excel into an iterative calculation mode that allows circular references to be created. In this mode, Excel will cease calculations either after a specific number of cycles or the new value calculated comes within a fixed value of the previous value. The toolkit is currently capable of detecting cyclic references and extracting all the cells that exist in the path. Visualisations could be created to address characteristics like the spatial dimensions of the cycle, the number of cells involved in each cycle, and the functions involved in the cycle.

As detailed in the background chapter, Clermont observed that users would sometimes perform a quick fix when using the replication function by entering correct values over the formula for those cells that produced erroneous values. This allowed the user to avoid debugging the formula and to finish the spreadsheet with less effort, but creates a potential source of errors for later modifications. A visualisation could be created to specifically address this issue by helping the user detect one off values within larger blocks of formula.

Rendering visualisations directly over the image of a blank spreadsheet, or possibly even the source spreadsheet, could make the relationship more apparent between Excel and visualisations that display relatively low-level data.

Chapter 5

Conclusions

This project involved visualising low-level structures in individual spreadsheets and data from corpus analysis.

In chapter 2, the background details such as what spreadsheets are, how they work, and the related research are covered. Chapter 3 presents the toolkit developed for this project, outlines how it was constructed, and the purpose of each application. Chapter 4 contains a brief background on information visualisation, the spreadsheet model used as a basis for the visualisations, and presents example visualisations that can be created using the toolkit for both individual spreadsheets and corpus analysis data.

5.1 Contributions

5.1.1 Designed and implemented a toolkit for low level analysis and visualisation of spreadsheets

Undertaking this project required a system to be designed and programmed that supports the creation of spreadsheet visualisations independently from the applications they were created in. The toolkit supports the full process required to find and collect the spreadsheets that are scattered around the Internet, extract the artefacts of interest and then convert the information collected into visualisations. Significant parts of this work include the internal representation and the associated formula parser, the process of reliably extracting the information from the spreadsheets, the visualisation applications, and the corpus analysis tools.

5.1.2 Developed a principled model of spreadsheet structures

This model describes the low-level structures that are present in the spreadsheet in terms of spatial and logical relationships between cells. The model is used as a basis for constructing the visualisations presented and for reasoning about the observed spreadsheet characteristics. The structure for this model is based on related research undertaken in the field of end-user programming and spreadsheets, as well as observation of applications and low-level structures present.

5.1.3 Demonstrated sample spreadsheet visualisations

Examples are made demonstrating the use of the toolkit to inspect the low-level structures present in a single spreadsheet. These visualisations are in part motivated by an exploration process for end-users to learn more about the layout and dataflow structures in spreadsheets that they are unfamiliar with. The visualisations serve to augment those capabilities already

provided by Excel in helping the user understand the dataflow model for a spreadsheet. Each visualisation is based on the principled underlying model, which aids in discussion of observed characteristics. A second motivation was to inspect the structures and patterns that result when end-users program a spreadsheet.

5.1.4 Demonstrated sample corpus visualisations

Using the automated applications in the toolkit, a corpus of more than 8000 spreadsheets was collected to provide a source of empirical data to answer questions about how spreadsheets are utilised via the inspection of low-level structures.

An advantage of corpora is that when constructed correctly, they are representative of the population of naturally occurring data. Thus the corpus analysis approach can be used as an early research tool that would be applied before more in-depth analysis such as usability testing, interviews or auditing.

5.2 Related work

The following outlines how work in this project expands or complements related work that is covered in more detail in the background chapter.

Panko

Raymond Panko has done in depth studies into the causes of spreadsheet errors. Based on this research, and that of others, he has found alarming error rates that indicate users have difficulty creating reliable models of problems using spreadsheets. He attributes these problems to several areas, such as overconfidence, the unwillingness to test models for errors, and the difficulties caused by characteristics such as the transparent dataflow model. As mentioned above, our project uses corpus analysis to research into many times more spreadsheets than Panko is able to do manually. However, Panko's auditing processes are more detailed and focused on errors than the analysis undertaken in this project.

Igarashi

Takeo Igarashi has researched visualisations that make the usually transparent dataflow structures more apparent to end-users. In addition, he has also created techniques that expand the spreadsheet interface to allow users to express relationships between cells in a more powerful way. There are similarities between the some of the static visualisations he presents and those we have created for displaying the dataflow graph. Igarashi's work relies on a specially built spreadsheet application to integrate the new visualisations into the interface. While this approach allows him to create innovative visualisations, he comments on the need to integrate them into a more realistic spreadsheet in future work. By creating the visualisations independently of the application, our toolkit is more versatile as it is not constrained by the limitations of the application.

Burnett

Margaret Burnett's recent work in visual programming languages has been motivated to examine spreadsheets due to the high number to errors observed. She mainly does this through the Forms/3 application, which is capable of helping the user perform visual testing of models they create. Her research into end-user programming and cognitive dimensions has influenced this project when considering how best to convey information back to the user. Also of relevance is her work on creating a cell-relation graph model, which had an influence on the principled model of spreadsheets that was presented in chapter 4.

Clermont

Like the other researchers in this field, Markus Clermont's research is motivated by the high degree of errors. His research into the differences between traditional and end-user

programming was considered in this project when looking for characteristics to observe. Other relevant material he produced includes the grammar for describing Excel formula syntax, equivalence classes for comparing formula, and the implementation of a tool to perform automated error detection.

The grammar he describes for formula syntax is used as the basis for the creation of a grammar for our toolkits parser. There is the potential in future work to utilise his model of formula equivalence in a metric that could be used to conceptually group cells prior to visualisation.

5.3 Future Work

Through the addition of a principled query language for the corpus it would be possible to create a tool of great value. The current form of the toolkit allows new queries and forms of analysis to be constructed programmatically using the toolkit's internal spreadsheet model and corpus analysis basics. The expansion of the toolkit to include a proper query language would allow queries of interest to be answered/verified with much greater ease. Much of the work in this area could be based on the principles of concordance construction from corpus linguistics.

With or without the aid of a query language, there is still the prospect to harness the full potential of the toolkit in creating new visualisations. Several potential visualisations that have not been possible due to time constraints, and are mentioned at the end of the visualisations chapter, with many only requiring minor modifications of the toolkit to achieve.

The metrics presented along with the spreadsheet model have the potential to be utilised in a great number of the visualisations to create valuable characteristic analysis. These are by no means the only metrics available, with the opportunity for future expansion of the set and integration into query language.

There is also the potential to expand the visualisation applications to integrate with the query language. In the ideal case the query would be proposed and the data formatted correctly for visualisations.

Any new work in query languages or visualisation should be properly evaluated to measure its effectiveness and possible weakness when utilised by the end-user. New visualisations should be accessed using approaches such as that of cognitive dimensions, and both the visualisations and query language need to be assessed via usability evaluation.

All future work should be based on observation and modelling of end-user spreadsheet usage. This will allow us to address the needs of spreadsheet users, and provide insight into better end-user computing.

Bibliography

[1] David L. Cohn Alan G. Yoder. Domain-specific and general-purpose aspects of spreadsheet languages, 2002. Distributed Computing Research Lab, University of Notre Dame. http://www-sal.cs.uiuc.edu/ kamin/dsl/papers/yoder.ps.

[2] alphaworks ibm. Excelaccessor. http://www.alphaworks.ibm.com/ab.nsf/bean/ExcelAccessor.

[3] Nicola Ken Barozzi Andrew C. Oliver. Hssf (horrible spreadsheet format) part of poi, 2002. Apache Software Foundation. http://jakarta.apache.org/poi/hssf/index.html.

[4] David Hemmendinger Anthony Ralston, Edwin D.Relly. *Encyclopedia of Computer Science*, chapter 1. Nature Publishiing Group, 4th edition, 2000.

[5] Aurigin. Themescape. http://www.aurigin.com/aureka.html.

[6] I. Small B. Price, R. Baecker. A principled taxonomy of software visualization. *Visual Languages and Computing.*, 1993.

[7] Margaret Burnett Beckwith, Laura and Curtis Cook. Reasoning about many-to-many requirement relationships in spreadsheets. In *IEEE Symposium on Human-Centric Computing Languages and Environments, Arlington, VA*, Sept 2002. ftp://ftp.cs.orst.edu/pub/burnett/hcc02.gridAssertions.pdf.

[8] Jacques Bertin. *Semiology of Graphics.* The University of Wisconsin Press, 1967.

[9] Jonathan Blackwood. Staroffice suite may be bitter pill for ms to swallow, May 2002. ZDNET, CNET Networks, Inc. http://techupdate.zdnet.com/techupdate/stories/main/0,14179,2865566,00.html.

[10] BNC. British national corpus (bnc), 1991. Oxford University Computing Services. http://www.hcu.ox.ac.uk/BNC/.

[11] Dan Bricklin. website. http://danbricklin.com/.

[12] Margaret M. Burnett. website, 2002. Department of Computer Science, Oregon State University. http://www.cs.orst.edu/ burnett/.

[13] Cacas. Regular expressions for java, October 2001. http://www.cacas.org/java/gnu/regexp/.

[14] Patrick Chanezon. Patch for google apis beta 1 to handle proxy settings, 2002. http://www.chanezon.com/pat/google_proxy_patch.html.

[15] Kamalasen Rajalingham David Chadwick, Brian Knight. Quality control in spreadsheets: A visual approach using color codings to reduce errors in formulae, 2002. Information Integrity Research Centre, SCMS, University of Greenwich. http://www.kamalasen.com/chadwick-00.pdf.

[16] Andreas Eberhart. Rdf crawler readme. http://www.i-u.de/schools/eberhart/rdf/crawler/.

[17] S. Ellershaw and M. Oudshoorn. Program visualization - the state of the art, 1994.

[18] Google. Google. http://www.google.com.

[19] Google. Google web apis, 2002. http://www.google.com/apis/.

[20] Herkimer J. Gottfried and Margaret M. Burnett. Graphical definitions: Making spreadsheets visual through direct manipulation and gestures. In *Visual Languages*, pages 250–257, 1997.

[21] Madhusudhan Govindaraju, Aleksander Slominski, Venkatesh Choppella, Randall Bramley, and Dennis Gannon. Requirements for and evaluation of rmi protocols for scientific computing. Indiana University. http://www.extreme.indiana.edu/xgws/papers/sc00_paper/.

[22] T. R. G. Green and Marian Petre. Usability analysis of visual programming environments: A 'cognitive dimensions' framework. *Journal of Visual Languages and Computing*, 7(2):131–174, 1996.

[23] Bill Hibbard. Visad, java component library. Space Science and Engineering Center - University of Wisconsin - Madison. http://www.ssec.wisc.edu/ billh/visad.html.

[24] IBM. Ibm jinsight, 2002. http://www.research.ibm.com/jinsight/.

[25] Takeo Igarashi, Jock D. Mackinlay, Bay-Wei Chang, and Polle Zellweger. Fluid visualization for spreadsheet structures. In *Visual Languages*, pages 118–125, 1998.

[26] A. Kay. *Computer Software.* Scientific American, Sept 1984. 53-59.

[27] Graeme D. Kennedy. *An introduction to corpus linguistics.* Longman, 1998.

[28] Andy Khan. Excelread - a java api to read excel 97 spreadsheets. http://www.andykhan.com/excelread/index.html.

[29] Tom Krazit. Staroffice set to challenge microsoft's office, May 2002. IDG News Service. http://www.pcworld.com/news/article/0,aid,99643,00.asp.

[30] Susan Lammers. *Programmers at Work.* Tempus Books of Microsoft Press, 1989.

[31] Michele Lanza. Combining metrics and graphs for object oriented reverse engineering., 1999. Diploma thesis, University of Bern.

[32] Stan Liebowitz. *Rethinking the Network Economy.* American Management Association, 1999. Chapter 8 Major Markets-WordProcessors and Spreadsheets. http://www.utdallas.edu/ liebowit/book/sheets/sheet.html.

[33] B. Ren M. Burnett, A. Sheretov and G. Rothermel. Testing homogeneous spreadsheet grids with the "what you see is what you test" methodology. In *IEEE Trans. Software Engineering*, June 2002. 576-594. ftp://ftp.cs.orst.edu/pub/burnett/TSE.gridTesting.preprint.pdf.

[34] Yirsaw Ayalew Markus. Detecting errors in spreadsheets.

[35] Roland Mittermeir Markus Clermont, Christian Hanin. A spreadsheet auditing tool evaluated in an industrial context. In *EuSpRIG 2002 symposium*, 2002. http://www.sysmod.com/eusprig02.htm.

[36] B. MM and M. DW. Visual programming, 1995.

[37] Brad A. Myers. Graphical techniques in a spreadsheet for specifying user interfaces. In *ACM CHI'91 Conference on Human Factors in Computing Systems*, pages 243–249. ACM Press, 1991.

[38] Bonnie A. Nardi. *A Small Matter of Programming: Perspectives on End User Computing.* MIT Press, 1993.

[39] James R. Nardi, Bonnie A. Miller. The spreadsheet interface: A basis for end user programming, 1990. HP Labs. http://www.hpl.hp.com/techreports/90/HPL-90-08.html.

[40] James Noble and Robert Biddle. Visualising 1,051 visual programs — module choice and layout in the nord modular patch language. In *Australian Computer Science Conferance(ACSC 2002)*, Melbourne, Australia, 2001. Australian Computer Society, Inc.

[41] James Noble and Robert Biddle. Program visualisation for visual programs. In John Grundy and Paul Calder, editors, *Third Australasian User Interface Conference (AUIC2002)*, Melbourne, Australia, 2002. ACS.

[42] Raymond R Panko. Spreadsheet research repository, 1997. University of Hawaii. http://panko.cba.hawaii.edu/ssr/.

[43] Raymond R Panko. What we know about spreadsheet errors. In *Journal of End User Computing.*, 1998. http://panko.cba.hawaii.edu/ssr/Mypapers/whatknow.htm.

[44] Chadwick D Rajalingham K and Knight B. Classification of spreadsheet errors. *British Computer Society (BCS) Computer Audit Specialist Group (CASG) Journal*, 2000. Vol 10, No 4 (Autumn 2000), pp5-10. http://www.kamalasen.com/rajalingham-00b.pdf.

[45] Daniel Rentz. Openoffice.org's documentation of the microsoft(r) excel file format. http://sc.openoffice.org/.

[46] Karen J. Rothermel, Curtis R. Cook, Margaret M. Burnett, Justin Schonfeld, T. R. G. Green, and Gregg Rothermel. WYSIWYT testing in the spreadsheet paradigm: an empirical evaluation. In *International Conference on Software Engineering*, pages 230–239, 2000.

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

[48] B. Shneiderman. Direct manipulation: a step beyond programming languages, August 1983. Computer 16(8): 57-69.

[49] B. Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* Addison-Wesley, Reading, MA., 1992.

[50] Tony Sintes. It's excel-lent - read ms excel files with java, June 2001. http://www.javaworld.com/javaworld/javaqa/2001-06/04-qa-0629-excel.html.

[51] Sun. Java 3d(tm) api. Sun Microsystems, Inc. http://java.sun.com/products/java-media/3D/.

[52] S. Tanimoto. Viva: a visual language for image processing. *Journal of Visual Languages, Computing 2(2)*, June 1990. 127-139.

[53] Henrik Frystyk Nielsen Tim Berners-Lee, Roy T. Fielding. Rfc 1945 – hypertext transfer protocol – http/1.0, May 1996. IESG.

[54] Andrew Wilson Tony McEnery. *Corpus Linguistics*. Edinburgh Textbooks, 1996.

[55] W3C. Soap 1.2 working draft, June 2002. http://www.xmlhack.com/read.php?item=1708.

[56] J Walkenbach. Feature-rich spreadsheet improves upon 1-2-3 (software review), March 1988. InfoWorld, March 7, 1988, 61-63. http://www.j-walk.com/ss/about/articles.htm.

[57] Wikipedia. Wikipedia (free encyclopaedia). http://www.wikipedia.com/wiki.phtml?title=Spreadsheet.

[58] N. Wilde. A wysiwyc (what you see is what you compute) spreadsheet. In *IEEE Symp. on Visual Languages, Bergen, Norway, Aug. 24-27, 1993, 72-76.*, 1993.

[59] Alan Zisman. Lotus improv, review by alan zisman, June 1993. First published in Our Computer Player. http://www.zisman.ca/Articles/1993/Improv.html.