Design for conceptual knowledge processing: case studies in applied formal concept analysis

Jon Ducrou
University of Wollongong


This paper is posted at Research Online.
NOTE

This online version of the thesis may have different page formatting and pagination from the paper copy held in the University of Wollongong Library.

UNIVERSITY OF WOLLONGONG

COPYRIGHT WARNING

You may print or download ONE copy of this document for the purpose of your own research or study. The University does not authorise you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site. You are reminded of the following:

Copyright owners are entitled to take legal action against persons who infringe their copyright. A reproduction of material that is protected by copyright may be a copyright infringement. A court may impose penalties and award damages in relation to offences and infringements relating to copyright material. Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.
Design for Conceptual Knowledge Processing: Case Studies in Applied Formal Concept Analysis

A thesis submitted in fulfilment of the requirements for the award of the degree

Doctor of Philosophy

from

University of Wollongong

by

Jon Ducrou

Faculty of Informatics

2007
CERTIFICATION

I, Jon Robert Ducrou, declare that this thesis, submitted in partial fulfilment of the requirements for the award of Doctor of Philosophy, in the Faculty of Informatics, University of Wollongong, is wholly my own work unless otherwise referenced or acknowledged. The document has not been submitted for qualifications at any other academic institution.

Jon Robert Ducrou
December 4, 2007
1. **Introduction** .................................................. 21
   1.1 Approach ................................................ 22
   1.2 Method .................................................. 23
   1.3 Summary of Results .................................... 25
   1.4 Structure of the Thesis ................................ 26

2. **Formal Concept Analysis** ................................... 27
   2.1 Definition ................................................ 27
   2.2 Algorithms for FCA ..................................... 31
      2.2.1 Complete Lattice Computation .................... 31
      2.2.2 Conceptual Neighbourhood Computation .......... 32
      2.2.3 Lattice Diagram Layout .......................... 32
      2.2.4 Conclusion ......................................... 39

3. **Ten Tasks of CKP** ........................................ 45
   3.1 Exploration .............................................. 46
   3.2 Search .................................................. 47
   3.3 Recognition ............................................. 48
   3.4 Identification .......................................... 48
   3.5 Analysis ................................................ 49
   3.6 Investigation .......................................... 50
   3.7 Decision ............................................... 50
   3.8 Improvement ............................................ 51
   3.9 Restructuring .......................................... 51
   3.10 Memorisation .......................................... 52
   3.11 Summary ................................................ 53

4. **MailSleuth** .................................................. 61
   4.1 Domain .................................................. 61
   4.2 Knowledge Storage and Access ......................... 63
      4.2.1 Knowledge Base .................................. 64
      4.2.2 Attribute Hierarchy .............................. 64
      4.2.3 Visualisation ..................................... 66
      4.2.4 Usage of the MailSleuth Model .................. 68
   4.3 Usability Evaluation .................................... 68
      4.3.1 Comparative Functionality Review ............... 69
      4.3.2 User-based Evaluation ............................ 70
      4.3.3 Findings and Actions ............................. 71
      4.3.4 Design Aids for Interpreting Line Diagrams ..... 75
   4.4 Evolution ............................................... 81
4.4.1 MailSleuth’s Evolution ........................................ 81
4.4.2 MailSleuth and the CKP Tasks ............................... 82

5. SurfMachine ..................................................... 87
  5.1 Domain ......................................................... 88
     5.1.1 The Mechanics of Beach Selection in Surfing .......... 88
     5.1.2 Sources of Data ......................................... 90
  5.2 Approach ....................................................... 93
     5.2.1 SurfMachine ............................................. 93
  5.3 Evolution ....................................................... 97
     5.3.1 SurfMachine and the CKP Tasks ......................... 97
     5.3.2 Changes from MailSleuth ................................. 101

6. DSift .............................................................. 103
  6.1 Domain ......................................................... 104
  6.2 Approach ....................................................... 105
     6.2.1 Case Scenario One ...................................... 108
     6.2.2 Case Scenario Two .................................... 111
  6.3 Evolution ....................................................... 113
     6.3.1 DSift and the CKP Tasks ............................... 113
     6.3.2 Changes from MailSleuth and SurfMachine ............. 114

7. ImageSleuth ....................................................... 117
  7.1 Domain ......................................................... 117
  7.2 Approach ....................................................... 118
     7.2.1 Restriction of the attribute set ....................... 118
     7.2.2 Moving to upper and lower neighbours ............... 121
     7.2.3 Search and Query-By-Example .......................... 122
     7.2.4 Similarity .............................................. 122
  7.3 Evolution ....................................................... 128
     7.3.1 ImageSleuth and the CKP Tasks ....................... 129
     7.3.2 Changes from MailSleuth, SurfMachine and DSift .... 130

8. SearchSleuth ...................................................... 133
  8.1 Domain ......................................................... 133
  8.2 Approach ....................................................... 136
     8.2.1 An Example Interaction ................................. 140
  8.3 Evolution ....................................................... 142
     8.3.1 Changes from MailSleuth, SurfMachine, DSift and ImageSleuth ........................................ 144

9. Conclusion ......................................................... 145
  9.1 Exposure of Conceptual Structures ........................... 145
     9.1.1 Lattice View/Neighbourhood View ...................... 146
     9.1.2 Conceptual Scales ....................................... 146
     9.1.3 Meaningful Concept Realisation .......................... 147
     9.1.4 Limited Attribute Vocabulary ............................ 148
     9.1.5 Intentional Focus ....................................... 148
     9.1.6 Extensional Focus ...................................... 149
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>General methodology of design research. Taken from [VK04] . . 24</td>
</tr>
<tr>
<td>2.1</td>
<td>Lattice diagram of the planets context . . . . . . . . . . . 30</td>
</tr>
<tr>
<td>2.2</td>
<td>Pseudocode for the Objects Intersection algorithm . . . . . . 40</td>
</tr>
<tr>
<td>2.3</td>
<td>Pseudocode for computing Lower Neighbours . . . . . . . . . . 40</td>
</tr>
<tr>
<td>2.4</td>
<td>Vector Layout Example . . . . . . . . . . . . . . . . . . . . . . . 41</td>
</tr>
<tr>
<td>2.5</td>
<td>Padded Vector Layout Example . . . . . . . . . . . . . . . . . . . 41</td>
</tr>
<tr>
<td>2.6</td>
<td>Vector with Layers Layout Example . . . . . . . . . . . . . . . . 42</td>
</tr>
<tr>
<td>2.7</td>
<td>Force Directed Layout Example . . . . . . . . . . . . . . . . . . . 42</td>
</tr>
<tr>
<td>2.8</td>
<td>Heuristic Layout Example . . . . . . . . . . . . . . . . . . . . . . 43</td>
</tr>
<tr>
<td>2.9</td>
<td>Heuristic Vector Layout Example . . . . . . . . . . . . . . . . . . 43</td>
</tr>
<tr>
<td>3.1</td>
<td>Carpineto and Romano’s CREDO Web Search Application . . . . . . 54</td>
</tr>
<tr>
<td>3.2</td>
<td>Cigarran’s JBraindead Web Search Application . . . . . . . . . . 55</td>
</tr>
<tr>
<td>3.3</td>
<td>Becker’s Docco File Search Application . . . . . . . . . . . . . 56</td>
</tr>
<tr>
<td>3.4</td>
<td>Becker’s Elba Analysis Preparation Application . . . . . . . . . 57</td>
</tr>
<tr>
<td>3.5</td>
<td>Becker’s ToscanaJ Analysis Application . . . . . . . . . . . . . 58</td>
</tr>
<tr>
<td>3.6</td>
<td>Becker’s ToscanaJ Analysis Application showing Nesting . . . . . 59</td>
</tr>
<tr>
<td>3.7</td>
<td>Koester’s FooCA Web Search Application . . . . . . . . . . . . . 60</td>
</tr>
<tr>
<td>3.8</td>
<td>Cole, Tilley and Becker’s CASS Application . . . . . . . . . . . 60</td>
</tr>
<tr>
<td>4.1</td>
<td>Example of traditional email categorisation . . . . . . . . . . . 63</td>
</tr>
<tr>
<td>4.2</td>
<td>Complete list of index types in MailSleuth . . . . . . . . . . . 65</td>
</tr>
<tr>
<td>4.3</td>
<td>MailSleuth drill down folder and its produced lattice . . . . . . 67</td>
</tr>
<tr>
<td>4.4</td>
<td>View of arrow widget used in MailSleuth . . . . . . . . . . . . . 70</td>
</tr>
<tr>
<td>4.5</td>
<td>Final version of MailSleuth . . . . . . . . . . . . . . . . . . . . 71</td>
</tr>
<tr>
<td>4.6</td>
<td>August 2003 version of MailSleuth . . . . . . . . . . . . . . . . . 77</td>
</tr>
<tr>
<td>4.7</td>
<td>Example of layer shading algorithm producing confusing results . 78</td>
</tr>
<tr>
<td>4.8</td>
<td>May 2003 version of MailSleuth . . . . . . . . . . . . . . . . . . . 79</td>
</tr>
<tr>
<td>4.9</td>
<td>Screenshot of HierMail . . . . . . . . . . . . . . . . . . . . . . . . 81</td>
</tr>
<tr>
<td>4.10</td>
<td>Comparison of interface evolution in MailSleuth . . . . . . . . . 85</td>
</tr>
<tr>
<td>4.11</td>
<td>Screenshot of MailSleuth showing Analysis . . . . . . . . . . . . 86</td>
</tr>
<tr>
<td>5.1</td>
<td>Waves optimally break along the beach . . . . . . . . . . . . . . . 89</td>
</tr>
<tr>
<td>5.2</td>
<td>Photos of good surfing conditions . . . . . . . . . . . . . . . . . 92</td>
</tr>
<tr>
<td>5.3</td>
<td>Input devices used in SurfMachine (blank) . . . . . . . . . . . . 93</td>
</tr>
<tr>
<td>5.4</td>
<td>Input devices used in SurfMachine (with query) . . . . . . . . . . 93</td>
</tr>
<tr>
<td>5.5</td>
<td>Lattice output of SurfMachine . . . . . . . . . . . . . . . . . . . . 94</td>
</tr>
<tr>
<td>5.6</td>
<td>Example of map system used in SurfMachine . . . . . . . . . . . . 95</td>
</tr>
<tr>
<td>5.7</td>
<td>The complete interface view of SurfMachine . . . . . . . . . . . . 96</td>
</tr>
<tr>
<td>5.8</td>
<td>Block Diagram of SurfMachine’s architecture . . . . . . . . . . . 97</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------------</td>
</tr>
<tr>
<td>5.9</td>
<td>Example of <em>analysis</em> in <strong>SurfMachine</strong></td>
</tr>
<tr>
<td>6.1</td>
<td>Concept lattices showing the different types of numeric scaling</td>
</tr>
<tr>
<td>6.2</td>
<td>Diagram generated by <strong>DSift</strong></td>
</tr>
<tr>
<td>6.3</td>
<td>Case Scenario One: Initial Search</td>
</tr>
<tr>
<td>6.4</td>
<td>Case Scenario One: Second Search</td>
</tr>
<tr>
<td>6.5</td>
<td>Case Scenario Two: Initial Search</td>
</tr>
<tr>
<td>6.6</td>
<td>Case Scenario Two: Second Search</td>
</tr>
<tr>
<td>6.7</td>
<td>Case Scenario Two: Third Search</td>
</tr>
<tr>
<td>7.1</td>
<td>Screenshot of <strong>ImageSleuth</strong> and its conceptual neighbourhood</td>
</tr>
<tr>
<td>7.2</td>
<td>An example of lattice traversal</td>
</tr>
<tr>
<td>7.3</td>
<td>Pseudocode representation of search traversal</td>
</tr>
<tr>
<td>7.4</td>
<td>Results of a concept traversal</td>
</tr>
<tr>
<td>8.1</td>
<td>Diagram demonstrating the <em>sibling concepts</em></td>
</tr>
<tr>
<td>8.2</td>
<td><strong>SearchSleuth</strong>: ‘formal concept analysis’</td>
</tr>
<tr>
<td>8.3</td>
<td><strong>SearchSleuth</strong>: ‘tiger’</td>
</tr>
<tr>
<td>8.4</td>
<td><strong>SearchSleuth</strong>: ‘tiger information’</td>
</tr>
<tr>
<td>8.5</td>
<td><strong>SearchSleuth</strong>: ‘tiger os’</td>
</tr>
<tr>
<td>8.6</td>
<td><strong>SearchSleuth</strong>: ‘tiger cat’</td>
</tr>
<tr>
<td>8.7</td>
<td><strong>SearchSleuth</strong>: ‘tiger feature’</td>
</tr>
<tr>
<td>8.8</td>
<td><strong>SearchSleuth</strong>: ‘tiger woods’</td>
</tr>
<tr>
<td>9.1</td>
<td>Concept lattice corresponding to the context shown in Table 9.1</td>
</tr>
<tr>
<td>9.2</td>
<td>Concept lattice corresponding to the context shown in Table 9.2</td>
</tr>
<tr>
<td>9.3</td>
<td>Combination of lattices from Fig. 9.1 and Fig. 9.2</td>
</tr>
</tbody>
</table>
LIST OF TABLES

2.1 Planets Context .................................................. 28
4.1 CKP Task breakdown for MailSleuth ......................... 62
4.2 Statement ratings from usability testing of MailSleuth ... 74
5.1 CKP Task breakdown for SurfMachine ....................... 87
6.1 CKP Task breakdown for DSift ................................. 103
7.1 CKP Task breakdown for ImageSleuth ...................... 117
8.1 CKP Task breakdown for SearchSleuth ...................... 134
9.1 Breakdown of conceptual structure exposure types ........ 151
9.2 Summary of the primary tasks of CKP ...................... 152
LIST OF ABBREVIATIONS

ATC Access Testing Centre
CASS Conceptual Analysis of Software Structures
CEM Conceptual Email Manager
CIS Conceptual Information System
CKP Conceptual Knowledge Processing
CLK Conceptual Landscapes of Knowledge
CSV Comma Separated Values
DSIFT Dynamic Simple Intuitive FCA Tool
ECA Email Concept Analysis
FCA Formal Concept Analysis
GIS Geographical Information System
LN Lower Neighbour
TOSCANA TOolS for Conceptual ANAlysis
UN Upper Neighbour
ACKNOWLEDGEMENTS

Of course, the most important acknowledgement is to my supervisor and friend, Prof. Peter Eklund. His ability to convey his enthusiasm for the research area made the PhD process exciting rather than tedious.

This thesis would not have been possible without the support of collaborators – from writing papers to designing software together. Dr. Richard Cole and Peter Becker provided entertaining coffee breaks with arguments about the various facets of the latest programming languages. Bastian Wormuth and Björn Vormbrock, travelled from Darmstadt to argue about mathematical foundations, and provided a theoretical landscape for the construction of software. Honours students Shaun Domingo and Timothy Wilson put many more hours than was required into producing and analysing usability tests and surveys.

Other notable influences and supporters include: Dr. Frithjof Dau, always available for coffee and discussion of Peirce; Bjoern Koester whose enthusiasm for a visiting Aussie was greatly appreciated; and Dr. Mark Sifer who always made himself available for coffee. Finally, Dr. Tom Tilley, whose contribution was by far the most important – the project suffix -sleuth.

The support of my parents, Bob and Debra Ducrou, made this whole thesis possible. Dad with exasperatingly pedantic spelling and grammar checks and Mum with unconditional support.

Most importantly, I have to acknowledge for the infinite love and support of Amanda Ryan. Without you, I would never have made it this far in life.
‘The purpose of computing is insight, not numbers.’

Mathematician Richard Hamming (1915-1998)
Conceptual Knowledge Processing (CKP) is a knowledge management and data analysis technique that makes use of conceptual structures. Formal Concept Analysis (FCA) is a CKP methodology that uses lattice theory to represent units of thought, or concepts. When FCA is used in software applications, it makes use of a process called Mixed Initiative. Mixed Initiative breaks down the roles of user and machine, allowing each to play to their strengths. This process allows the computer, which can process vast amounts of data, to produce interaction options from which the user can select. A human can interpret semantic knowledge contained within the data that a computer cannot. This synergy of user and computer allows complex tasks to be performed. Wille [Wil99] proposed ten atomic tasks of CKP which are combined to make these more complex tasks. The ten tasks are exploration, search, recognition, identification, analysis, investigation, decision, improvement, restructuring and memorisation. Individually, these tasks represent facets of interaction with conceptual systems.

This thesis uses the ten tasks of Conceptual Knowledge Processing as a framework for experimentation with applications that use Formal Concept Analysis. The applications used for this analysis are MailSleuth, SurfMachine, DSift, ImageSleuth and SearchSleuth. These applications approach various problems, using FCA as the primary knowledge structure and interaction framework. Each application uses various interface components and varying degrees and types of exposure to the FCA structures on which they are based. The connection between CKP tasks and interface exposure is then explored and reported.
1. INTRODUCTION

Formal Concept Analysis (FCA) is an unsupervised learning technique which forms structures from data based on the creation of natural clusters of objects and attributes. The purpose of building FCA structures is to allow insight into data collections and provide a framework to enact data mining operations against collections. By allowing control over the creation of these structures, specific facets of the collection can be given emphasis for analysis.

Formal Concept Analysis precedes the digital age and was originally conceived as a human-driven, manual process. Software applications have made it possible to perform the labourious calculation tasks quickly and precisely. Automation can only go so far, and recognition of important facts from the collection cannot always be performed by computer. So, the structures derived from these calculations are used to yield visualisations and representations which human users can interact with to reveal information within the data. This process is called mixed initiative.

Mixed initiative [Hor99] is a process in human-computer interaction involving humans and machines sharing tasks best suited to their individual abilities. In short, the computer performs computationally intensive tasks and prompts human clients to intervene when the machine is unsuited or resource limitations demand human intervention.

The interaction of the user with Conceptual Knowledge Processing software allows various tasks to be performed over the data. Tasks can be considered as the ‘human’ portion of mixed initiative in practice.
1.1 Approach

The design of Information Systems which use Formal Concept Analysis as the principal data analysis technique has never been fully studied. Eklund et al. [EGSW00] presented a developmental method centered on the idea of a Toscana-system but this approach is largely software specific to the application of the Toscana FCA-toolkit. In other work, Hereth and Becker [BH04] introduced the term Conceptual Information System in an attempt to further generalize the systems development idea to broadly embrace other toolkits, in particular Cernato and ConExp.

Wille [Wil05] introduced the idea of Conceptual Knowledge Processing to describe a range of ideas embracing the construction and manipulation of contexts, conceptual scaling, object zooming, nested line diagrams and extensions to FCA that are not tied to software frameworks or toolkits. Wille’s CKP ideas also enable a characterization of relational approaches in FCA that include Power Context Families and connections (from FCA) to more expressive knowledge representation frameworks involving relations and predicates. However, the CKP framework is largely a catalogue of technical mathematical methods derived from lattice and order theory which is frequently applied in the FCA approach. These technical methods are not human-centred or task-based but rather mathematically sound recipes for handling different data structures and types.

The starting point for this research was to consider a task-based categorization of FCA, again introduced by Wille [Wil99]. Wille’s ten tasks of Conceptual Knowledge Processing represent possible actions permitted by the application of conceptual structures to knowledge management situations. There are tasks which involve finding information, such as exploration, identification and search, which all fall under the umbrella of navigation tasks, tasks which involve understanding information, such as recognition, analysis and investigation, and tasks of perception such as memorisation and decision. The last two tasks are
improvement and restructuring which involve alteration of the knowledge.

The ten tasks represent different possible interactions derived from the conceptual structures. The approach followed in this work is to develop Information Systems via case-studies based on FCA that can be used to satisfy all of Wille’s ten tasks, and then to ask what, if anything, can be learned about the dimensions of systems development from the case studies. The overall hypothesis is that there are emergent design patterns that can be applied many times and that these patterns form a basis for both technical CKP methods and task-based approaches to CKP development.

The applications presented in this thesis were developed to address a variety of different knowledge management and discovery problems. Engineered over a four year period and presented in chronological order, these applications all use Formal Concept Analysis. MailSleuth applies FCA in several ways to electronic mail collections. SurfMachine combines geographic data with domain knowledge to aid in finding the optimal location for the sport of surfing. DSift allows any simple database to be accessed and analysed with standard FCA techniques. ImageSleuth provides a navigation paradigm for browsing and searching annotated image collections. Finally, SearchSleuth applies FCA to refinement of web search queries. These applications have different domains with varying aims.

1.2 Method

Design Research, which is frequently deployed in Information Systems and Computer Science, best describes the method of this thesis. The FCA applications developed in the thesis help explain and improve a characterisation of the behaviour and engineering of CKP systems, as will be shown.

In Information Systems (IS) research, a recognized drift away from design research to management and organizational IS issues has occurred, but there is increasing support for the rediscovery of design research as a legitimate IS method [OI01]. In Computer Science research, the systems-based, developmen-
1.2. METHOD

A methodological approach to research – which is both positivist and interpretative – has remained the standard approach. This thesis has elements of both Computer Science and Information Systems research and can be considered as a contribution to both disciplines.

Design research, as presented by Vaishnavi and Kuechler [VK04], follows a pattern of phases. These phases are: Awareness, Suggestion, Development, Evaluation, and Conclusion. Each of these phases has an output, which respectively are: a Proposal, Tentative Design, Artifact, Performance Measures, and Results. A flowchart of the phases is shown in Fig. 1.1.

![Flowchart of Design Research Phases](Fig. 1.1: General methodology of design research. Taken from [VK04])

Awareness of a problem is the identification of an area of research or problem that is lacking a solution, much like any research method. Once found, a proposal for a new research effort is made. The proposal includes not only the research outline, but also how the resulting design Artifact can be evaluated. This leads into Suggestion which is the process of gathering information and ideas toward a tentative design. The Tentative Design and Proposal are tightly
coupled, as the Tentative Design is targeted to that specific to the research effort.

From the Tentative Design, a Development process can begin with the output being a prototype or Artifact. The Design process itself may involve complex problem-solving, or be a combination of existing technologies – it is the design which represents the research effort, not the prototype.

With a functional Artifact, the design can now be Evaluated. Evaluation uses the Artifact to test the design against performance measures defined in the Proposal. Evaluation will yield conclusions, or become the basis for a new design within the same research effort. The conclusion signifies the final process of Design Methodology. At this point the efforts of the research effort are compiled, analysed and typically published.

Each of the applications analysed in this thesis started as a Proposal to address a potential research effort. Some of the presented applications failed evaluation and underwent adjustments to the Proposal, and in turn adjustments to their Design.

1.3 Summary of Results

The relationship between CKP tasks and conceptual structure exposure form the primary results of this thesis, and are revealed in part by each of the five applications presented.

The focus of MailSleuth’s rich interface is on the tasks of navigation and understanding, allowing user-built structures, visualisations and meaningful realisation of collection objects. However, this is found to come at the cost of usability.

Following on from the findings of MailSleuth, ease of use concerns are addressed in SurfMachine. This is done by limiting the number of enabled tasks, but this method of object realisation is found to have an interesting advantage by allowing tasks of perception.

In the next application, DSift, tasks are focussed on understanding relation-
ships in the collection. DSift’s interface allows the user to control the creation of visualisations, and concentrates on the attributes describing the collection more than the objects of the collection.

ImageSleuth sees a departure from standard FCA visualisation techniques, and embeds the user within the structure created. This removes the holistic nature of the previous applications, but allows for better access to navigation tasks and removes the need for the user to construct diagrams.

The final application presented is SearchSleuth. Using a similar display paradigm to ImageSleuth, SearchSleuth allows tasks of alteration of the information on which the interface is based.

A number of conceptual structure exposure types emerge through these applications. In the overall design patterns, the emergent exposure types – which are Conceptual Knowledge Processing software modules – ascribe similarities and differences between the ten CKP tasks. The relationships between exposure types and CKP tasks are the main results from this thesis and can be used to aid future CKP development.

1.4 Structure of the Thesis

This thesis is a monograph of the published works on the applications listed. Each application was developed and published separately. A complete listing of source publications can be found in Appendix A.

Chapter 2 introduces Formal Concept Analysis and discusses algorithms for computing concepts and lattices. The ten tasks of Conceptual Knowledge Processing are then introduced in Chapter 3. The applications are presented individually in Chapters 4 to 8.
2. FORMAL CONCEPT ANALYSIS

Formal Concept Analysis (FCA) is a technique for data analysis, knowledge representation and information management proposed by Rudolf Wille [Wil82] and formalised by Wille and Bernhard Ganter [WG99]. In the last decade FCA has grown in popularity as a method for analysis and representation. With this surge in popularity, FCA-based software tools, which lie on the continuum between general framework for FCA and purpose-built domain-specific application, are needed. This chapter will introduce the formal background of Formal Concept Analysis and present algorithms for programmatic creation of concepts, lattices and diagrams.

2.1 Definition

Philosophically, a concept is a unit of human thought. A concept can be determined through all attributes which describe the concept (the intent), or all objects which are members of the concept (the extent). Representation of a real world unit of thought is often difficult because of the numerous objects and attributes used to define real world concepts, so Formal Concept Analysis typically works within a context which has fixed objects and attributes.

For example, with the formal context of ‘Cartoons’, the formal concept of ‘Dog’ might have ‘Four Legs’ in its intent, and ‘Snoopy’ in its extent.

A more thorough formal context is presented below in Table 2.1. It is the planets example of Davey and Priestley [DP02]. Objects of the context are the planets of the solar system, and attributes related to planet size, distance from

---

1 Derived from a tutorial by Rudolph Wille
2 Prior to the August 24, 2006 redefinition of what constitutes a planet.
2.1. DEFINITION

2. FORMAL CONCEPT ANALYSIS

<table>
<thead>
<tr>
<th></th>
<th>size</th>
<th>medium</th>
<th>large</th>
<th>distance from sun</th>
<th>moon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mercury</td>
<td>×</td>
<td></td>
<td></td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Venus</td>
<td>×</td>
<td></td>
<td></td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Earth</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Mars</td>
<td>×</td>
<td></td>
<td></td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Jupiter</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Saturn</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Uranus</td>
<td>×</td>
<td></td>
<td></td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Neptune</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Pluto</td>
<td>×</td>
<td></td>
<td></td>
<td>×</td>
<td>×</td>
</tr>
</tbody>
</table>

Tab. 2.1: Planets Context

the sun and the presence of a moon.

This context is shown as a cross-table, with rows representing objects and columns representing attributes. An × in a row, i, and column, j, indicates that the object in i has the attribute in j, or conversely that the attribute in j describes the object in i.

A concept of this context can be derived by taking an object and collecting all attributes describing it, then collecting all objects described by those attributes. For instance, taking the planet Mars, let B be its attributes.

\[ B = \{ \text{size} - \text{small}, \text{distance} - \text{near}, \text{moon} - \text{yes} \} \]

Now, let A be the objects which have all attributes in B.

\[ A = \{ \text{Earth}, \text{Mars} \} \]

\((A,B)\) is a formal concept of the context. Any object or set of objects can be used in this way to derive a concept. Dually, concepts can be derived from attributes or attribute sets.

Formal concepts of a context have an ordering. A formal concept, \((A_1,B_1)\) is less general than another concept, \((A_2,B_2)\), if the extent the first concept is contained by the second. Thus an order relation over formal concepts can be defined:
2. FORMAL CONCEPT ANALYSIS

2.1. DEFINITION

\((A_1, B_1) \leq (A_2, B_2) \iff A_1 \subseteq A_2\)

This is equivalent to:

\((A_1, B_1) \leq (A_2, B_2) \iff B_2 \subseteq B_1\)

The formal context can be defined as \((G, M, I)\) where \(G\) and \(M\) are sets and \(I \subseteq G \times M\), where \(G\) are objects\(^3\) and \(M\) are attributes\(^4\).

For \(A \subseteq G\) and \(B \subseteq M\):

\[A' = \{m \in M | (\forall g \in A) g I m\}\]

\[B' = \{g \in G | (\forall m \in B) g I m\}\]

The set \(A'\) is the set of attributes common to all objects in \(A\), and conversely \(B'\) is the set of objects common to all attributes in \(B\). A concept of \((G, M, I)\) is a pair \((A, B)\) where \(A \subseteq G\), \(B \subseteq M\), \(A' = B\) and \(A = B'\).

The complete set of concepts of a context \((G, M, I)\) is denoted \(\mathcal{B}(G, M, I)\). Combining the complete set of concepts and the fact that there is an ordering over concepts means a complete lattice, or concept lattice, can be constructed\(^5\). Concept lattices can be visualised as a stylised form of Hasse diagram called a line diagram. The concept lattice of the planets example from Table 2.1 is shown in Fig. 2.1

The labelling on the lattice diagram shows objects directly below and attributes directly above a concept. Each object and attribute is shown only once, with the labels on concepts representing the contingent. The object contingent of a concept \((A, B)\) is the set of objects which do not appear in any concepts \(< (A, B)\). Given a concept \((A, B)\) and the set of concepts \(< (A, B)\), \((C_1, D_1)...(C_n, D_n)\), the object contingent of \((A, B)\) is defined as \(A - \bigcup_{i=1}^{n} C_i\).

---

\(^3\) G comes from the German word Gegenstände

\(^4\) M comes from the German word Merkmale

\(^5\) See [WG99] for proof.
2.1. DEFINITION 2. FORMAL CONCEPT ANALYSIS

Fig. 2.1: Lattice diagram of the planets context shown in Table 2.1

Dually, a concept’s attribute contingent can be calculated in much the same way with concepts > the concept.

To read the intent of a concept, all attribute labels appearing on concepts above (without any downward connections) are in the intent. For example, the concept in Fig. 2.1 with the label ‘Earth Mars’ has the intent \{distance-near, size-small, moon-yes\}. The extent of a concept is derived by reading down the diagram. For example, the concept in Fig. 2.1 with the label ‘size-small’ has the extent \{Mercury, Venus, Earth, Mars, Pluto\}.

This would mean that the unlabelled concept in the center of Fig. 2.1 has the extent \{Earth, Mars, Pluto\} and intent \{size-small, moon-yes\}. This is the concept of small planets with moons.

A concept \((A_1, B_1)\) is said to be the upper neighbour (or cover) of a concept \((A_2, B_2)\) iff we have \((A_1, B_1) > (A_2, B_2)\), and there is no concept \((A_3, B_3)\) with \((A_1, B_1) > (A_3, B_3) > (A_2, B_2)\).

A concept \((A_1, B_1)\) is said to be the lower neighbour of (or covered by) a concept \((A_2, B_2)\) iff we have \((A_1, B_1) < (A_2, B_2)\), and there is no concept \((A_3, B_3)\) with \((A_1, B_1) < (A_3, B_3) < (A_2, B_2)\).
Upper and lower neighbours of a concept \( C \) are written as \( UN(C) \) and \( LN(C) \) respectively in this thesis.

The concept labelled ‘distance-far’ in Fig. 2.1 has one upper neighbour and three lower neighbours. Moving from a concept to its upper neighbour is a generalisation step. Moving up from the concept labelled ‘distance-far’ reduces the intent by removing the attribute distance-far and increases the extent by adding the objects Earth and Mars. Moving from a concept to its lower neighbour is a specialisation step. Moving down from the concept labelled ‘distance-far’ to the concept labelled ‘Pluto’ increases the intent by adding the attribute size-small and decreases the extent by removing the objects Neptune, Uranus, Saturn and Jupiter.

### 2.2 Algorithms for FCA

The algorithms discussed in the section are for the creation of concepts, concept lattices and lattice diagram layout, and are presented without reference to a particular programming language. There are many different algorithms available; these represent algorithms used in projects discussed in this thesis.

#### 2.2.1 Complete Lattice Computation

The Object Intersections algorithm is possibly the most common algorithm for computing a complete lattice. It takes advantage of the observation that every concept’s extent is the intersection of attribute concepts’ (for an attribute \( m \) the attribute concept is \( (m', m'') \)) extents and every concept’s intent is the intersection of object concepts’ (for an object \( g \) the object concept is \( (g'', g') \)) intents. This approach finds the intersection of each attribute concepts’ extent with the extent of all previously generated concepts, and generates the concept for that intent. If the concept is new it is added to the lattice. Fig. 2.2 shows pseudocode of the algorithm.

The complexity of the Objects Intersections algorithm is \( O(|G||C||M|) \). This algorithm is well suited to creation of diagrams because it supports the iterative
construction of the lattice.

2.2.2 Conceptual Neighbourhood Computation

In some of the chapters to follow, the complete lattice is never computed – only the neighbourhood of a given concept. The lower neighbours are computed by creating concepts which are generated by adding the concept’s intent to each attribute not in the concept’s intent. Finding upper neighbours uses a similar approach with objects. Fig. 2.3 shows pseudocode for computing the lower neighbours of a concept.

The dual of this algorithm computes upper neighbours. Lower Neighbours has a complexity of $O(|G||M|^2)$, while Upper Neighbours has a complexity of $O(|G|^2|M|)$.

2.2.3 Lattice Diagram Layout

Lattice layout of Hasse diagrams is a problem without a clear solution. The problem is made worse by the fact that people have differing opinions on what is considered a good lattice diagram. Diagrams are typically projected in two dimensions and conform (for the most part) to the original pen and paper style used by lattice theorists [Bir40].

In Formal Concept Analysis, the purpose of the concept lattice diagram is to convey information to the human reader [CDE06]. A diagram of a concept lattice should convey the following:

- **Intents**: The attributes that are shared by an object or group of objects.
- **Extents**: The objects that are shared by an attribute or group of attributes.
- **Attribute Implications**: Which groups of attributes imply other groups of attributes.
- **Object Implications**: Which groups of objects imply other groups of objects.
• **Partial Implications**: The proportion of objects with attributes \( X \) that additionally have the attributes \( Y \).

Layout research is primarily concerned with optimizing the efficiency with which this information is conveyed. A layout which conveys information more clearly is more efficient. It is usually considered that the more aesthetically appealing a lattice layout is, the more efficient it is at conveying its information content.

**Considerations when Drawing Concept Lattices**

When computing the layout of Hasse diagrams, any algorithm must take into account the restriction on the \( y \)-axis that a concept, \( A \), must not appear above a concept, \( B \), if \( A < B \). If interaction with the lattice is possible, a breach of this restriction is either disallowed or highlighted to the user. Breaking this restriction renders the meaning of the lattice either incorrect or unreadable.

Within a diagram, any concept or edge occupies a region of two dimensional space. If these regions overlap in certain ways the meaning of the lattice can be altered or rendered nonsensical. This can occur when multiple concepts overlap (partially or completely) or edges overlap with concepts. Edges can however cross edges without causing confusion.

There are certain geometric structures which are considered valuable to the readability of a diagram. For instance, the cube structure (as a subcomponent of a larger lattice) indicates that three attribute concepts combine with each other in every possible way. Incomplete cube structures give quick insight into the absence of a combination.

Given the display capabilities of modern computers, the traditional drawing style for concept lattices can be enhanced with colour information and other visual cues. An example of this is the colouring of concepts as shown in Fig. 2.1. The larger the extent of a concept, the deeper the blue. The smaller the extent the more yellow the concept. Blue and yellow are chosen as, when printed in black and white, or viewed by colour blind individuals, the difference is still
distinguishable. This style of colouring is taken from the work of Becker and his application ToscanaJ [BH04]. Other visual cues include concept size, indication of whether the concept is an attribute and/or object concept, and support (in data mining / rule extraction).

Terms Used in Concept Lattice Layout

Meet: The meet of two concepts is the greatest lower bound of the two. Let $A$ and $B$ be concepts of the lattice $L$. An element $C$ of $L$ is the meet of $A$ and $B$ if the following two conditions are met:

1. $C \preceq A$ and $C \preceq B$ (i.e., $C$ is a lower bound of $A$ and $B$); and
2. for any $D \in L$, such that $D \preceq A$ and $D \preceq B$, we have $D \preceq C$ (i.e., $C$ is greater than any other lower bound of $A$ and $B$).

Usually, the meet of $A$ and $B$ is denoted $A \land B$.

Join: The meet of two concepts is the least upper bound of the two. Let $A$ and $B$ be concepts of the lattice $L$. An element $C$ of $L$ is the join of $A$ and $B$ if the following two conditions are met:

1. $A \preceq C$ and $B \preceq C$ (i.e., $C$ is an upper bound of $A$ and $B$); and
2. for any $D \in L$, such that $A \preceq D$ and $B \preceq D$, we have $C \preceq D$ (i.e., $C$ is less than any other upper bound of $A$ and $B$).

Usually, the join of $A$ and $B$ is denoted $A \lor B$.

Meet Irreducible Concept: A meet irreducible concept is not the meet of any other concepts, and is easily identified as a concept which has exactly one upper neighbour. These concepts will always have a non-empty attribute contingent, and are often considered the key structure concepts when computing layout. In Fig. 2.1, all attribute concepts are meet irreducible concepts.
2. FORMAL CONCEPT ANALYSIS 2.2. ALGORITHMS FOR FCA

Up-Set of a Concept: The set of concepts $\geq$ a given concept is called the up-set of that concept. This is can be thought of as all concepts that are ‘above’ a given concept in the lattice. The up-set of a concept, $A$ in a lattice, $L$, is defined as:

$$\uparrow_L (A) := \{ B \in L | B \geq A \}$$

Layer of a Concept: The minimum number of edges from the top-most concept to the given concept is its layer value. In Fig. 2.1, the concept labelled ‘Earth Mars’ has a layer value of three. Layers can also be computed from the bottom-most concept, or as an average of the two for placement purposes.

Unrealised Concept: An unrealised concept is a concept that is inserted into the lattice to aid layout. A concept is said to be unrealised if it has no contingent. Unrealised concepts are usually created by producing a distributive or boolean version of the lattice, using its structure for layout, and then applying labels using the original lattice. Unrealised concepts are often displayed as smaller than other concepts, or as a meet of edges without a concept.

Distributive Lattice: Distributive lattices are lattices in which the joins and meets distribute over each other. For concepts $A$, $B$ and $C$ in a lattice, the following holds:

$$A \land (B \lor C) = (A \land B) \lor (A \land C)$$

Distributive lattices have an advantage over non-distributive lattices, in that they are more regular in structure, and this allows for predictable outcomes when computing layout.

Boolean lattice: A lattice where every concept has a complement. This means that for every concept there is another concept for which the meet and join of the two will give the bottom-most and top-most concept respectively. All boolean
lattices are distributive, but the reverse is not true.

**Artificial emphasis:** Concepts that are placed in such a way as to add emphasis to a subset of the lattice may cause the viewer to miss detail or not appreciate the diagram whole. This is called artificial emphasis. For example, a lattice which contains two implications, one of which is displayed prominently, the other obscured in the layout, may cause viewers of the lattice to overlook or ignore an equally important implication [Bec05b, CDE06].

*Attribute-Additive Vector Approach*

The most common approach to lattice layout is the Attribute-Additive Vector approach, where meet irreducible concepts are assigned vectors and all other concepts are the sum of the vectors in their upset. Vectors are usually statically defined (numerically or as a function) and are assigned to meet irreducible concepts. In the *MailSleuth* application (presented in Chapter 4), the produced vectors are based on the formulas:

\[
\begin{align*}
\text{index} & = 0 & x & = 0 \\
 & & y & = 0 \\
\text{index} & > 0 & x & = 2.5 \left\lceil \text{index} \right\rceil^{-1} \times -1^{\text{index}+1} \\
 & & y & = 1.8 \left\lceil \text{index} \right\rceil^{-1}
\end{align*}
\]

This produces the following vectors:

\[[0, 0], [1, 1], [-1, 1], [2.5, 1.8], [-2.5, 1.8], [6.25, 3.24], [-6.25, 3.24], \text{etc}\]

The layout shown in Fig. 2.4 is computed from the same algorithm. This vector function is used to avoid concept-concept and concept-line overlaps. It is particularly effective on boolean or distributive lattices.

Fig. 2.5 shows the same lattice as shown in Fig. 2.4 embedded within distributive completion of the lattice. Note that the ‘missing’ structure from
Another approach for Attribute-Additive Vector layout is x-Dimensional Additive Layer layout. This restricts the $y$-values of the concepts to their layer. This places concepts with the same distance from the top-most concept at the same vertical position from the top-most concept, and this addresses artificial emphasis problems associated with vertical placement. Fig. 2.6 shows an example of this. This approach can appear cluttered with larger, more distributive lattices.

**Force Directed Layout**

Force directed layout uses spring-like forces on each edge, and electrical forces between concepts, to iteratively form the layout. By allowing edges and concepts to exert forces against each other, a natural state where forces are equally balanced forms the final layout.

Typically this approach uses a combination of Hooke’s law (spring force) and Coulomb’s law (force of attraction between electrically charged particles). Success is usually found when this technique is applied to three dimensional geometric spaces, but good two dimensional results are often achieved. For more information see [Fre04]. An example of Force Directed lattice layout can be seen in Fig. 2.7. Note that layers are not preserved in this layout.

A drawback of this style of lattice is that the results are not predictable; some complex lattices can be resolved to easily readable diagrams, while some simple diagrams can be rendered almost unreadable.

**Optimal Heuristic Layout**

Optimal Heuristic Layout refers to layout algorithms that produce multiple diagrams, assess each of them against various metrics and select the diagram that best optimises the metrics without causing concept/concept or edge/concept overlaps.

Example metrics include (but are not limited to):
2.2. ALGORITHMS FOR FCA 2. FORMAL CONCEPT ANALYSIS

- Total length of edges
- Number of edge crossings
- Number of unique absolute edge gradients
- Percentage of concepts with children placed symmetrically below
- Percentage of concepts with two or more children which have two children placed at $x + 1$ and $x - 1$ where $x$ is the parent’s $x$-value
- Shift between top-most and bottom-most concept’s $x$-values

This technique best suits small lattices, as the overhead of producing many diagrams is prohibitive for large lattices. One way of reducing the search space for these types of lattice is to use a layer restriction on the vertical placement of concepts.

One heuristic layout method is layer-centered permutational heuristic algorithms which arrange each layer of concepts in every possible permutation, and each layer arrangement permuted with each other layer. This may seem computationally expensive, but small lattices (less than 25 concepts) can still be exhaustively tested within a reasonable time-frame (a few seconds). Results typically lack clearly noticeable substructures, but produce very symmetrical, compact diagrams. An example is shown in Fig. 2.8. An example of a layer-centered permutational heuristic algorithm is Cole’s backtracking layout algorithm [Col01b].

Another form of this algorithm is to permute the vectors of meet irreducible concepts at each iteration completing the layout. The computation required for this is dependent on the number of meet irreducible concepts in the lattice and the number of vectors that the algorithm is willing to attempt. The example shown in Fig. 2.9 shows a layout produced by this method. Note the partial preservation of substructures while still remaining symmetrical, a result typical of this algorithm.
Overall, optimal heuristic methods tend to produce better quality layouts for less distributive lattices. They also do not produce overlaps, as every produced layout is tested for these types of faults— even the best single pass layout can usually be forced into an overlap situation.

Another possible advantage in producing multiple layouts is that users are presented with the top scoring layouts for selection, leaving the final decision to the user. This solves the problem of the subjectivity of what is considered a good layout.

Optimal heuristic models incur computational time cost and can sometimes fail to find any layouts that do not have collisions (e.g. when the possible vectors are limited).

With heuristic methods there is much argument on the suitability of metrics, and this area has not been fully researched at this time. Some work on the topic is explored in [CDE06].

2.2.4 Conclusion

The majority of the presented algorithms and techniques are used in the projects presented in this thesis. Efficient computation and effective diagram layout are needed in FCA based applications in order for them to be useful.
See print copy for figure 2.2

Fig. 2.2. Pseudocode for the Objects Intersection algorithm. Sourced from Carpineto and Romano [CR04a]

See print copy for figure 2.3

Fig. 2.3. Pseudocode for computing Lower Neighbours. Sourced from Carpineto and Romano [CR04a]
Fig. 2.4: Vector Layout Example: Note the concept labelled *layout* is positioned lower than other single attribute concepts.

Fig. 2.5: Padded Vector Layout Example: Note the smaller unrealised concepts which provide clues as to the missing combinations of attributes.
Fig. 2.6: Vector with Layers Layout Example. Note that layout is now at the same vertical positioning as other single attribute concepts (unlike in Fig. 2.4).

Fig. 2.7: Force Directed Layout Example. Note that the concepts labelled 1 and 78 appear to be related to other concepts incorrectly because of edge/concept collisions.
Fig. 2.8: Heuristic Layout Example: Note the symmetry of the diagram – at the cost of regular structures. The potential number of diagrams tested to produce this diagram is 5760.

Fig. 2.9: Heuristic Vector Layout Example: Note the presence of partial structures. The potential number of diagrams tested to produce this diagram is 16384.
3. TEN TASKS OF THE CONCEPTUAL LANDSCAPE OF KNOWLEDGE

Wille [Wil99] introduces Conceptual Landscapes of Knowledge as a pragmatic paradigm for knowledge processing. Using the metaphor of landscapes, Wille outlines ten tasks that can be performed with Conceptual Knowledge Processing applications. Knowledge in this instance is understood on the basis of C.S. Peirce’s Pragmatism [Pei35]; the landscape metaphor reinforces Peirce’s notion of practical bearings being a requirement for meaningful statements and knowledge.

“The idea of landscape is becoming increasingly influential in the field of knowledge representation and processing. Especially, the frequently used term of “navigation” suggests how this idea is becoming a leading metaphor. That view also is supported by the actual development of viewing computers as medium. This development shows that it is time for explicating the pragmatic landscape paradigm for knowledge processing.’

Rudolph Wille in [Wil99](emphasis by Wille)

The tasks are, in the order presented by Wille, exploration, search, recognition, identification, analysis, investigation, decision, improvement, restructuring and memorisation. Conceptual Knowledge Processing tasks are not always distinctly identifiable as a single operation, interaction or process, and can sometimes be thought of as representative of more than one task. For instance, exploration and search are very closely related and may follow the same process in
3.1. EXPLORATION

an application to complete. These ten tasks provide a basis for experimentation with Conceptual Knowledge Processing applications.

3.1 Exploration

*Exploration* is the task of looking for something where the understanding of the item is vague or not well known. Within Information Retrieval, the *exploration* task is often referred to as *browsing*.

An example of this is library research, whereby a person has a topic of interest in mind and must explore the shelves which would most likely contain books on the topic. In this instance, the information requirement is general and could potentially be satisfied by any number of books.

Carpineto and Romano’s 1995 software prototype, *Ulysses* [CR95], uses a *search and bound* work flow to allow FCA-based exploration of a document collection. *Ulysses* uses a previously built lattice representation of the collection as a basic framework (called *Galois*). This framework defines a structure and the query terms available for *exploration*.

A *Ulysses* user enters a query term and is presented with a visualised lattice displaying the various conceptual clusters associated with the term. By introducing new query terms and applying hierarchic constraints on the resultant lattice view, the collection can be modified. This allows a user-guided exploration over the data space. *Ulysses* relies heavily on the *Galois* process producing a high quality structure and that the controlled vocabulary it produces being meaningful.

Another example of *exploration* using FCA is the work of Kim and Compton and their prototype *KANavigator* [KC01a, KC01b]. This work uses annotated documents which can then be browsed by keyword. Rather than displaying a visualisation, only the direct neighbourhood (in particular the lower neighbours) is shown to the user. This system emphasises the use of textual labels as opposed to visualised structure to guide exploration.
3. TEN TASKS OF CKP

3.2  Search

To search, there must be a precise idea of what is being searched for. The search task is the localisation of this target within an information space. Within Information Retrieval, the search task is often considered the alternative to browsing (exploration) and is referred to as querying.

To search for a particular quote in a book is a good example. The quote is known to the searcher, but its exact location within the text is unknown. Knowledge such as chapter titles, or information extracted from the index, could be used in finding the quote.

Search using FCA has been applied primarily as an aid to existing Information Retrieval methods, in most cases as a form of local analysis or clustering.

Carpineto and Romano’s application, CREDO [CR04b], uses an iceberg lattice (an incomplete lattice derived using only a collection of the top-most concepts of a complete lattice) to generate clusters for given search terms. Searched terms are submitted to an online search engine and the returned results are entered into a context. The context is built with the results as objects, and the terms found in the result summaries as attributes. The most general concepts of the concept lattice are computed and displayed as a tree. The user can then interact with the tree to view named clusters of the results. CREDO provides an interface that extends the collected result set, and provides a landscape over the information space, which can be used to aid in search retrieval. A screenshot of CREDO is shown in Fig. 3.1.

Cigarran et al [CGPnV04] provide a similar tool called JBraindead which, rather than showing only the top-most concepts, allows browsing to continue to the most specific concepts of the lattice. This is achieved through the use of a novel interface that shows concept sublattices selectively, reducing the complexity of the display. As can be seen in Fig. 3.2, JBrainDead not only uses single terms, but also uses a natural language processing engine to recognise phrases for context creation – a feature which sets it apart from other FCA-based web
3.3. RECOGNITION

search.

Another example of an FCA-based search application is Becker’s *Docco* [Bec05a] application. *Docco* displays a lattice visualising the distribution and overlap of the term words entered into a search field. The context is constructed using documents as objects, and entered terms as attributes. Concepts of the produced concept lattice translate to document clusters that share the terms in the concept’s intent. These clusters can be realised and the files explored via a tree widget. *Docco* enhances the search task by showing the underlying inter-document term relationships. A screenshot of *Docco* is shown in Fig. 3.3.

3.3 Recognition

*Recognition* occurs when a clear correlation of relations and circumstances is perceived in a data collection. In the context of applications this is best understood as the correct perception of the current localisation within the information space; the user can recognise the space as being correct in terms of their expectations.

This task is rarely the focus of an application, but is more a product of a good interface. An appropriately designed interface that enables *Recognition* allows a user to comprehend what is being viewed or used at a level aligned with their own understanding.

3.4 Identification

The task of *identification* is the positioning of objects or concepts in relation to other objects or concepts. An example is data mining, whereby rules, trends or patterns are extracted from raw, messy or noisy data.

Valtchev et al [VMG04] propose that FCA is well suited to this task, and outlines methods for using FCA for Association Rule Mining (ARM) and Knowledge Discovery in Databases (KDD). As concept lattices create structure from data, FCA offers an appropriate methodology for these areas. Concept lattices
3. TEN TASKS OF CKP

by definition contain implications and equivalences as well as other structural information. By using these innate properties with mining algorithms, association rules and other information can be mined from data.

3.5 Analysis

*Analysis* is the task of examining data with a declared purpose or theoretical view to achieve some understanding. *Analysis* can be aided by data representation or controls to allow the underlying data to be viewed in different ways.

An example of analysis is the various *Toscan* applications [VWW95, SWW98, BHS02, BH04, HK04]. *Toscan* (TOolS for Conceptual ANAlysis, and also a reference to Tuscany in Italy\(^1\)) provides a platform for FCA-based analysis of information. *Toscan* has been through many development iterations and has been re-developed as *Toscan*\(^{J}\). *Toscan*\(^{J}\), created by Becker et al [BHS02, BH04], is an open-source Java reimplementation of *Toscan*.

Using a *Toscan* system requires an FCA expert and a domain expert (these may be the same individual) to work together to create scales over the data collection. Fig. 3.4 shows *Elba*; the tool used for creating scales for *Toscan*\(^{J}\). The system then produces diagrams (laid out by the FCA expert at construction time) and fills the extent and intent of the diagrams with results extracted from the data. Fig. 3.5 shows *Toscan*\(^{J}\) displaying the lattices defined in Fig. 3.4. These diagrams can then have operations performed on them such as zooming and nesting. Zooming is the process of restricting a diagram’s extent to the extent of a concept from another diagram. Nesting is a visualisation technique where one diagram is displayed within each concept containing the diagram of another scale (This is shown in Fig. 3.6).

The workflow developed for *Toscan* systems is intended to primarily accommodate analysis, but also to some extent investigation.

\(^{1}\) To quote Wille [Wil99], “…*Tuscany* is viewed as the prototype of a cultural landscape which stimulated many important innovations and discoveries, and is rich in its diversity and attractive for wandering in.”
3.6 Investigation

*Investigation* is the process of revealing truth by observation and interrogation of the information within the landscape.

Yevtushenko’s *ConExp* [Yev] software boasts the largest set of operations defined in Wille and Ganter’s FCA book ([WG99]) [Til04a] and is an example of FCA software that supports *investigation*. While still requiring some level of FCA training to use, *ConExp* can load a context and then provide the user with tools to create lattices from the data and allow modification of the view of the information space. *Investigation* is further supported with many options for extracting knowledge, such as implication extraction and ARM [Stu02]. The most interesting aid to *investigation* is attribute exploration, which provides an iterative questioning of attribute implications and equivalence to allow users to focus on the underlying meaning of the data at a concept level.

3.7 Decision

The *decision* task uses representation of data to aid in the resolution of an uncertain state. This task uses the information landscape to allow a single optimal option to be selected from multiple available options. Often the process of facilitating *decision* is referred to as Decision Support.

A decision could be made via constraint satisfaction, elimination of options or selection from similar options. *Decision* can also be performed by more than one person using the information space as a basis for argument and negotiation.

An example of this is Tilley’s work on representation of the formal specification language, Z, with concept lattices [Til04b]. By augmenting the software specification methodology with FCA, *decision* tasks can be improved. This means the resulting specification is more scrutinised and therefore more likely to be of high quality.

Another interesting application of FCA that aids *decision* is Cole and Becker’s agreement contexts [CB04]. This work showed that two sources of information
about the same topic can be combined into a single context for the purpose of enhanced argumentation.

### 3.8 Improvement

*Improvement* tasks are typically those that increase the value of the landscape, rather than the underlying data. This means that the information landscape is enriched, which in turn offers support for other conceptual tasks. This makes *improvement* a very general task, with goals defined by what is considered an increase in quality.

A simple example of *improvement* is query refinement, whereby correct adjustment to search criteria yields more relevant results. To facilitate *improvement*, the ability to make correct adjustments must be available.

One such application of query refinement with FCA is Koester’s *FooCA* [Koe06, Koe05] web search application. *FooCA* is a local analysis search engine relying on input from traditional Web-based Search Engines. It provides one of the richest interfaces available for the construction of a formal context based on web search results. By taking results as formal objects, and terms found in the results as formal attributes, a formal context is created and presented to the user. This can then be edited directly (or visualised) to aid in the *improvement* of the query terms used, and in turn aid the *search* task itself. A screenshot of *FooCA* is shown in Fig. 3.7.

### 3.9 Restructuring

*Restructuring* tasks directly affect the underlying information by changing its structure or organisation. This process can be used to enhance the quality of the underlying information (similar to *improvement*) or to alter it for another purpose.

The classic example of this is software refactoring – where source or intermediate code, or the syntactic parse tree, is analysed with an aim to improving
code structure. Restructuring of this type may be to improve the source code, or to adapt it to a different environment or use.

Snelting [Sne96] used preprocessor symbols to analyse the configuration structure of C source code, using software examples that run on many different platforms. The generated lattices and inferences yield insight into the complexity of the source code and, in turn, the software itself. It is proposed that this insight could potentially be used to refactor the C source code.

Godin et al [GMM+98] use the hierarchic nature of concept lattices to infer the Object Oriented (OO) class hierarchy. This is supported by a mixed-initiative approach, where system designers can directly alter the specification and view the new OO structure as a visualised lattice. These lattices can have metrics applied to them to rate their structure. This method of active restructuring means the class hierarchy is always void of any redundancy and can be considered consistent with the specification.

Cole, Tilley and Becker [CT03, Til04b, CB05] present a methodology and application for the exploration of software source code based on intermediate Java code. The application, called CASS (Conceptual Analysis of Software Structures), is a plugin for the Eclipse Integrated Development Environment (IDE). A screenshot of CASS is shown in Fig. 3.8. The primary aims of this methodology are the identification of unnecessary dependencies between classes or packages, and potential structural deviations from specification structure. These aims are mostly achieved by giving a big picture of the source code details, rather than the typical approach using global views which have courser granularity of detail. Insights from using CASS aid in restructuring of source code.

3.10 Memorisation

Memorisation is the learning and retaining of knowledge or information. This task is not a typical aim of FCA at this time. Memorisation may not necessarily refer to value–perfect memory of data but a notion of seeing something before,
or remembering a previously drawn conclusion. In some ways, this is like a pie chart; it is easier for a human to memorise the graphical representation than the (possibly many) numeric figures it is based on. Memorisation in this context is a human centric task, and not the ability of a computer to recall information.

3.11 Summary

The ten Conceptual Knowledge Processing tasks listed here are rarely singularly focussed in software or software processes. Some tasks are not the focus of any known FCA-based software; memorisation being the prime example. Other tasks are not a primary aim of the software but occur as a byproduct of the interface; recognition is an example of this. Some applications provide access to tasks in a non-FCA-based manner. Tasks achieved without the use of Formal Concept Analysis techniques are ignored in this thesis.

These ten tasks of Conceptual Knowledge Processing, and the Conceptual Landscapes of Knowledge metaphor, will be used throughout this thesis in an effort to follow and analyse the abilities of FCA-based systems. Each task an application can perform is dependent on the user–interface and level of exposure to, and allowed manipulation of, the underlying knowledge structure. The intended use of the individual application also plays a part in the tasks possible, but only when explicitly made available through the user–interface.
See print copy for figure 3.1

Fig. 3.1: Carpineto and Romano’s CREDO Web Search Application [CR04b]: An example of search. CREDO displays the uppermost concepts derived from keywords featured in search results.
Fig. 3.2: Cigarran’s *JBraindead* Web Search Application [CGPnV04, CPnGV05]: An example of search. *JBraindead* uses natural language processing to recognise phrases within the information space then displays the concept sublattices as results using an innovative interface. Note: This screenshot was taken in June 2007 of an unfinished alpha version.
Fig. 3.3: Becker's Docco File Search Application [Bec05a]: An example of search.
Docco allows users to visualise the inter-document term relationships.
Fig. 3.4: Becker’s Elba Analysis Preparation Application [BHS02, BH04]: Elba is used to prepare a database for analysis by ToscanaJ. The attributes of the diagram are SQL queries to be executed against the database while ToscanaJ is in operation. Elba is used by the domain expert and the FCA expert. **Top:** Bus Type is displayed. **Bottom:** Case Type is displayed. (Note: Elba performs a similar operation for ToscanaJ as Anaconda does for Toscana.)
See print copy for figure 3.5

Fig. 3.5: Becker’s ToscanaJ Analysis Application [BHS02, BH04]: Here it can be seen that the lattices defined in Elba (see Fig. 3.4) have been filled with details sourced from the database. **Top:** Bus Type is displayed. **Bottom:** Case Type is displayed.
Fig. 3.6: Becker's ToscanaJ Analysis Application [BHS02, BH04]: Here it can be seen that the lattices from Fig. 3.5 have been nested within each other. **Top:** Case Type embedded in Bus Type. **Bottom:** Bus Type embedded in Case Type.

See print copy for figure 3.6
Fig. 3.7: Koester’s FooCA Web Search Application [Koe06, Koe05]: An example of improvement. By allowing users to see and access the quality of the various terms in the current search results, successive search results increase in quality.

Fig. 3.8: Cole, Tilley and Becker’s CASS Software Structure Analysis Application [CT03, CB05]: CASS allows visualisation of complex software structures to facilitate restructuring tasks on software during development.
4. **MailSleuth**: PERSONAL CONCEPTUAL INFORMATION SYSTEM FOR EMAIL COLLECTIONS

**MailSleuth** is a personal productivity tool that allows individuals to manage and visualise their email collection using lattice diagrams. By integrating a traditional email browsing paradigm with a structured document management approach, **MailSleuth** extends the standard email management methodology. **MailSleuth** does this by enhancing an existing email application, Microsoft Outlook™, with new features that integrate into an existing mail collection.

This project was the next step in a series of projects; those preceding **MailSleuth** were *CEM, ECA, Warp9* [CE99, CES00, CS00, ECS02, CES03].

The **MailSleuth** software has been extensively assessed for usability. It also provided a platform for the testing of Formal Concept Analysis and concept lattices for practicability in the hands of non-FCA trained users.

This chapter introduces the domain of the project, its implementation and the evolution the software took as part of usability testing. This is followed by analysis of the functionality with regard to Wille’s ten tasks of Conceptual Knowledge Processing (Chapter 3).

### 4.1 Domain

Email documents are well-structured, text-based documents used primarily for inter-personal communication. Email is the primary business communication medium, both within and between companies. Structural components within emails are well defined and typically include information on sender and receiver(s), subject, date and the body, or content, of the email.
Email is typically stored on the user’s computer or on a mail server which the user accesses. Traditionally, the organisation of these documents is not unlike the filing of paper-based mail. One mail item has one place in the filing system. There are differing levels of complexity of email filing methods, from simple flat methods to elaborate inheritance structures.

Simple flat methods have all email collected in a single one dimensional list. This is the simplest of storage and retrieval methods. Usually storage is maintained chronologically, but often tools for finding and sorting emails to assist retrieval are featured. While this method is simple, it does not scale well to large, diverse collections.

Inheritance structures, usually called ‘folder’ structures, allow categorisation of email into single inheritance hierarchies where the root level contains folders and emails, and any folder can contain folders and emails. Typically, the folder depth rarely exceeds one level. An email exists once in a single folder at any level of the hierarchy. Folder structure and email list components are traditionally separated within the display as shown in Fig. 4.1.

Each folder is a named categorisation or subcategorisation, which contains email items. Emails physically exist in a single folder, so if an email could be categorised into two folders, it would need to be placed into only one, or duplicated to be in both. For example, if a user has a category for conferences and another for collaboration with a colleague, then where should email from...
See print copy for figure 4.1

Fig. 4.1: Example of traditional email categorisation. A hierarchic folder list is shown on the left as a tree structure. This is complemented with a one dimensional list of emails, which is shown on the top right. Emails are produced via the preview pane shown on the lower right.

the colleague about a conference go? Placing the mail in one of the categories means finding that email may require looking in both folders. Duplication on the other hand produces redundancy.

A more appropriate methodology should allow emails to be found in multiple locations within the system, without physical duplication. Namely, an email about joint conference submission with Richard should be in both ‘Conference’ and ‘Richard’. Some attributes should be able to be derived from others, for instance ‘SIGIR’\(^1\) should automatically imply ‘Conference’. And ‘Conference’ may in turn imply ‘Research Activities’, etc. The mail in each of these folders should be able to be analysed separately as a subcollection, allowing analysis to be localised within the overall collection. Formal Concept Analysis supports this multiple inheritance hierarchy and forms a theoretical framework for applying this idea.

4.2 Knowledge Storage and Access

For MailSleuth to overlay FCA onto an existing mail system efficiently, appropriate storage and access is required.

\(^1\) SIGIR refers to the conference of the Special Interest Group: Information Retrieval.
4.2. KNOWLEDGE STORAGE AND ACCESS

4.2.1 Knowledge Base

MailSleuth uses a knowledge base to store all information relating to its operation and state. This includes a full text index, the attribute hierarchy, user settings and all other information used by the software. The knowledge base is, in essence, a triple store which automatically applies transitive closure operations on new triples with certain relationship types (e.g. asserting ‘a is-a b’ and ‘b is-a c’ will automatically assert ‘a is-a c’). This design allows partial order operations (frequently occurring in Formal Concept Analysis applications) to be performed automatically and efficiently.

An efficient knowledge base allows MailSleuth to create a structured, full text index of the user’s email collection. This can be thought of as an all encompassing formal context for the MailSleuth system. While the context is maintained, it is never used in its entirety for the creation of a lattice structure (given the millions of relations created in an email collection, such a task is computationally prohibitive). Each email is broken into its component parts (See Fig. 4.2) and indexed based on these parts. Indexing is based on one of four relation types: ‘index relation’, ‘before relation’, ‘after relation’ and ‘relative relation’. These types dictate the behaviour of the query engine in returning results from the index. Components that are primarily natural language based, such as the email body and subject, are processed using standard stopword and stemming methods.

4.2.2 Attribute Hierarchy

To provide the user with access to multiple inheritance functionality, the software allows creation and modification of an attribute hierarchy. The simplest way to represent this without deviating unnecessarily from standard user control types is with a folder tree that enforces a partial order over its structure. This means that some branches of the folder tree will be duplicates of others – but their content is not duplicated. Each folder’s name is a unique identifier.
### Term Type

<table>
<thead>
<tr>
<th>Term Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email-Body</td>
<td>Searches Email Body</td>
</tr>
<tr>
<td>Email-Subject</td>
<td>Searches Email Subject</td>
</tr>
<tr>
<td>Attachment-Contains</td>
<td>Searches Email Attachments</td>
</tr>
<tr>
<td>Attachment-Type</td>
<td>Searches Email Attachment Extension</td>
</tr>
<tr>
<td>Attachment-Size-More-Than</td>
<td>Restricts results to Mail with total attachment size greater than the given value (in Bytes)</td>
</tr>
<tr>
<td>Attachment-Size-Less-Than</td>
<td>Restricts results to Mail with total attachment size less than the given value (in Bytes)</td>
</tr>
<tr>
<td>Attachment-Present</td>
<td>Restricts results to Mail with one or more attachments</td>
</tr>
<tr>
<td>Attachment-Filename</td>
<td>Searches Email Attachment Filename</td>
</tr>
<tr>
<td>Flag-Status</td>
<td>Restricts to One Flag Status</td>
</tr>
<tr>
<td>Arrived-Relative-To-Today</td>
<td>Restricts Results to Mail which Arrived within the last Week or Day</td>
</tr>
<tr>
<td>To-Email-Address</td>
<td>Searches TO field for an Email Address</td>
</tr>
<tr>
<td>CC-Display-Name</td>
<td>Searches CC field. Can be address or display name</td>
</tr>
<tr>
<td>BCC-Display-Name</td>
<td>Searches BCC field. Can be address or display name.</td>
</tr>
<tr>
<td>From-Email-Address</td>
<td>Searches FROM field for an Email Address</td>
</tr>
<tr>
<td>To-Email-Address-User</td>
<td>Searches TO field for the Prefix (Username) of an Email Address</td>
</tr>
<tr>
<td>From-Email-Address-User</td>
<td>Searches FROM field for the Prefix (Username) of an Email Address</td>
</tr>
<tr>
<td>To-Email-Address-Domain</td>
<td>Searches TO field for the Suffix (Server) of an Email Address</td>
</tr>
<tr>
<td>From-Email-Address-Domain</td>
<td>Searches FROM field for the Suffix (Server) of an Email Address</td>
</tr>
<tr>
<td>From-Display-Name</td>
<td>Searches FROM field for Display name associated with an Email Address</td>
</tr>
<tr>
<td>To-Display-Name</td>
<td>Searches TO field for Display name associated with an Email Address</td>
</tr>
<tr>
<td>Restrict-Query-to-Subfolder</td>
<td>Restricts results to a given Subfolder.</td>
</tr>
<tr>
<td>Restrict-Query-to-Folder</td>
<td>Restricts results to a given Top-Level Folder.</td>
</tr>
<tr>
<td>Arrived-On-Date</td>
<td>Restricts results to Mail which arrived on a given date</td>
</tr>
<tr>
<td>Arrived-Before-Date</td>
<td>Restricts results to Mail which arrived before a given date</td>
</tr>
<tr>
<td>Arrived-After-Date</td>
<td>Restricts results to Mail which arrived after a given date</td>
</tr>
</tbody>
</table>

---

**Fig. 4.2:** Email components used in indexing for MailSleuth. Some components, such as ‘Restrict-Query-to-Folder’ and ‘Flag-Status’, are derived from the Outlook™ environment MailSleuth is embedded within and not from the email’s content.
which is representative of its content and the content of all child folders. If a
folder is created with an existing name that does not break the partial order,
it will automatically have all child folders associated with that name created
under it. This style integrates well with the host software (Outlook™) which
uses a folder view for traditional email management.

The attribute hierarchy is comprised of labels. Each label is part of a partial-
order. For the set $M$ of labels, the following compatibility condition holds:

$$\forall g \in G, m, n \in M : (g, m) \in I, m \leq n \Rightarrow (g, n) \in I$$

The attribute hierarchy is the mechanism that allows the user to create concep-
tual scales for viewing lattices. It is strictly enforced. Users creating, moving
or deleting folders are warned if the action will cause repercussions on other
attributes in the partial-order.

Labels can have one or more queries attached to them. A label’s queries
generate objects which are associated with that label. When more than one
query is used, the result is the intersection of each query result. Each label
is a ‘virtual folder’ or ‘drill-down folder’. ‘Virtual folders’ display the email
matching the query associated with the label in the traditional one dimensional
list, which is kept consistent with the collection. Typically, all leaf labels (those
without children) are used this way. ‘Drill-down folders’ can be used to view a
lattice with its child folders as the scale. When used in a lattice view, all child
labels represent the union of their descendants’ objects. This way, the higher in
the attribute hierarchy, the more general the objects.

4.2.3 Visualisation

As mentioned previously, the complete lattice structure is never created or visu-
alised due to the complexity of the knowledge base. To control the complexity
of the lattice, only points in the virtual folder hierarchy, and therefore the at-
tribute hierarchy, can be used for creating concept lattices. This limits the user
to a countably finite set of lattices, each of which has a label representative of its information content.

Creation of the lattice context for a given label in the attribute hierarchy is performed by taking all direct child labels as attributes. The object set for context is the union of all queries from the given label to all its child labels (direct and transitive). Each attribute is assigned objects (emails) which match the union of all queries from that label and its child labels (direct and transitive).

For example, in Fig. 4.3 the lattice is generated for the label ‘Attachments’. The attribute set is the direct children, namely ‘Mail with Documents’, ‘Mail with Images’, ‘Large Attachment Size’ and ‘Small Attachment Size’. The object set will be the combination of the queries from the label ‘Attachments’ and all direct and indirect child labels. Each attribute is then assigned the union of its query and its child queries. The ‘Attachments’ folder itself has the query ‘Attachment-Present True’ so that ‘Attachments’ represents all email with attachments, and not just those that match its children’s queries.

The visualised lattice offers various manipulations to allow the user to peruse the presented information. Concepts can be arranged, the view can be zoomed,
4.3. USABILITY EVALUATION

panned and centered, and font size can be increased or decreased. The visualised concepts themselves offer the user one of two choices, extent or contingent, which can be realised as emails for closer inspection.

4.2.4 Usage of the MailSleuth Model

Initially the user has an email collection. MailSleuth is then integrated into the user’s existing mail system. The user is then required to create/adjust the attribute hierarchy to suit their needs. This is done by creating folders with meaningful names and assigning queries, or nesting more folders within the created folders.

For instance, the creation of a ‘Conferences’ Drill-Down Folder, that contains a ‘SIGIR’ Virtual Folder with a query ‘Email-Body sigir’, and another Virtual Folder ‘ICFCA’ with the query ‘Email-Body icfca’. In order to ensure that the ‘Conferences’ label is more correctly represented at higher levels, the user attaches the query ‘Email-Body conference’.

A lattice view of the conference information space can now be seen via the ‘Conferences’ Drill-Down Folder. At a high level in the attribute hierarchy the ‘Conferences’ folder will represent the union of the three queries:

\[ \text{Email-Body conference} \cup \text{Email-Body sigir} \cup \text{Email-Body icfca} \]

These labels can be duplicated within the attribute hierarchy (provided they do not violate the partial order) to combine facets of the information space into new structures. These structures will be kept consistent with the mail collection as new mail enters, old mail is archived, or unwanted mail is deleted.

4.3 Usability Evaluation

MailSleuth underwent usability testing at the Access Testing Centre (ATC). Testing was broken into two components; a comparative functionality review and user–based evaluation sessions.
4.3.1 Comparative Functionality Review

The comparative functionality review was conducted by two ATC analysts performing self-determined exploratory testing. The evaluation had significant comparative aims: comparing the ease of executing various key functions in MailSleuth against competitor applications.

The May 2003 version of MailSleuth had no initial virtual folders when the program was first installed and the user had to go through a folder configuration process from scratch. In FCA terms, MailSleuth had no pre-defined conceptual attribute hierarchy (or conceptual scales). While this may suit advanced users, or users expecting the program to be a general purpose framework for document browsing using FCA, the majority of users are likely to encounter difficulties with this. ATC recommended a number of useful pre-defined Virtual Folders be employed such as “This Week”, and “Attachment” folders for email attachments of different document types and sizes. These form the basis of the Folder List shown to the left of Fig. 4.5. This recommendation was followed and in subsequent versions pre-defined Virtual Folders were added, including the folders mentioned above (various popular document and image attachment types and sizes) and also a “follow-up” folder which tests the Outlook™ follow-up flag. These serve as examples and a useful starting point from which users can extend the Virtual Folder structure (attribute hierarchy), while benefiting immediately from the software. Other comparable products derive Virtual Folders from reading the mailbox but the structure (once built) cannot be modified or extended as with MailSleuth. This advantage is highlighted by including an extensible pre-defined folder structure when the MailSleuth program is first installed.

At the same time that pre-defined Virtual Folders were added, the idea of “User Judgments” (reported in [CES00]) was eliminated. User Judgments allow the user to over-ride the automatic classification specified in the attached Query of the Virtual Folder. Emails could be drag-and-dropped from regular folders
into (or out of) existing Virtual Folders. ATC reviewers found this to be a powerful feature but one that would appeal only to expert users.

4.3.2 User-based Evaluation

The user-based evaluation involved one-on-one interviews and was intended to evaluate the ease of use and expectations of the user community. Six users were drawn from the core target demographic. There was a balance of male and female degree qualified individuals who had expressed an interest in new techniques to categorise and handle their email. Ages spread from 25 to 50 with at least one under 30 and at least one over 40. Included in the group were a Librarian, an Insurance Manager, a Financial Analyst, a Recruitment Manager, an Imaging Specialist and a Personal Assistant. Each user session lasted at most 90 minutes and was directed by a usability analyst who observed tasks and recorded relevant data. Each session was then analysed to identify any usability issues and quantitative measures were compiled.

Fig. 4.4: The red arrow button (which realises the contingent, is shown pointing right and displayed to the left of the concept) and blue arrow button (which realises the extent, is shown pointing down and displayed below and left of the concept) are displayed when the cursor hovers over the envelope concept icon. The extent and contingent sizes are displayed next to the envelope as a fraction (contingent / extent) and the corresponding numbers are underlined when the user interacts with the arrows.
4. MAILSLEUTH

4.3. USABILITY EVALUATION

Fig. 4.5: Screenshot of the final version of Mail-Sleuth. The line diagram is highly stylised and interactive. Folders “lift” from the view surface and visual clues (red and blue arrows) suggest the queries that can be performed on vertices. Layer colours and other visual features are configurable. Unrealised concepts are not drawn and derived Virtual Folders are differentiated from Named Virtual Folders. A high level of integration with the Folder List to the left and the Folder Manager is intended to promote a single-user Conceptual Information System task flow using small diagrams. Note that the buttons in Fig. 4.8 have been replaced with tabs and that the help system is more consistently located to the top right. The Quick Search bar is visually highlighted and placed toward the bottom of the screen for greater emphasis.

4.3.3 Findings and Actions

The majority of participants were able to learn the basic operations associated with MailSleuth and to complete the small number of pre-defined tasks. With a simple orientation script (in the place of a help system, incomplete at the time of testing), participants quickly learnt to use the software. For example, once introduced to the concepts of Virtual Folders and how they are associated with a query (or queries), participants were able to use the application to create their own folders and populate them with appropriate queries. Participants indicated they found the interface reasonably intuitive and easy to use.

‘An encouraging finding was that participants were able to read the lattice diagrams without prompting. Subject six even used the word
lattice without it having been mentioned to her. Participants correctly interpreted the major elements - for example, how the ‘envelope icons related to the mail folders and how derived vertices represented the intersection of two folders.’

ATC Final Report, Usability Analysis, September 2003

There were a number of improvements that could be made to the visualisation component, in order to present the lattice more clearly:

- ‘The start and end nodes could be removed from the legend and realisation arrows could be added.’

The introduction of the realisation arrows (coloured red and blue) into the lattice diagram highlights the interactive nature of the lattice diagram as a tool for querying emails. This compensates for the fact that only named folders can be accessed via the Folder List. The realisation arrows are clues from the line diagram that the extent and contingent are available and that Derived Folders can be created by manipulating the diagram.

- ‘For more complicated structures, less emphasis could be placed on regions that are essentially ‘unmatched’. This would reduce visual clutter and further highlight the relationships that do exist.’

This comment refers to the small concept icons that are used to represent unrealised concepts, and many of the test subjects expressed similar sentiments. To solve this, concept representations at unrealised vertices in the line diagram were eliminated. The introduction of a reduced line-diagram was also included as an option for advanced users.

- ‘The format for representing total and dispersed emails associated with each folder could be more clearly represented - some users indicated that the present format (using brackets) represented total and ‘unread’ e-mails. A reference to the format could be included in the legend.’
Tying together the textual representation of extent and contingent to the red and blue arrows and the total (extent) and dispersed (contingent) sizes being represented as a fraction resulted (as shown in Fig. 4.4).

- ‘The initial/default node view could be improved – when elements are close their labels can overlap. An interesting finding was that some users found more complicated diagrammatic representations better conveyed the relationships to the left-hand folder list.’

The ability to adjust the highlights and font sizes for diagram labels was included (along with the ability to colour the layered shading). The observation that more complex line diagrams more strongly linked the line diagram to the Folder List is because a larger line diagram contains more labels appearing in the Folder List. Thus, the correspondence from line diagram to Folder List is more easily made when there are a larger number of intersecting elements.

Finally, user responses in this small demographic give encouraging indications of an implicit understanding of information visualisation using line diagrams. When shown a very large line diagram, the librarian found it overwhelming but was certain that there was “value in a lattice of the information space”. More specifically, one user said that she preferred a reduced line diagram, namely she saw “no reason that points without corresponding data should be drawn at all”.

When asked what they liked most about the application users responded with statements such as; “Defined searches – better time management. Ability to separate text from e-mails, program creates folders for you”. We interpret this to mean that this user understands that a permanent standing query is created attached to a Virtual Folder. The term “Virtual Folder” was also used by another respondent when asked the same question “Drilling down through virtual folders to locate specific emails etc.”, this indicates a familiarity with the idea of a Virtual Folder, either pre-existing or learned during the 30-40 minutes
using the program. Further, the use of the term “drilling down” in the appropriate context of data mining and visualisation suggest an encouraging level of comfort among the target user group with the terminology of the program.

Table 4.2 shows that the user group could use MailSleuth and had a clear understanding of its utility. While questions 8 & 9 (which relate to visualisation of line diagrams) scored relatively poorly compared to other questions, it is apparent that the results are nonetheless positive and doubtful that other question groups would have been so highly scored if the line diagrams had not been understood. Nonetheless, improvements to the visualisation aspects of the program did result, mostly on the basis of the users written comments.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Ave. Resp.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  Clear how the application is to be used</td>
<td>1.3</td>
</tr>
<tr>
<td>2  The interface was simple to use</td>
<td>0.8</td>
</tr>
<tr>
<td>3  The application appears to be a useful tool</td>
<td>1.8</td>
</tr>
<tr>
<td>4  I liked the layout of the pages</td>
<td>1.2</td>
</tr>
<tr>
<td>5  I found the icons intuitive</td>
<td>0.5</td>
</tr>
<tr>
<td>6  I found the Quick Search feature was useful</td>
<td>1.0</td>
</tr>
<tr>
<td>7  I found the folder view intuitive</td>
<td>1.3</td>
</tr>
<tr>
<td>8  I found the diagrammatic view intuitive</td>
<td>0.8</td>
</tr>
<tr>
<td>9  Clear relationship, folder view to diagrammatic view</td>
<td>0.7</td>
</tr>
<tr>
<td>10 The configuration functionality was useful</td>
<td>0.8</td>
</tr>
<tr>
<td>11 I would use this application</td>
<td>1.7</td>
</tr>
<tr>
<td>12 I will recommend this application to others</td>
<td>1.7</td>
</tr>
</tbody>
</table>

Tab. 4.2: Participants were presented with a number of statements and they were asked to select a rating. The range of the ratings went from -2 to +2, to indicate the extent to which they agreed with the statement. Here -2 = ‘Definitely No’, 0 = ‘Uncertain’ and +2 = ‘Definitely Yes’.

Considerable time was spent in the development process responding to negative comments by users during software evaluations. Negative comments were solicited when the group was asked “what they liked least” about the MailSleuth application. Responses included: “it takes a few moments to understand the 3-D concept as most people are used to a flat & hierarchical folder layout”. The response to this has been to include a careful introductory tutorial/help system to explain Virtual Folders and Structures and introduce specific “simplified” terminology to facilitate an understanding of MailSleuth. It
is noteworthy that the Virtual Folder idea also appears as one of the features that people liked most. The comment that the “diagram is a bit overwhelming and has badly chosen colours” was addressed by giving people the option of choosing their own colour schemes and font sizes and trying to simplify the line diagram as described in the next section.

4.3.4 Design Aids for Interpreting Line Diagrams

During the comparative review of MailSleuth in May 2003, a comment was made by ATC tester Peter Brawn that “the drawing conventions for a lattice diagram were no different from a graph in Mathematics. What makes this a lattice diagram and not a graph? How do I know that I should read this top to bottom?”

A line diagram (or concept lattice) is a specialised Hasse diagram with several notational extensions. Line diagrams contain vertices and edges with the vertices often labelled dually with the intent (above) and extent (below). Rather than labeling each node in the line diagram with its intent and extent, a reduced labeling scheme can be used and each object (and attribute) appears only once. In many Toscana-systems (and in CEM) a listing of the extent is often replaced with a number representing the cardinality of the extent (and/or the contingent).

In Hasse diagrams, edges (representing the cover relation) are unlabelled. It is well understood in Mathematics that a partially ordered set is transitive, reflexive and antisymmetric. To simplify the drawing of an ordered set (via its cover relation) the reflexive and transitive edges are removed, and the directional arrows of the relation are dropped. It is therefore meant to be “understood” that the Hasse diagram is hierarchical with the edges pointing upward. In other words, if \( x < y \) in the partially ordered set then \( x \) appears at a clearly lower point than \( y \) in the diagram.

‘The highlighting of adjoining lines is meant to illustrate relationships within the lattice and this could be clearer. There is a hierar-
Access Testing Centre (ATC) suggested arrowheads be used in the line diagram to reinforce its hierarchical character. This represents an unacceptable violation of a convention dating back to (at least) Helmut Hasse’s 1926 book *Höhere Algebra*, so some other mechanism to reinforce hierarchy without tampering with the edge notation in the line diagram had to be found.

To insinuate structure, the idea of a line diagram with shaded layers was introduced. The principle is iterative darkening, with dark at the top and light at the bottom; the shades progressively lighter as one moves from one level to the next. This is shown in Fig. 4.6. The top and bottom elements of the lattice have also been replaced with special icons indicating “All Mail” and “No Mail” (when the bottom element is the empty set of objects). In combination, layer shading and icon shapes are intended to suggest the top-to-bottom reading of the line diagram.

Shading does not interfere with the conventions of drawing line diagrams because it operates as a backdrop to the line diagram. It can also be turned off if the line diagram is to be embedded in a printed document. However, the interaction of the layout algorithm and background layer shading fails (background layers are not aligned) in line diagrams with high dimensionality as shown in Fig. 4.7 (left) requiring human intervention to produce something readable as shown in Fig. 4.7 (right). It is possible to use the alignment of the background layers to guide the manual layout process. Nonetheless, once layer shading was used, it was apparent from test subjects that they were (without prompting) able to explain (and read) the line diagram from top-to-bottom and bottom-to-top.

*It was observed that most nodes in the lattice are depicted using the exact same icon, even though there are a variety of nodes. In par-

ATC Functional Testing Report, May 2003
Fig. 4.6: A line diagram from the August 2003 version of MailSleuth. Layer shading is evident to suggest a hierarchical reading. Top and bottom elements have been especially iconified. Unrealised concepts are differentiated. Realised concepts are split into two iconic categories “Named Folders” with an intent label with a white envelope and “Derived Folders”, whose intent needs to be derived as an orange envelope. Cardinality labels have been replaced with dual labels as “extent (contingent)”. Users complained that the help system was hard to activate or they could not find it and did not recognise the ‘?’ icon as being related to help. Note: This version includes a Quick Search bar at the top which provides an entry point for text-based search within the current information space.

In MailSleuth the top of the lattice represents all emails in the collection. Some of the vertices shown in the line diagram correspond with actual Virtual Folders that exist in the Folder List to the left, while other vertices represent derivations of the named Virtual Folders. It is useful to indicate, through different icon types, which vertices are named Virtual Folders (appearing in the Folder List), and which are derived. This led to the idea of a “Derived Folder”, a type of Virtual Folder that does not appear in the Folder List and whose
existence is derived from the named Virtual Folders (attribute names) above it in the line diagram.

The number of e-mails represented in each node could also be more clearly illustrated. For example, where totals for vertices and intersections are concerned, two numbers could be displayed corresponding to the extent and contingent size in the form extent size (contingent size). This is shown in Fig. 4.6.

‘Hide the lattice-work where no relationships exists.’

[User 6]

When drawing “reduced line diagrams”, unrealised concepts are excluded but automatic layout can be problematic with this type of diagram. As Mailsleuth is designed for the non-expert, it is important that the lattice diagram always be as readable as possible by default, and therefore reduced-line diagrams are an optional feature. Where elements of a scale are unrealised, the entire label is excluded from the diagram. However, what remains is drawn as a distributive lattice. This means that certain combinations of realised scale elements may themselves be unrealised.

Convention dictated that these be displayed as a vertex in the lattice somehow distinguishable from realised concepts (or not at all). In Fig. 4.8, unrealised concepts are the same shape and size as realised concepts, the only difference
4. MAILSLEUTH

4.3. USABILITY EVALUATION

Fig. 4.8: A line diagram from the May 2003 version of MailSleuth. Most of the usual FCA line diagram labeling conventions are followed with the exception of iconifying vertices with an envelope. There is no obvious “search point” (meaning no clear starting place to commence the search) and limited visual highlighting in the diagram itself. Structural diagrammatic constraints are imposed, for instance concepts cannot be moved above their superconcepts.

being the presence or otherwise of an envelope icon within the vertex. To distinguish unrealised from realised concepts, they were reduced in size as shown in Fig. 4.6. Top and bottom concepts (when the bottom was an empty set of objects) were also iconified. In addition, realised concepts are identified in two ways. The first is where the intent label matches a “Named Folder” in the Folder List of Outlook™ (to the left of Fig. 4.5). The second is where vertices represent the intent labels of the upper covers – these may have common attribute names (Named Folder names) and are coloured orange. To avoid cluttering the diagram with labels on all vertices, the interface gives scope to query an orange envelope and the result is a new Virtual Folder, named after the intent labels of its upper covers appearing in the “MailSleuth search results” in the Folder List.

‘... get rid of the grey blobs...’

[User 2]
Because we are dealing with objects that are emails, it was natural to replace the stylised vertices (a legacy of the Hasse diagram) with a literal iconic representation relevant to the domain. In the case where “Derived Folders” are unrealised, no vertex is drawn. Where data is present, an envelope replaces the envelope/ball icon combination as shown in Fig. 4.5. Top and (empty) bottom vertices appear at most once in a line diagram and so are removed from the legend (shown in the legend of Fig. 4.6 but not in Fig. 4.5) and labelled accordingly in the diagram itself (shown in Fig. 4.5). The ability to manipulate the line diagram in four directions via the “Pan” widget appears in Fig. 4.5, and the envelopes animate by “appearing to lift” on rollover with drop shadowing. This helps suggest that vertices in the line diagram can be moved and therefore manually adjusted by the user.

Edge highlighting has been used to emphasise relationships in line diagrams in both ToscanaJ and in CEM. This idea is mainly used as a method to orient the current vertex in the overall line diagram so that relationships can be identified. ToscanaJ allows the edges of the line diagram to be labelled with the ratio of object counts to approximate the idea of “support” in data mining. That program also uses the size of the extent to determine the colour of a vertex. A number of other significant functions for listing, averaging and visualising the extent at a vertex are also provided by ToscanaJ.

Trying to create a new user community with MailSleuth is an interesting exercise, but the original user community also requires attention. HierMail is a version of MailSleuth for the FCA community that conforms to the diagrammatic conventions of ToscanaJ. It took only a matter of days to rollback the design lessons learned from over four months of usability testing and design refinement with MailSleuth to produce HierMail as shown in Fig. 4.9.
4. **MAILSLEUTH**

4.4 **EVOLUTION**

MailSleuth tries to accommodate a new user community to document browsing using FCA. HierMail (shown above) has a much stronger conformity to diagrammatic traditions in FCA. It is effectively a version of MailSleuth with a ToscanaJ-like visual style.

### 4.4.1 **MailSleuth**’s Evolution

MailSleuth underwent considerable changes in the way the lattice diagrams were presented to the user during its development. Initially, representation was derived from the FCA’s original pen-and-paper style. Deviation from this style removed circles as concepts, and replaced them with more descriptive icons of what the concepts represent. Top and bottom concepts were replaced with iconic arrows emphasising that the relative vertical position of the concepts has meaning. Vertical position is further emphasised with use of ‘layer shading’, in which each layer of the diagram is distinctly highlighted. Highlighting is further used to show the connection between comparable concepts, by both emboldened labels and edges.

Interestingly, throughout testing few participants had difficulty using and understanding the attribute hierarchy as represented in a folder structure. Usage
of this important part of the system was implicitly understood.

**MailSleuth** provided validation of Formal Concept Analysis applicability to wider audiences than that of FCA trained individuals. While the interpretation of line diagrams was not instant, even with visual clues, participants did eventually find the representation useful, or potentially useful.

Following **MailSleuth**, a web-based version, called **MailStrainer**, was engineered and tested by Honours student Shaun Domingo [DE05a]. This version was much the same in its representation as **MailSleuth**, but allowed more freedom with respect to lattice display; lattices of significant complexity were allowed to be requested. The usability tests were performed by 16 Honours level students of the University of Wollongong. All had experience with using web-based email clients. Outcomes, in terms of task completion, were very good, but many participants were not overly happy with the line diagrams, finding them overwhelming as the number of concepts grew. Overall, testing of **MailStrainer** showed potential for FCA based systems to be used in knowledge management applications.

> ‘The prospects for novice Formal Concept Analysis users to read and interpret line diagrams remains promising but are not (as yet) considered overwhelming using present tools’

Domingo and Eklund [DE05a]

### 4.4.2 **MailSleuth** and the CKP Tasks

The main tasks that **MailSleuth** addresses are *exploration, search, recognition, identification* and *analysis*.

*Exploration* is facilitated by providing a structure over the email collection which is more expressive than the traditional single inheritance hierarchy. This structure, referred to as Virtual Folders, is represented visually with a folder structure (in the same style as the traditional display). The structure’s content
is dynamically updated – as new emails are sent and received the underlying collection changes. Exploration is further provided by the lattice views available. These views allow an overview of the email collection based on given facets of the mail in the collection. This overview can then be used to provide access to more specific clusters of mail than are represented directly in the Virtual folder structure.

Search is enabled in much the same manner as exploration, but is extended via a more traditional information retrieval keyword search system. Keyword search is possible over the entire document collection, but is more powerfully used in conjunction with the Lattice View component. Keyword searches can be restricted to a single lattice allowing an initial search space reduction step. The user accesses the Lattice View which is representative of the mail item being searched for and enters keywords that are restricted to only that lattice. Control over the search space aids search by reducing the result set, and by allowing search terms to be specific to only a subset of the collection (e.g. if looking for mail from ‘Jon’ in a folder view about ‘Jon’, there is no need to include the term ‘Jon’ if searching that view).

Identification and recognition are facilitated directly through the lattice view. For identification, the visualised inter-document relationships give insight into the nature of the virtual folders, or facets, of the collection. This allows the taxonomic position of a facet to be identified in the context of varying information spaces. Similarly recognition is made available by way of the derived folders, whether expected or not, which allows insight into the way in which emails share facets.

Support for analysis is provided by lattice views, which give an overview of how the emails within the collection are distributed with respect to specified facets of the email. For instance, forensic analysis is supported by creation of important or of-interest virtual folders. The combination of these can be visualised to allow insight into the collection. Fig. 4.11 shows an example of this technique – it can be seen that various virtual folder structures have been
created. These Virtual Folders include communication patterns (‘To’ and ‘From’ people), mail about publishers and research topics. Three of these topics are combined to allow a view over the collection focused on ‘Call for Papers’, ‘IEEE’ and ‘Conference’.
See print copy for figure 4.10

*Fig. 4.10:* Comparison of three versions of MailSleuth starting with May 2003 (before initial interface analysis), followed by August 2003 (version used in user testing) and the final, December 2003, version.
Fig. 4.11: Screenshot of MailSleuth showing Analysis. The Virtual Folder ‘Combination’ has subfolders defined as part of other Virtual Folders. Interestingly, there is an email that matches ‘Call for Papers’ and ‘IEEE’, without matching ‘Conference’. 

See print copy for figure 4.11
SurfMachine is a web-based application, which applies FCA techniques to aid a decision making process which occurs in the sport of surfing. Choosing the optimal location to surf on a given day relies on many factors. Traditionally, this decision process is made with use of experience and oral communication between surfers, combined with trial and error. The SurfMachine application elaborates this process by allowing analysis of surfing locations under given conditions. This allows search and exploration tasks which primarily allow decision tasks to be completed, but at the same time considerable focus is put onto memorisation. Memorisation is a key part of the system, allowing individual surfers to better understand the surfing locations in their local area.

<table>
<thead>
<tr>
<th>Conceptual Processing Tasks</th>
<th>Memarisation</th>
<th>Decision</th>
<th>Exploration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary</td>
<td>Search</td>
<td>Recognition</td>
<td>Identification</td>
</tr>
<tr>
<td>Secondary</td>
<td>Improvement</td>
<td>Restructuring</td>
<td></td>
</tr>
</tbody>
</table>

Tab. 5.1: CKP Task breakdown for SurfMachine.
5.1 Domain

A survey of existing web-based surfing tools reveals their reliance on Web-cams and the absence of any analytical features.\(^1\) These sites rely on low-quality streamed video inputs that are unreliable – they are often off-line, do not work in poor lighting conditions or at night and often “point” in the wrong direction to give any clear indication of the prevailing conditions.

The objective of SurfMachine is to improve on these portals by providing a more principled analysis of surfing breaks based on geographic information and weather inputs.

While the dimensionality of the input data to the system described is low, reflecting the detail (or lack thereof) contained within the input primary data sources, the SurfMachine system is a prototypical example of a Web-based CIS that integrates (in a natural way) with spatial data and improves the predominant Web-cam paradigm for portals aimed at surfers. The “landscape of knowledge” that results therefore closely reflects the practical knowledge that surfers apply when deciding where to surf.

This chapter is structured as follows. First, an overview of how a surfing location’s shape and orientation are affected by weather conditions, then a description of the data source and its limitations. SurfMachine is then described followed by how this Conceptual Information System (CIS) relates to Wille’s CKP tasks.

5.1.1 The Mechanics of Beach Selection in Surfing

One of the most important aspects of surfing is locating the best waves. Different surf locations – or breaks – on a coastline have different physical properties setting them apart. The shape of the coastline and seabed determines how and where waves will form. Ideally, a wave should break along the beach, rather than

---

\(^1\) http://www.coastalwatch.com.au
\(^2\) http://www.surfit.com.au
\(^3\) http://www.realsurf.com
\(^4\) http://www.wannasurf.com
Fig. 5.1: The angle between the swell and the shore should be sufficient that the wave breaks along the beach (left), rather than all at once against the beach (right).

all at once, thus headlands and curved beaches usually provide better surfing conditions than long straight stretches of sand (See Fig. 5.1). The exception is the occurrence of reefs\(^5\) which cause the waves to break on both sides of the reef. With knowledge of the structure of the local breaks, the determining factor on any given day is based on the current weather conditions.

The primary weather condition that determines if a break will be good is the wind direction and strength. Wind should optimally be offshore, i.e. blowing against the oncoming wave. This holds the wave up, slowing the break, and giving the surfer time to navigate the face of the wave. Wind is affected by the land structures surrounding or adjacent to a break; for example, wind can be diverted around a headland, becoming an offshore breeze when it reaches the sea. The intensity of the wind should never be too strong (in any direction), but some breaks are protected from strong offshore breezes by their geography (such as cliffs).

The secondary condition which affects the quality of a break is the swell direction and size. Swell waves are generated by stable wind systems or storm conditions far from the point where they reach the coastline. As swell travels from its point of origin, it becomes organized into groups with similar height and period. The greater the distance, the more organised the swell becomes. These

\(^5\) Underwater structures, either natural (rocks) or man-made (pipes and purpose-built artificial reefs).
groups can travel thousands of kilometres without significant changes to their height and period. An example of this are Australian “cyclone swells”, which may cause swells to start in northern Queensland, and are felt as far south as the east coast of Victoria (some 3,000kms away).

When a swell reaches a coast, the way it interacts with the shape and orientation of each break dictates how the wave will break. As mentioned earlier, the wave should break along the beach. When a break is curved, it will be better in a wider range of swell directions. Point breaks (a wave that breaks along a headland or promontory) have an interesting property that causes swell to wrap around the headland and so also allows for better surfing. Swell size, for the most part, relates to the skills and fitness of the surfer, but in situations like reef breaks it can directly influence the break’s quality (i.e. the wave has to be sufficiently large to form over a reef).

Tidal conditions are the third factor of importance. In all but the smallest swell, an incoming tide will have the effect of increasing the swell size. This is called tidal surge. The shape of the sand banks on a beach at any given time will determine the surf quality for a given tidal condition, either incoming or outgoing tide, but this is not usually something that can be codified in a surfing guide and is largely variable, depending on shifting sands. Ideal tide conditions can, however, be a predicting factor for more permanent undersea terrains, such as rock or point breaks. For the novice surfer, point and rock breaks are to be avoided because they require much higher skill levels to be ridden safely.

In general, the combination of wind and swell direction, and tidal condition mean that each break has its own characteristics and nuances, which are usually learnt through personal experience and local knowledge.

5.1.2 Sources of Data

For this study, the primary data source was, “The Surf Report - Journal of World-wide Surfing Destinations”. Despite being published in the US by Surf
5. **SURFMACHINE**  

5.1. **DOMAIN**

Publications (Volume 17 No. 2, 1996)$^6$, this is the most authoritative guide on Australian surfing known to the author$^7$. It is a detailed (often obsessively so) list of all breaks – listing some 120 different breaks over almost 500 kilometres of coastline on the South New South Wales coast. Only 44 of the breaks were used in our study (considered “local” breaks), mainly due to data entry constraints. The 44 selected breaks are those around the Illawarra area from *Garrie Beach* at the Royal National Park (near Sydney) down to *Minnamurra* (north of Kiama) covering 94 kilometres of coastline.

Wind and swell directions are uniformly presented in “The Surf Report - Journal of World-wide Surfing Destinations” making data acquisition in these dimensions straightforward. The same cannot be said for other attributes such as *tide conditions*, *wind strength* and *swell size*. These attributes are somewhat obscured in the text, for instance the use of fuzzy phrases like “works well on large swells”, and surf idioms like “southerly buster”, “double over-head..” and “ebb-tide danger-spot..”. These give character to the narrative as well as useful information about swell size, wind and tidal conditions (if you are a surfer and can understand the idioms). These features were not presented for most breaks, and those that were created difficulties when extracting and quantifying for use in data mining.

Data inconsistencies were checked with a secondary source[War98] and with local surfers, for most part to normalize the local name of the break rather than use the official map location name$^8$.

The context derived from this data uses the breaks as the object set $G$. This is because the breaks form part of the desired result set.

$$G := \{\text{Garrie Beach, ..., Minnamurra}\}$$

$^6$ First Published in 1984.

$^7$ Given the narrative nature of the material it is difficult to quantify the quality of these types of information sources, so this assessment is based on opinion.

$^8$ Surfers often use colloquial names for breaks, for example, Kilalea Beach is known as “The Farm” because for many years access to that beach required surfers to pay a toll to the local farmer in order to cross his land.
The attribute set $M$ is comprised of possible wind and swell conditions, which form a constraint over the objects. These conditions are broken into eight possible compass directions. (Note: $W_N$ is representative of Northerly Wind. Also, this does not include swells that cannot occur, i.e., swells do not originate from inland.)

$$M := \{W_N, W_{NE}, \ldots, W_W, W_{NW}, S_N, S_{NE}, S_E, S_{SE}, S_S\}$$

The incidence relation $I$ between the elements of $G$ and the elements of $M$ can be thought of as “works well in” and is represented by:

$$gIm \iff \text{break } g \text{ works well in condition } m, \text{ where } g \in G \text{ and } m \in M.$$
5. SURFMACHINE

5.2. APPROACH

The SurfMachine system is a simple Web-based CIS and comprises an area to capture the user’s query, a lattice display and a geographical map of the areas covered.

The user enters a query with two facets, wind and swell direction via a graphical representation of two compasses. This provides an input control so that only valid compass points can be entered (see Fig. 5.3). Both wind and swell directions are split into 8 discrete values (North, North-East, East, South-East, South, South-West, West, North-West) and queries are limited to a range of directions not exceeding a range over 4 compass points, e.g. North through South-East, which is 4 values (N, NE, E, SE). This is done because wind is frequently reported over a range or as shift over time (e.g. “SW winds turning SE”). Fig. 5.4 shows the result of a query formulation. Queries are limited to four inputs per modality to ensure the produced lattice is not overly complex.
Fig. 5.5: The lattice component displays the break data as a conceptual breakdown of the current weather conditions.

Each of the values are used in a scale representing current weather and the corresponding concept lattice is generated (see Fig. 5.5). The extent of each concept of the lattice translates to a set of locations on the map, and clicking the concept realises its extent as locations on the map (see Fig. 5.6).

In this way, the resulting formal concepts map to sets of geographic point data. A given formal concept may therefore contain multiple surf breaks geographically dispersed over the total area of the study. Visualizing the information space as a concept lattice allows the user to view multiple results by representing the objects in the extent as points on a map.

An important feature of SurfMachine is that the surfer can see breaks which are “similar to one another” from the concept lattice. In so doing, the concept lattice reaffirms local knowledge about which breaks work in which conditions, and suggests alternate locations to those the surfers may know and like. For instance, from the concept lattice in Fig. 5.5 we can see that Garie Beach, Thirroul, Sandon Point, ... , Austinmer all cluster to the same formal
Fig. 5.6: (left) The map component uses a simplified vector-based map of the extended Illawarra coastline; and (right) The map component realises concepts as point data for the given weather conditions, and places markers and labels on the map, then zooms to optimize the map display.
5.2. APPROACH 5. SURFMACHINE

Fig. 5.7: The complete interface view of SurfMachine.

concept under the input weather conditions. They will therefore all be similar in a SW Wind and a N Swell, so the surfer would feel inclined to visit the closest on the day or, if they feeling adventurous, try a different break with knowledge that that break should be similar to one already surfed in the given conditions.

SurfMachine uses the Model–View–Controller design pattern, with the server holding all data and performing processing of the conceptual data (the model). The client controls all transactions and data flow (the controller) and uses an embedded plug-in for lattice drawing and a map view component to render the spatial data (the view). The architecture is shown in Fig. 5.8.

The model goes through two phases, a creation phase and a usage phase. The creation phase makes use of the various data sources to create a concept lattice that encompasses the entire context and a database of coordinates. This phase was used initially to set up the databases and then subsequently to restructure the data when faults or deficiencies were found. The usage phase centres around a Query Processor which accepts the query from a client and returns XML
representing the lattice to be displayed. The Query Processor is divided into four steps; filtering, rebuilding, adding map information and mark-up into XML. The term ‘filtering’ is used in the lattice theory sense, whereby the attribute set of a lattice is restricted to produce a sub-lattice. The next step rebuilds the layout so that it is optimised and the filtered lattice will graphically fit better in the lattice viewer component. The point data is then added to the concepts by querying the spatial database and extracting the breaks for each concept. Finally, the various data is brought together and formatted into XML for transport to the browser client.

5.3 Evolution

5.3.1 SurfMachine and the CKP Tasks

The main focus task of SurfMachine is the Decision task. A surfer knows the current weather conditions, but where should they surf? By restricting the information space to the current weather conditions, a conceptual ranking of breaks is presented. This ranking allows the surfer to decide which facets of...
the current conditions are most important, or use the map representation to eliminate breaks too far away. The workflow for such a process, once conditions have been set, would normally start at the bottom-most concept, this concept’s extent being the breaks which are best suited to the conditions. If the extent is empty, the breaks too far away, or the breaks undesirable for any reason, the upper neighbours are the next concepts to be considered. Emphasis would be put on those concepts that have both Wind- and Swell-type attributes (rather than only one type), as ignoring an entire facet of the current conditions may lead to a poor decision.

In Fig. 5.5, we can see the bottom-most concept’s extent is empty - there is no break meeting all of the conditions ($\{N \text{ Swell}, SW \text{ Wind}, S \text{ Wind}, SE \text{ Wind}\}$). The upper neighbours of the bottom-most concept have intents of $\{N \text{ Swell}, SW \text{ Wind}, S \text{ Wind}\}$ and $\{SW \text{ Wind}, S \text{ Wind}, SE \text{ Wind}\}$. It would be preferable to choose a break from the concept with the intent that has both Wind- and Swell-type attributes. We can see that this concept has a extent of $\{Headlands\}$, which is probably the best break for these conditions. Of course, if this break was undesirable then this process could continue by accepting only Wind-type attributes, or relenting another attribute and looking at the upper neighbours of the $\{Headlands\}$ concept.

Another important task provided by SurfMachine is memorisation. Within the information space, similar beaches are always grouped allowing the user to quickly assimilate breaks with similar features. By using SurfMachine, a surfer can gain experience with their local coastline at an accelerated rate. The knowledge bases used in the system contains a lifetime of local experience which is arranged and accessed in a meaningful manner. SurfMachine achieves this by complementing the traditional learning process with a holistic view of various local breaks under various conditions.

The exploration task is closely related to both decision and memorisation tasks, neither of which could be facilitated without performing some amount of exploration. Exploration is the process of moving through the lattice, looking
at the breaks listed, and comparing these with their attributes. This process may lead to a decision or could be undirected exploration over the information space.

*Exploration, decision and memorisation* are the primary tasks facilitated by SurfMachine. Other tasks are made possible via the interface; albeit to a lesser degree.

The *search* task looks for a known break within a given set of conditions. The map can help facilitate this requirement. Once found, the break’s concept can be seen within the structure of the information space, and therefore in some way implicitly ranked within the conditions against other breaks. Higher rankings will be achieved by concepts lower in the lattice structure as they can be considered closer to ideal (or the bottom-most concept).

*Identification* is easily performed, as SurfMachine displays a lattice structure as a result of queries. This means that all breaks are part of a taxonomy of breaks, allowing easy classification by the various conditions set by the current query. This, in turn, allows *recognition* tasks, breaks known to the user are shown in terms of similarity and quality, with equivalent breaks being grouped as a concept and better breaks being closer to the bottom-most concept.

By using the structure of the conceptual language and the way it supports the collected knowledge, *analysis* tasks can be performed. For instance, two of the most reliable beaches in the Illawarra region during summer are *Port Beach* and *Woonona* - everyone who surfs locally “understands” this fact. Therefore, by applying the prevailing summer conditions (*North East* wind & *North*, *North East* swell), it is reassuring that SurfMachine shows these two beaches as the extent of the bottom-most concept.

By *investigation* of different sample conditions, it is possible to see that there is a group of beaches which always appear together. The concept with these breaks stands out under most conditions as having a comparatively large extent. It is visible in Fig. 5.5, Fig. 5.7 and Fig. 5.9. This means the group represented by this formal concept is always seen together and because they
Fig. 5.9: The two best beaches in summer on the Illawarra are Woonona and Port Beach because they ‘work’ on the prevailing summer winds and swells.

share a large portion of the possible attributes these beaches can be considered “safe” beaches, because they are recommended in the majority of conditions. At the other end of the spectrum is a break by the name of ‘Stoney’s’, which is the only break in the data which has ‘E Wind’ as an attribute. Initially, this was assumed to be an error, but after some research it was found that ‘Stoney’s’ is in fact one of the only breaks on the entire east coast of Australia considered surfable in an onshore breeze because it breaks on the western edge of an off-shore island.

Improvement and restructuring tasks are not supported by SurfMachine. This is because the knowledge base of break features and geographic locations is static and therefore these physical and defined attributes of the space are not mutable, disallowing these tasks. However, these tasks were in some sense used during construction of the knowledge base - using the knowledge and experience of a local surfer, erroneous or incomplete data was amended.
5. SURFMACHINE 5.3. EVOLUTION

5.3.2 Changes from MailSleuth

Many of the lessons learnt testing the usability of MailSleuth are retained in SurfMachine. Like MailSleuth, SurfMachine displays data as a concept lattice. The lattice rendering uses the same component used in MailSleuth, but with ToscanaJ–style concepts (discarding the iconic concept representation).

SurfMachine has a much different task focus to MailSleuth and this is reflected in the exposure and control of the underlying conceptual structure. While MailSleuth supports user generated and specified queries which are put together to form scales, SurfMachine has flat (non–hierarchic), fixed scales with a fixed values. This simplifies the complexity of the system, enabling the memorisation task.

Another key difference is the in–line realisation mechanism used in SurfMachine; surf locations are represented next to the lattice as points on a map. This allows a tight coupling between the structure over the information space and the extension of a given concept. In MailSleuth, viewing realised objects is performed on a separate screen from the lattice.

Compared with MailSleuth, SurfMachine has a much less complex interface and this reduces its capability to address more complex tasks such as analysis and investigation. The simplification of the interface however allows easier decision and memorisation.
6. **DSift: SIMPLIFIED CONCEPTUAL ANALYSIS**

**DSift** is a framework for relational schema-based navigation via a web-based application that uses Formal Concept Analysis as a metaphor for user interaction. **DSift** is intended to provide users who are untrained in FCA with practical and intuitive access to the core functionality of FCA for the purpose of exploration of relational data.

This is achieved by enabling the user to build concept lattices interactively through the dynamic definition of search boundaries (via interaction with an object “zoom” feature) and dynamic selection of search scales (via interaction with an attribute “filter” feature) based on the attribute values contained within the database and its schema.

<table>
<thead>
<tr>
<th>Conceptual Processing Tasks</th>
<th>Primary</th>
<th>Secondary</th>
<th>Not</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exploration</td>
<td>Recognition</td>
<td>Decision</td>
<td>Search</td>
</tr>
<tr>
<td>Identification</td>
<td>Investigation</td>
<td>Improvement</td>
<td>Restructuring</td>
</tr>
<tr>
<td>Investigation</td>
<td>Analysis</td>
<td></td>
<td>Memorisation</td>
</tr>
</tbody>
</table>

*Tab. 6.1: CKP Task breakdown for **DSift**.*

This chapter presents the architecture of **DSift** and illustrates the system on an example database. The two examples presented demonstrate the generality of system integration outcomes from **DSift** to browsing using Formal Concept Analysis.
6.1 Domain

The DSift prototype accepts simplified database information in the form of a user-supplied *comma separated values* (CSV) database and provides an interface to this database in the form of a conceptual information system. As such, the input format of DSift closely aligns with the typical export format of relational database management systems (RDBMS) and common applications such as *Excel* and *OpenOffice*.

The CSV format is simple, easy to read and easy to edit. It is a common optional output format for most modern and legacy applications and database systems. CSV files are constrained to only contain data that can be expressed as text; this caters to the input requirements of DSift. To translate the CSV database into a Conceptual Information System, the user indicates how DSift should treat each field. This requires the user to indicate a field which is each entry’s identifier and then group the remaining fields into nominal or numerical scale models.

In order to extract objects with meaningful names, the user identifies the field which provides an identifier for the database (e.g. a candidate key such as *name* in a database of people). Nominal data, in FCA terms, is usually text (e.g. *names* or *locations* in the people database), and sometimes represents boolean values (e.g. attributes such as *gender* or attributes with values such as *yes/no*). Numerical data is represented by numbers which, over the scope of the entire field, have some form of ordering (e.g. a schema attribute such as *length in meters* with some entries longer than others).

There are instances where database attributes consisting of numeric data should not be scaled ordinally; for example, identifiers such as social security numbers, which may or may not be indicative of an order over the data values. DSift also gives the option to drop fields that are not of interest to the user (e.g. *comment* or *ID* fields), by tagging those fields as such.

Interaction with the CSV file as described allows DSift to collect enough
DSIFT

In this chapter, a database of mobile phones and their various attributes is used for examples and demonstration. This database provides a large range of objects within a restricted domain and a variety of attribute types. There are 217 mobile phones, and 16 attribute groups (or scales) in the database.

6.2 Approach

DSift offers the user a flexible tool for viewing the various structures and relationships that are present in a database. The user only needs some understanding of the data they are viewing – enough to understand the objects being dealt with and the meaning of attributes, and some level of ability reading lattice diagrams. The owner of a database should know its content, and as illustrated in Chapter 4, MailSleuth testing showed that users can quickly become competent at reading lattice diagrams with little or no formal training.

The user constructs queries by selecting query elements of interest and assigning them to one of two lists corresponding to zoom and filter.

Query elements are made up of one or more nominally-scaled or numerically-scaled attributes. Nominally-scaled attributes comprise attribute groups and an attribute value. Numerically-scaled attributes comprise an attribute group, a size and an order. The size of a numerically scaled attribute can be thought of as the number of intervals which will be produced, while the order specifies the way in which the values should be compared. The orders are of three types, Ordinal Up, Ordinal Down and Interordinal. Ordinal Up and Down correspond to comparisons based on $\geq$ and $\leq$ respectively. Interordinal generates both Ordinal Up and Down simultaneously (See Fig. 6.1).

The zoom list should be populated with query elements that are ‘required’. The elements of the zoom list are used to restrict the object set of the context only to objects with elements in the list. This is a conjunctive query, so if dichotomous elements are in the zoom list the object set will be empty. Query elements in the zoom list will always appear as attributes of the topmost concept.
Fig. 6.1: Concept lattices showing the different types of numeric scaling available via the DSift interface are (from left to right) Ordinal Up, Ordinal Down and Interordinal. All are shown with a size of 3.
of the lattice diagram. Ordinal element groups cannot be added, nor can two
element values from the same element group.

The filter list should be populated with query elements that are ‘of interest’. The elements in the filter list are used to restrict the attribute set of the context. This means that only elements in the filter list (and any from the zoom list) will feature in the resulting lattice. It is these attributes that are used to show structural relations in the database.

Using this query building paradigm of ‘required’ and ‘of interest’, the user can perform exploratory tasks against the database. The simplest example of this is the idea of a ‘search’ for an object that meets several criteria, or aids in the discovery of the ‘next best’ when the exact result is unavailable. In our example, a database concerning mobile phones, we imagine a potential customer for a new phone. The user may have a rough idea of the technical features but no understanding of which of these they really need or want. After obtaining an overview of the data, the user can sort the features into those that are essential and those that are softer constraints on the search. The user may have already encountered dichotomous features, but not knowing which to eliminate may continue to use both. Step-by-step the user will make decisions and compromises before selecting the phone with the features that are most desired and satisfy the search criteria. The last part of the search process will require many comparisons and iterations when exploring the information landscape with multiple dichotomous attributes.

As more filter elements are added, the complexity of the resulting lattice will most likely increase exponentially. To counter this increasing complexity, which can make the diagram difficult to understand, elements from the filter list can be promoted to the zoom list. This will decrease both the object set and the number of attributes used to show the structure of the data, which in turn reduces the complexity of the lattice diagram.

The advantage of having the structure as a lattice is that the user can ‘see’ relationships. Of these relationships it is easiest to see relations such as mutual
Please see print copy for figure 6.2

Fig. 6.2: Diagram generated from the Phones database with mp3player:yes, games:yes and chat:yes as filter elements.

exclusivity and implication. Figure 6.2 shows a simple lattice diagram. The user can see that mp3player:yes and chat:yes are mutually exclusive (there are no phones with both an MP3 player and a chat function) because the point where the concepts join (reading the diagram downwards) has an extent size of 0. Also, it can be seen that chat:yes implies games:yes (every phone with a chat function also has games). In a search context, where the desired result has all query elements in the filter list, it can be seen that there are 0 total matches (bottommost concept extent size is 0), but there are 7 phones that meet 2 of the query elements (the concepts directly above the bottommost concept).

6.2.1 Case Scenario One

This section will demonstrate some of the ideas described in the previous sections by showing a more concrete interaction scenario.

This scenario takes an approach that uses identification, analysis and recognition tasks on the user’s part to arrive at a decision about which mobile phone suits their requirements. This is done by iteratively narrowing the information space until a solution is found. The user must be able to comprehend the structure of the information space in order to perform meaningful iterations toward
their goal.

The user in this scenario has a range of features they consider to be important (and desirable) in a new mobile phone. In this case scenario, the user wants:

1. GPRS support
2. Infrared capabilities
3. Built-in MP3 player
4. Built-in organiser
5. Vibration alert
6. Voice-dial
7. WAP support

The user adds all the corresponding attributes as filter attributes resulting in the lattice shown in Fig. 6.3.

Please see print copy for figure 6.3

Fig. 6.3: Case Scenario One: Initial Search

The resulting line diagram from these filter attributes is too large for the user to make an instant decision. However, the line diagram gives an overview of the search space and, as such, it is possible to conclude the following:
1. The top-most concept has a contingent of 60, therefore there are 60 phones with \textit{none} of the desired features.

2. The bottom-most concept has an empty extent, therefore there is no phone with \textit{all} of the desired features.

3. The attributes \texttt{vibe:yes}, \texttt{voicedial:yes} and \texttt{wap:yes} are most common in this diagram. This is recognizable by the fact that they are darker in colour - indicating a larger extent in comparison to other concepts. \footnote{This is the coloring style explained further in Chapter 2.}.

The \textit{analysis} of this knowledge leads the user to \textit{zoom} on the three common attributes, which reduces the complexity of the data while still maintaining the majority of the phones. The resulting lattice is shown in Fig. 6.4.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure6.4.png}
\caption{Case Scenario One: Second Search}
\end{figure}

The result shows that at least one desired feature cannot be kept, and that the selection is from 7 phones in 3 groups – each group has one of the desired features missing.

- The \textit{Siemens SL 42}, \textit{Siemens SL 45} and \textit{Siemens SL 45i} do not have GPRS Support.
6. DSIFT

6.2. APPROACH

- The Ericsson T65 does not have infrared capabilities.

- The Motorola Accompli 008, Nokia 6310 and Nokia 8310 do not have a built-in MP3 player.

At this point the user must decide to either change their requirements (perhaps by dropping one feature) or opt to wait until a phone is made that meets their requirements.

6.2.2 Case Scenario Two

This scenario uses exploration, investigation and identification on the user’s part to move toward a decision. Rather than starting with a large list of required features, this scenario starts with only two features and reflexively applies personal needs to the current information view to move toward a result.

The user knows that two features they definitely want in a mobile phone are predictive text and infrared capabilities. They add t9dict:yes and infrared:yes as filter attributes and start the search. This produces a simple lattice, shown in Fig. 6.5, with 27 phones on the bottom concept. This means there are 27 phones with both predictive text and infrared capabilities.

Please see print copy for figure 6.5

Fig. 6.5: Case Scenario Two: Initial Search
The list of 27 phones is too large for the user to reach a decision, so the user promotes \texttt{t9dict:yes} and \texttt{infrared:yes} to \textit{zoom} attributes. The user decides that an organiser and a long stand-by time are also important features which would influence their decision, so they add the corresponding attributes as filters on the data. When adding stand-by time – the aim being to emphasise phones with a greater stand-by time – the user configures the \texttt{stand-by} attribute to be ‘Ordinal Up’ resulting in the diagram shown in Fig. 6.6.

\begin{center}
Please see print copy for figure 6.6
\end{center}

\textit{Fig. 6.6: Case Scenario Two: Second Search}

After looking at the generated lattice, the user decides enough phones come with an organiser to warrant adding \texttt{organiser:yes} as a \textit{zoom} attribute. The user also remembers that a long stand-by time might come at the cost of increased phone weight. To ensure that the phone chosen is not too heavy for the user’s needs, \texttt{weight} is added with the order ‘Ordinal Down’ so that lighter phones are emphasised, resulting in the diagram shown in Fig. 6.7.

This diagram allows the user to quickly choose an optimum weight / stand-by time combination. It can be seen in the diagram that the phone most suitable to the requirements specified is the \textit{Nokia 8310}. This phone balances weight and stand-by time, while also having the features desired by the user.
6. DSIFT

6.3 EvOLUTION

This chapter has presented the architecture of the DSift browser, and has illustrated the resulting DSift-system on two case scenarios against a database of cellular phones. During data import, DSift uses CSV files, deriving attribute groups from schema information and allowing users to classify these groups for scaling purposes. The DSift interface then allows navigation of the information space. One of two modalities of operation can to be assigned to attributes or attribute groups to produce a lattice diagram. The user is directly exposed to controls that restrict and elaborate the current view of the information space. This is achieved by access to the scale attributes, and by allowing the assignment of any number of filter or zoom attributes.

6.3.1 DSift and the CKP Tasks

The primary tasks performed when using DSift are those that involve reading and understanding the lattice and modifying the current view on the information space. Exploration is performed when the intentional boundaries of the current view are altered, such as when a filter attribute is promoted to a zoom attribute.
This type of interaction moves the user through the information space.

Identification and analysis of the diagram are also key tasks, enabled by the combination of the attribute filter and zoom methodology with the lattice view. These tasks are required to enable the user to make meaningful changes to the view of the information space, as well as to extract knowledge from the data.

The addition of attributes as either zoom or filter allows investigation of specific facets of the data, without having to include information that is not of interest to the investigation.

Decision is somewhat supported by the ability to incorporate various facets of the database together with not having to use entire scales, and not being limited to a single scale (this is exemplified in the case scenarios shown previously). The visual representation of the relationships also aids argumentation.

The user’s current view can be iteratively manipulated and this allows improvement based on need. This only improves the current view over the information space and does not alter the underlying data.

Search is not specifically supported by the interface, and would not be a task that could be completed efficiently in this setting because results are shown in terms of inter-object relationships, rather than targeted on the best match. This is also true memorisation. Restructuring is not possible, as there is no way to alter the underlying structures.

6.3.2 Changes from MailSleuth and SurfMachine

Like MailSleuth and SurfMachine, lattice diagrams are the primary means of data display in DSift— in fact, they all use the same lattice rendering software component. Scales in DSift are defined by the database, in much the same way as MailSleuth scales are defined by the SMTP information. The user can access attribute values from these scales. Unlike MailSleuth however, attributes are not keywords entered by the user – instead, they are selected from available attributes associated with the scale.

The most important difference between the previous applications and DSift
is that the realisation of concepts is not performed in any manner other than the creation of a list of object names. **MailSleuth** allows realised concepts to be accessed as email items, and **SurfMachine** realises objects as points on a map. The generic nature of the information used in **DSift** prohibits such meaningful realisations. This is no efficient way of supporting realisation in an application that allows used of any generic data.

The interface of **DSift** has considerably more access to underlying conceptual structures than **SurfMachine**, and allows a more dynamic process of lattice construction than **MailSleuth**. Overall, the interface of **DSift** is less complex than that of **MailSleuth**, but more complex than that of **SurfMachine**. The balance allows the analytical and process oriented tasks, such as *analysis* and *investigation*, to be performed at the cost of simplicity.
ImageSleuth is a tool for navigating collections of annotated images. It combines methods of Formal Concept Analysis for information retrieval, with the graphical information conveyed in image thumbnails. Line diagrams cannot be efficiently utilised when concept extents are represented by thumbnails, and thus other navigation methods are necessary. In addition to established methods such as search and upper/lower neighbours, a query by example function and the possibility to restrict the attribute set are included. Metrics on conceptual distance and similarity are discussed and applied to automated discovery of relevant concepts, from both concepts and semiconcepts.

<table>
<thead>
<tr>
<th>Conceptual Processing Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary</td>
</tr>
<tr>
<td>Exploration</td>
</tr>
<tr>
<td>Recognition</td>
</tr>
<tr>
<td>Identification</td>
</tr>
<tr>
<td>Decision</td>
</tr>
<tr>
<td>Secondary</td>
</tr>
<tr>
<td>Memorisation</td>
</tr>
<tr>
<td>Investigation</td>
</tr>
<tr>
<td>Search</td>
</tr>
<tr>
<td>Not</td>
</tr>
<tr>
<td>Analysis</td>
</tr>
<tr>
<td>Improvement</td>
</tr>
<tr>
<td>Restructuring</td>
</tr>
</tbody>
</table>

Tab. 7.1: CKP Task breakdown for ImageSleuth.

7.1 Domain

Image collections exist in many situations from personal collections of digital photographs to museum archives. The established method for browsing collections of images is to display all images as thumbnails. A thumbnail is a smaller
version of the original image, small enough to view many images simultaneously but large enough to distinguish the main features of the full size image. Within a collection of thumbnails, each thumbnail usually has the same dimensions and is displayed in a two dimensional layout, sorted by a physical feature of the image’s file (e.g. filename, date, filesize, etc).

The desired outcome is to combine thumbnails (a technique that best conveys the content of an image), with the advantages of FCA information retrieval for the annotated information associated with the image. This requires that the concept lattice representation has a different presentation and navigation paradigm compared with that of text documents.

7.1.1 Test Collection

In order to test the usability of ImageSleuth, a sample dataset was needed. A dataset was created from the popular computer game “The Sims 2”. It features 412 objects of household furniture and fittings, described by 120 attributes which include in-game properties, suggestions for use and automatically extracted colour information. There are 7,516 concepts in the lattice derived from the context. Each attribute of the context is assigned to one or more perspectives (similar to scales, further described in 7.2). In this dataset, ten perspectives were constructed.

7.2 Approach

Following an approach used for browsing of text documents, ImageSleuth computes conceptual neighbourhoods from annotated image collections, with images as object and image features as attributes. These features may be information about the depicted object annotated by hand, as well as automatically extracted graphical information. In contrast to most approaches for FCA document retrieval, no line diagram of the lattice is displayed. Instead, following work by Kim et al[KP00, KC01a, KC01b], the user is always located at one concept of the concept lattice. This allows thumbnails of the images to be shown as the
extent of the present concept and thus to convey most of the graphical information characterising this concept. The intent is represented as a list of attributes. As no line diagram of the lattice is shown, lists of upper and lower neighbours are the only representation of the lattice structure around the present concept. Searching and browsing in the image collection then corresponds to moving from concept to concept in the lattice. By including new attributes in the intent, the user moves to a smaller concept where all images in the extent have these features. **ImageSleuth** offers the following possibilities to navigate in the concept lattice:

- Restriction/elaboration of the set of attributes in consideration
- Move to upper/lower neighbour
- Search by attribute(s)
- Search by object(s) (Query by example)
- Search for similar concepts

The possibility to restrict the set of attributes in consideration allows focus on the features that are relevant for the current navigation needs of the user. Attributes irrelevant to the user’s need serve only to increase the number of concepts, making navigation unnecessarily complex. **ImageSleuth** offers predefined sets of attributes (called **perspectives**) covering different aspects of the images. The user may combine these perspectives and include or remove perspectives at any point. Scale attributes are natural candidates for such attribute sets, but other sets are allowed (for example, overlapping perspectives and perspectives which are subsets of other perspectives).

The option to search for similar concepts requires a similarity measure. In order to use this similarity together with the normal search or query-by-example, (where the user may describe the searched concept with an attribute or object set which are not intent or extent of a concept) the similarity measure is defined for semiconcepts as introduced in [LW91] as a generalisation of concepts.
A semiconcept of a context $\mathbb{K} := (G, M, I)$ is a pair $(A, B)$ consisting of a set of objects $A \subseteq G$ and a set of attributes $B \subseteq M$ such that $A = B'$ or $B = A'$. The set of all semiconcepts of $\mathbb{K}$ is denoted by $\mathcal{H}(\mathbb{K})$.

Note that every concept is a semiconcept. The underlying structure of ImageSleuth is thus:

1. A context $\mathbb{K} := (G, M, I)$ with a collection of images as object set $G$, possible features as attribute set $M$ and an incidence relation $I$ assigning features to objects.

2. A collection $\mathcal{P}$ of subsets of $M$ called perspectives. Every subset $A \subseteq \mathcal{P}$ defines a subcontext $\mathbb{K}_A := (G, \bigcup A, I_A)$ with $I_A := I \cap (G \times \bigcup A)$ of $\mathbb{K}$.

3. A similarity measure

$$s : \bigcup_{A \subseteq \mathcal{P}} \mathcal{H}(\mathbb{K}_A)^2 \to [0, 1]$$

assigning to every pair of semiconcepts of a subcontext $\mathbb{K}_A$ a value between 0 and 1 which indicates the degree of similarity.

Since for every $A \subseteq \mathcal{P}$ the contexts $\mathbb{K}_A$ and $\mathbb{K}$ have the same object set and every attribute of $\mathbb{K}_A$ is an attribute of $\mathbb{K}$, it follows for every $m \in \bigcup A$ that $m' = m^{I_A}$. Since for $(A, B) \in \mathfrak{B}(\mathbb{K}_A)$ we have

$$A = B^{I_A} = \bigcap \{m^{I_A} \mid m \in B\} = \bigcap \{m' \mid m \in B\}$$

it follows that $A$ is the extent of a concept of $\mathfrak{B}(\mathbb{K})$. Therefore, $\phi(A, B) := (A, A')$ defines a map $\phi : \mathfrak{B}(\mathbb{K}_A) \to \mathfrak{B}(\mathbb{K})$ and the image of $\phi$ is a $\land$-subsemilattice of $\mathfrak{B}(\mathbb{K})$. In the following, the different navigation means based on this structure are described.
7.2.1 Restriction of the attribute set

By including different perspectives, the user defines a subcontext of $\mathbb{K}$ in which all operations are performed. The user may change this subcontext while browsing, thus obtaining at the present concept further information and search options. If at the concept $(A, A^I)$ the perspective $S \in \mathcal{P}$ is included (i.e. the set of attributes in consideration is increased), then ImageSleuth moves to the concept $(A^{I_{A \cup \{S\}}}, A^{I_{A \cup \{S\}}})$ of $\mathbb{B}(\mathbb{K}_{A \cup \{S\}})$. Since for $A \subseteq \mathcal{P}$ and $S \in \mathcal{P}$ the extent of every concept of $\mathbb{K}_A$ is an extent of $\mathbb{K}_{A \cup \{S\}}$ we have $A = A^{I_{A \cup \{S\}}}$ and the set of images shown does not need to be updated when a further perspective is included. This allows the addition of perspectives during the search without losing information. A similar strategy is known from Toscana systems ([BHS02]) where the user moves through different scales. At every point the user may also remove a perspective $S$ which takes them to the concept $(A^{I_{A \setminus \{S\}}}, A^{I_{A \setminus \{S\}}})$. If in this way an attribute of $A^I$ is removed from the current subcontext then the extent may be increased since $A^I \subseteq A^{I_{A \setminus \{S\}}}$.

7.2.2 Moving to upper and lower neighbours

ImageSleuth uses most of its interface to show thumbnails of images in the extent of the chosen concept. As a result, the user never sees the line diagram of a lattice. Instead, the lattice structure around the current concept is represented through the list of upper and lower neighbours, which allows the user to move to super- or subconcepts. For every upper neighbour $(C, D)$ of the current concept $(A, B)$ the user is offered to remove the set $B \setminus D$ of attributes from the current intent. Dually, for every lower neighbour $(E, F)$ the user may include the set $F \setminus B$ of attributes which takes them to this lower neighbour. By offering the sets $B \setminus D$ and $F \setminus B$ dependencies between these attributes are shown. Moving to the next concept not having a chosen attribute in its intent may imply the removal of a whole set of attributes. In order to ensure that the extent of the given concept is never empty, it is not possible to move to the bottom-most
concept if its extent is empty.

### 7.2.3 Search and Query-By-Example

Browsing of the image collection is achieved by moving to neighbouring concepts. In many cases the user will want to go directly to images having a certain set of attributes $B \subseteq \bigcup \mathcal{A}$. This is offered by the search function which computes, for the selected attributes, the concept $(B^I_A, B^I_A)$, its extent is the set of all images having these attributes, its intent contains all attributes implied by $B$.

Another type of search is performed by the query-by-example function. Instead of defining a set of attributes, a set of objects $A$ is defined as the sample set. The query-by-example function then computes the common attributes of these images (in the selected subcontext) and returns all other images having these attributes by moving to $(A^I_A, A^I_A)$. In this way, query-by-example is the dual of the search function. While the search for images having certain attributes is not affected by the removal or addition of perspectives to the subcontext, query-by-example depends strongly on the selected subcontext. The more attributes taken into consideration, the smaller the set of images that have exactly the same attributes as the examples.

### 7.2.4 Similarity

The aim of query-by-example is to find objects which are similar to the objects in a given sample set. This is a narrow understanding of similarity, implying equivalence in the considered subcontext; for the query-by-example function two objects $g, h$ are “similar” in a subcontext $\mathcal{K}_A$ if $g^I_A = h^I_A$. If the objects are uniquely described by the attributes in the chosen subcontext then query-by-example seldom yields new information. A more general approach is to define a similarity measure. In [Len01] several similarity measures on attribute sets are investigated. Similarity of two objects $g$ and $h$ is then described as the similarity of the attribute sets $g'$ and $h'$. In order to use the grouping of objects provided by the formal concepts, ImageSleuth works with a similarity measure.
7. IMAGESLEUTH 7.2. APPROACH

on semiconcepts which allows the return of a ranked list of similar concepts. Semiconcepts are used, since the set of sample images chosen by the user is not necessarily the extent of a concept. The similarity measure is derived from the following metric:

\[ d((A, B), (C, D)) := \frac{1}{2} \left( \frac{|A \setminus C| + |C \setminus A|}{|G|} + \frac{|B \setminus D| + |D \setminus B|}{|M|} \right). \]

This definition formalises the idea that two semiconcepts are close if there are few objects and attributes belonging to only one of them. In order to compare the number of objects and the number of attributes where they differ, these numbers are set in relation to the total number of objects or attributes. Semiconcepts with small distance are considered similar. ImageSleuth uses \( 1 - d((A, B), (C, D)) \) for scoring purposes – so that smaller distances equate to larger scores.

For a similar purpose Saquer and Deogun introduced in [SD01] a related similarity measure as

\[ s((A, B), (C, D)) := \frac{1}{2} \left( \frac{|A \cap C| + |B \cap D|}{|A \cup C| + |B \cup D|} \right). \]

This definition of similarity extends to semiconcepts \((A, B), (C, D)\) if \( A \cup C \neq \emptyset \) and \( B \cup D \neq \emptyset \). In particular, the similarity \( s((A, A'), (C, D)) \) is defined for every nonempty set \( A \) of objects and every concept \((C, D) \neq (G, \emptyset)\). For a sample set \( A \) of images, ImageSleuth uses a combination of both measures to return a ranked list of concepts similar to the semiconcept \((A, A^{1A})\).

The given metric on semiconcepts has two advantages. First, it allows the return of a list of similar concepts rather than just a list of images. This provides a reasonable grouping of the similar images and, since the attributes of the concepts are displayed, it shows in which way the images relate to the sample.
Second, in contrast to other approaches such as graph distance, the number of different objects of two concepts is taken into account. Instead of counting only the attributes in which two concept intents differ, we assume that the significance of this difference is reflected in the difference of their corresponding attribute sets. If \((A, B)\) is a concept and \((C, D), (E, F)\) are upper neighbours of \((A, B)\) with \(|C| \leq |E|\) then the attributes in \(B \setminus F\) are considered as more characteristic for the concept \((A, B)\) than the attributes in \(B \setminus D\). Thus, if \(|D| = |F|\) then \((C, D)\) is closer to \((A, B)\) than \((E, F)\) even though they differ from \((A, B)\) in the same number of attributes. In this way, even an incomparable concept may be the closest. This contradicts the intuition that, for a concept, its sub- and superconcepts should be closest. Yet upper and lower neighbours are directly accessible by other navigation means. The advantage of the search for similar concepts for a given concept is that it offers a selection of (in the lattice order) incomparable but close concepts which are otherwise invisible.

As the original query-by-example function described above is the dual of a search, this approach can also be used for the search function. If a search is carried out for a set of attributes \(B\), and if \(B'\) is empty, then the concept \((B', B'')\) contains only the information that these attributes do not occur together. No images are returned as a result of this search, since there are no images having the required attributes. In this case, the user may be shown a list of concepts similar to or with small distance to the semiconcept \((B', B)\).

The most common solution to concept searches in FCA, which result in an empty extent, is to offer attributes that can be removed from the search to supply a more general answer which meets a majority of search attributes. Most other forms of search (for example, text search) do not work this way - instead, they supply the user with a list of results that are ranked by a relevance to the query. ImageSleuth tries to address this using the semiconcept search result and a combination of distance and similarity measures (see 7.2.4). When a search is performed that would return the concept with an empty extent, the
user can opt to allow the system to find and rank conceptually relevant concepts. This process is achieved by finding possible neighbours of the semiconcept and performing a bounded traversal which ranks the traversed concepts. These possible neighbours become the first concepts traversed. Each concept visited has its relevance calculated and stored. A test is applied to each concept visited to calculate whether it is to be used for further traversal. The test condition is based on the distance metric compared with a weighted average of the query concepts intent and extent size. The condition is represented as:
7.2. APPROACH

\[ \text{Dist}((A, B), (C, D)) \times \text{SearchWidth} < \frac{1}{2}(|A|/|G| + |B|/|M|) \]

where \((A, B)\) is the query concept and \((C, D)\) is the current concept of the traversal. \text{SearchWidth} is a modifier to allow the search to be made wider or narrower. If the traversal is to continue, the concept’s neighbourhood is added to the traversal list, the concept is marked as visited and the process continues. This algorithm is shown as pseudocode in Fig. 7.3.

Relevance is calculated as the average of the similarity scores which is presented to the user as a percentage.

![Diagram of lattice traversal](image)

\textbf{Fig. 7.2:} An example of lattice traversal starting from a semi-concept. The traversal in this example is complete in 3 steps. The shaded area shows the computed concepts at each step.

The following shows a simple example of ImageSleuth’s semi-concept searching within the example collection. This example uses two perspectives, \textit{Function} and \textit{RoomType}, which have 20 attributes in total. The \textit{Function} perspective is a simple nominal scale with each object having one \textit{function} attribute. The \textit{RoomType} perspective, on the other hand, is more complex with each object having zero or more \textit{room type} attributes. With this context the complete lattice has 194 concepts.
7. IMAGESLEUTH

7.2. APPROACH

Search Traversal
Input: Concept (X, Y), Number Width
Output: Ranked List of Concept Set (not shown)

1. RankedConceptSets := ∅
2. Candidates := UN((X, Y)) ∪ LN((X, Y))
3. Visited := ((X, Y))
4. while |Candidates| > 0
5.    Concept := Candidates[0]
6.    Visited := Visited ∪ Concept
7.    .. compute and store rank information for Concept.
8.    ..
9.    if Dist((X, Y), Concept) × Width < ½(|X|/|G| + |Y|/|M|)
10.   Candidates := Candidates ∪ upperNeigh(Concept)
11.   Candidates := Candidates ∪ lowerNeigh(Concept)
12.   Candidates := Candidates/Visited
13. end if
14. end while

Fig. 7.3: Pseudocode representation of search traversal. Parameters are the starting concept or semiconcept and a numeric value used to modify the width of the search. Output is a ranked list of concept sets – not shown in this example for brevity.

The query for this example will be “Appliances, Electronics, Study”, the first two attributes from the Function perspective and the remaining one from RoomType. Function being nominally scaled, the inclusion of two attributes from this perspective means that if the concept was completed it would result in the empty extent concept or (∅, M). Although this result is technically correct, it does not suit the query’s intention.

To identify a concept that is more representative, a concept traversal is started using the semiconcept, (∅, (Appliances, Electronics, Study)). In this example, the traversal visits 12 concepts, four of which are conceptually close enough to extend the traversal. Consequently, only 6.19% of the total lattice is computed. The first three of five rankings are shown in Fig. 7.4. Relevance is shown as a large percentage, while individual distance and similarity scores are displayed below. Each result is displayed as a list of attributes representing the intent and a collection of thumbnails representing the extent. The highest
ranking concept, with relevance 64.92%, has the intent \((\text{Electronics, Study})\), which is two of the three original query attributes. Following that, at 55.74%, is the concept with the intent \((\text{Bedroom, Electronics, LivingRoom, Study})\). The third ranking, at 54.42% relevance, has two concepts, with the intents \((\text{Appliances})\) and \((\text{Electronics})\), which represent the mutually exclusive elements of the original query.

Fig. 7.4: Results of a concept traversal from the query \("\text{Appliances, Electronics, Study}\) using the perspectives \("\text{Function, RoomType}\)."

7.3 Evolution

The version of \textbf{ImageSleuth} presented here is the second version. The original prototype used concept neighbourhoods and include/remove attributes, but
was limited to traversal between three mutually exclusive subcontexts via single objects. It underwent user-evaluation to test functionality and opinion of ImageSleuth’s navigation paradigm. 29 honours level university students (from various disciplines) were asked to perform tasks and provide feedback on ImageSleuth v1. Results are overviewed in [DVE06b]. Results indicated that concept neighbourhoods offered a useful navigation method, users liked the “grouping of similar objects”\(^1\) (concept extents) and the efficient searching by selection of defined attributes. Negative feedback included complaints about the interface and the systems performance. Analysis of the task results revealed the biggest problem: if a search included mutually exclusive attributes, it returned an empty extent, which left users confused. According to [Coo99], making a user feel stupid is the worst possible software interaction fault.

The second version of ImageSleuth addressed the primary problems experienced by participants in the user testing sessions. These included interface layout, slow performance, inability to combine contexts and the empty extent search result problem. In the first version, include and search functionality was listed after the thumbnails, and users needed to scroll to the bottom of the page to continue navigation. This was repaired by partitioning the page into frames with each frame assigned a set amount of screen space and function. This means a given functionality is always found in the same location regardless of conceptual position in, or content of, the dataset.

### 7.3.1 ImageSleuth and the CKP Tasks

Much of the ImageSleuth interface is devoted to facilitating exploration of images. Screen space is reserved for the sole purpose of providing mechanisms for movement within the conceptual structure. Exploration is bounded to the current perspectives which are selected, and can be extended by addition and removal of perspectives.

By showing only a single concept, recognition and identification tasks are

---

\(^1\) A term used by more than one of the participants.
performed easily. Objects (images) are grouped (as extents) and can be visibly shown to share attributes. Upper neighbours show how the current grouping can be expanded, and how it is defined at this level in the structure. Lower neighbours show how the current group of images can be divided. These allow recognition of the concept’s meaning in relation to its neighbourhood and identification of the objects with its extent.

Decision is supported by the ability to make minimal changes to the current concept via the include and remove functions. Finding an image that optimises certain conditions is also supported by the concept ranking when exact results of an attribute-based search are not available.

Exploration, recognition, identification and decision are the primary tasks enabled by the interface of ImageSleuth. Other tasks that are facilitated to a lesser degree are memorisation, investigation and search.

Memorisation is made easier be the use of thumbnails which make use of the human minds ability to better remember and recognise images than text. Investigation is limited in that only a single concept is shown. Finally, search, while possible, would require a knowledge of how the searched image looks and the attributes it possesses.

Analysis, improvement and restructuring tasks are not possible with ImageSleuth. The single concept view makes analysis very difficult, while improvement and restructuring are not available as ImageSleuth is only a viewer and scales/perspectives and attributes are set.

7.3.2 Changes from MailSleuth, SurfMachine and DSift

The biggest difference between ImageSleuth and the previously presented systems is the removal of the concept lattice as a directly viewable component. It has been replaced by a single concept neighbourhood view, which shows removal and inclusion attributes representative of upper and lower neighbour concepts respectively. This visualisation style allows considerably more complex lattice structures to be visualised, albeit partially. This approach reduces the com-
plexity of the display. A lattice diagram requires interpretation and is limited to only a handful of attributes in order to maintain readability. In the case of ImageSleuth, it also allows maximal screen space for the concept extent to be realised as images.

Like previous systems, ImageSleuth uses scales to group attributes, and like SurfMachine and DSift attributes are restricted by the interface to those within the collection. The conceptual neighbourhood paradigm hides the underlying structures from view and this allows a simpler interface.
8. **SearchSleuth: THE CONCEPTUAL NEIGHBOURHOOD OF A QUERY**

**SearchSleuth** is a form of automatic local analysis that extends the standard web search interface to include a conceptual neighbourhood focused on a concept derived from a query. The conceptual neighbourhood is displayed with upper neighbours representative of a generalisation operation and lower neighbours representative of a specialisation operation. **SearchSleuth** also introduces the notion of a categorisation operation, where the conceptual focus is moved to a sibling concept of the search concept.

Existing FCA-based web search applications also focus on providing automatic local analysis of search results. This is usually done via the creation of a conceptual space from the search results which is displayed in various ways. All these applications fail to create a concept representing the query itself within the space – meaning the space is representative of the results, but not in relation to the query. **SearchSleuth** creates a conceptual space as a neighbourhood of the search concept. This neighbourhood is comprised of generalisations (upper neighbours), specialisations (lower neighbours) and categorisation (incomparable siblings).

By centering the conceptual space around the search concept, the resultant operations are more closely coupled to the search terms used to create the space.

### 8.1 Domain

Internet search is a difficult problem given that the Internet is immeasurably large and constantly being changed and altered, both in content and structure.
8.1. DOMAIN 8. SEARCHSLEUTH

<table>
<thead>
<tr>
<th>Conceptual Processing Tasks</th>
<th>Primary</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Search</td>
</tr>
<tr>
<td></td>
<td>Improvement</td>
</tr>
<tr>
<td></td>
<td>Identification</td>
</tr>
<tr>
<td></td>
<td>Restructuring</td>
</tr>
<tr>
<td>Secondary</td>
<td>Exploration</td>
</tr>
<tr>
<td></td>
<td>Recognition</td>
</tr>
<tr>
<td>Not</td>
<td>Memorisation</td>
</tr>
<tr>
<td></td>
<td>Decision</td>
</tr>
<tr>
<td></td>
<td>Analysis</td>
</tr>
<tr>
<td></td>
<td>Investigation</td>
</tr>
</tbody>
</table>

Tab. 8.1: CKP Task breakdown for SearchSleuth.

Traditionally, Internet search is done by entering one or more keywords into a search engine and then, by reviewing the results, the appropriate web site is found. This process requires a good understanding of the link between the search terms and the results.

Traditional Internet search returns a ranked list of results. Results usually take the form of the URL of the result, its title, a short summary of the document and various details such as date accessed. It is the text-based components of the result that aid in creating a conceptual information space of the results. One problem with the transition from web search results to FCA is that ranking information is lost – all results are treated equally. This issue is usually addressed by reproducing the ordering on any extent that is realised as a set of results.

Another problem with Internet search is that ranking methods use techniques such as link structure, popularity and referring pages. As such, it can not be assumed that all results of a multiple term query will contain all queried terms. Even a single-term query may yield a page that does not contain that term. This may seem counter-intuitive, but if there are enough webpages linking to the result page that do contain that term, that page’s rank may be inflated high enough to feature in the result set.

Like Credo [CR04b], JBrainDead [CGPnV04] and FooCA [Koe06, Koe05], SearchSleuth uses the ‘result has term’ representation for building a formal
context. This means each result from the search is considered an object, and all terms contained in the result’s title and summary are considered attributes. In the case of SearchSleuth (and optionally in FooCA) all words in the result are stop-word filtered and stemmed to their lexical root. This reduces the complexity of the conceptual space and also reduces redundant terms with common lexical roots; ‘car’ and ‘cars’ are best consolidated into a single term.

_Credo_, mentioned earlier in Chapter 3.2, does not include the search terms when creating the context. The reason for this omission is that it is expected that all results contain all query terms. As web search does not behave exclusively as a boolean search for keywords in a document, this assumption may not hold. _Credo_, unlike SearchSleuth, displays two levels of the lattice as a tree with users initially placed at the top-most concept. Display is initially restricted to a single level, but with user interaction a single top-level concept can be expanded. This reduces clutter and also user confusion; users are not confronted with multiple tree branches at the same level with the same label.

_JBrainDead_, mentioned earlier in Chapter 3.2, builds a complete lattice, with query terms, displayed as a tree with users initially placed at the root concept. While this may seem to be an overwhelming amount of information, _JBrainDead_ uses a natural language processing tool to derive pertinent phrases from search results. This reduces the structure of the lattice and gives each concept more information (as it represents an ‘important’ phrase, rather than a single term).

The FooCA application, mentioned earlier in Chapter 3.8, builds and displays the entire context derived from the search results. The context is built without query terms and provides numerous powerful controls over the information space. The user is shown the entire information space in one cross-table. Cross-tables are less than perfect for human interpretation, but the display does show great detail. By viewing the entire information space the user is never positioned without a perspective defined by the query.

SearchSleuth attempts to immerse the user in the information landscape,
8.2. APPROACH

with a perspective centered on the query used to create it. This gives more insight into the meaning of the query with respect to other terms that appear in the result set.

Also, SearchSleuth is the only FCA-based web search that uses multiple searches per query. This is done to expand the information space. The ancillary searches each yield half the results of the main search. This expands the bounds of the information space with more general queries and provides better clustering of facts.

8.2 Approach

SearchSleuth follows from the usability testing of ImageSleuth and employs the same conceptual neighbourhood paradigm for display purposes. Unlike ImageSleuth, SearchSleuth’s context is not static, so the space is rebuilt with each navigation step. This is because computing the entire Internet as a conceptual neighbourhood would be computationally prohibitive. Internet search results are obtained by SearchSleuth via the Yahoo! search server.

The formal context for SearchSleuth is created on demand for each query – this suits the dynamic nature of the Internet. The formal objects are the individual results, and the formal attributes are the terms contained in the title and summary of each result. Terms are extracted from the title and summary after stemming and stop-word filtering has been performed. Stemming reduces words to their lexical root (e.g. jump, jumping and jumps are all reduced to jump). Stop-word filtering removes words without individual semantic value, for example a, the and another. Removing these words reduces the complexity of the context without noticeable reduction in semantic quality.

The context is then reduced by removing attributes with low support. Every attribute that has less than 5% of the objects in the incidence relation is removed. This greatly decreases the computational overhead of most FCA algorithms. This reduction rarely affects the computed conceptual neighbourhood as the terms removed are scarce within the information space.
Once the formal context is constructed, the search concept is created. This is done by taking the provided query terms as attributes and deriving the concept. The upper neighbours of this concept are then derived and used to expand the context. This is done by querying the search engine with the attributes of each upper neighbour and inserting the results into the context. Results for these ancillary searches are limited to fewer results than the primary search.

This process of building the context increases the number of terms in the information space based on a single level of generalisation. This makes the information space larger and richer.

Once the context is expanded, the search concept is recomputed as it may have been invalidated by this process. The upper and lower neighbours are computed next, then the sibling concepts. The explanation of neighbour is found in 2.1 and the computation of neighbours is performed using the algorithm shown in 2.2.2. Sibling concepts are then calculated by finding all of the lower neighbours of upper neighbours which are upper neighbours of lower neighbours. Put another way, siblings are the removal of an attribute that defines an upper neighbour, and the inclusion of an attribute that defines a lower neighbour.

Consider the set of concepts $X$, $UN(X)$ is defined as the union of all upper neighbours of the concepts in $X$.

$$UN(X) := \bigcup\{UN(C) \mid C \in X\}$$

Dually, consider the set of concepts $X$, $LN(X)$ is defined as the union of all lower neighbours of the concepts in $X$.

$$LN(X) := \bigcup\{LN(C) \mid C \in X\}$$

For a lattice with concept, $C$, the set of concepts, $S$ is the siblings of $C$, defined:

$$S := [LN(UN(C)) \cap UN(LN(C))] \setminus \{C\}$$
8.2. APPROACH

Fig. 8.1: Diagram demonstrating the sibling concepts of the concept labelled with a C.

An example is shown in Fig. 8.1; concepts with a grey backing are siblings of the concept marked with a C.

A display is rendered for the user using the same labelling scheme as ImageSleuth, see Chapter 7.2.2, for upper and lower neighbours and using the full intent as labels of sibling concepts. The primary feature of the display is the text entry box in which the query is entered. This text entry box is considered representative of the search concept, and thus is centered in the display.

Upper neighbours are shown above the text entry box, displayed as text labels. The labels represent the attributes which would be removed to navigate to that upper neighbour. These labels are preceded by a minus symbol (−) to reinforce the notion of removal.

Lower neighbours are similarly displayed, but placed below the text entry box. These labels represent the attributes which would be added to navigate to that lower neighbour. Like upper neighbour labels, these labels are preceded by a symbol to reinforce the label’s meaning, namely the plus symbol (+) to indicate the notion of include.

The display order of the upper and lower neighbours is defined by extent size, with the concept with the largest extent displayed first (left-most). Extent is
8. SEARCHSLEUTH

8.2. APPROACH

representative of the importance or prominence of the concept within the current information space. This means the first upper or lower neighbour shown would have the least effect on the extent size of the search concept if the transition was made on a static context. Extent is also used to aid in the colouring of the labels’ background. The higher the extent on a lower neighbour, the deeper the blue behind that concept’s label. Upper neighbours are displayed with the same principle but in red shades.

siblings of the search concept are shown to the right of the text entry box. The complete intent of these concepts is displayed within square brackets proceeded by a tilde symbol (~[...]). This helps group the concept intents and aids in distinguishing between concepts. Unlike upper and lower neighbours, siblings are ordered by similarity using the formula presented in Chapter 7.2.4, while maintaining the highlighting based on extent. The colouring on sibling labels is based on grey shades.

The results of the primary search are shown below the representation of the conceptual neighbourhood.

By clicking on any of the possible concept labels, the query is set to the intent of the selected concept and the query process is restarted. This is an important restructuring step as a change in the query will change the result set, and the information needs to be recomputed to be valid.

An example of the interface is show in Fig. 8.2. The search concept is based on the query formal concept analysis. It shows a single upper neighbour analysis which interestingly shows that formal and concept are implied by analysis. The first of the lower neighbours is the acronym fca. This is followed by terms such as lattice, mathematics and theory. These terms are good examples of specialisation from the concept of formal concept analysis.

The neighbourhood in this example is based on 115 formal objects. The initial number of formal attributes was 623; after reducing the context this was lowered to 40. This offers a tremendous reduction in context complexity, and therefore computation time.
Analysis of the performance of the prototype shows that the vast majority of time taken to display the results of a query are represented in the transfer of search results from the search server. For the page shown in Fig. 8.2, computation of formal concepts took a total of 422ms, while transfer between the *Yahoo!™* search server and the *SearchSleuth* host took 4062ms.

A more detailed example is shown in the next section.

---

**Fig. 8.2:** SearchSleuth display, including top results, after a search for ‘formal concept analysis’.

---

### 8.2.1 An Example Interaction

The following exemplifies a possible navigation of a dynamic information space centered, initially, on the search term ‘tiger’\(^1\).

In Fig. 8.3, it can be seen that the information space has no generalisations or categorisations. This is because the search concept is the top-most concept of the lattice, and therefore all objects have the attribute tiger. Notable specialisations of this search concept are:

\(^1\) Searching for big cat codenames for Mac OS X versions is an unofficial, but established, standard for FCA web-search examples.
Big Cat Adding +cat would probably focus on the Tiger species.
Mac OS Adding +os would probably focus on the operating system (OS) used by Mac computers.
Result Facets +information, feature, +photo and +facts are types of results, and can guide the user.

Please see print copy for figure 8.3

Fig. 8.3: SearchSleuth display after a search for ‘tiger’.

Fig. 8.4 shows the addition of information via specialisation. It can be seen that this has changed the information space surrounding the query with more generic terms, such as +pictures and +site. Also, information about the animal tiger, such as endangered, conserving and white are made available. Categorisations in this space focus on combining tiger with elements found by combining tiger with information, such as tiger species and tiger wild.

Please see print copy for figure 8.4

Fig. 8.4: SearchSleuth display after clicking the ‘+information’ link in Fig. 8.3.

Fig. 8.5 shows the addition of os via specialisation from Fig. 8.3. This specialisation has created an information space focusing on the Apple Mac operating system, in particular version 10.4 which is also known as ‘tiger’. The first two specialisations are representative of the company which makes the OS. The next two specialisations are a reference to the version of the OS. Categorisations in this space tend toward the removal of the term tiger and a focus on os.

Please see print copy for figure 8.5

Fig. 8.5: SearchSleuth display after clicking the ‘+os’ link in Fig. 8.3.
Specialising on cat from Fig. 8.3 gives the conceptual neighbourhood shown in Fig. 8.6.

Fig. 8.6: SearchSleuth display after clicking the ‘+cat’ link in Fig. 8.3.

Specialising on feature from Fig. 8.3 reveals an interesting categorisation shown in Fig. 8.7. The golfer, Tiger Woods, is a category, revealing another possible meaning for the term tiger.

Fig. 8.7: SearchSleuth display after clicking the ‘+feature’ link in Fig. 8.3.

The tiger woods search, shown in Fig. 8.8, has a strong golf focus.

Fig. 8.8: SearchSleuth display after clicking the ‘~[tiger woods]’ link in Fig. 8.7.

8.3 Evolution

SearchSleuth’s design is based on the neighbourhood paradigm presented and tested by the ImageSleuth prototypes (discussed in Chapter 7). The concept of expanding the neighbourhood paradigm with categorisation allows typical include-remove combination to be formed into a single step – a process which is more appropriate with text than images.

In SearchSleuth, the extent is displayed below the control mechanisms rather than centered within it. Also, query by example is not possible as there is no direct access to the index.
Information *Search* is the primary goal of **SearchSleuth**. While the initial search (which is, in essence, a single traditional search) may yield the desired outcome, providing options for rebuilding the space makes query refinement easier. This enables a greater command of the search task.

The result set has a structure built from its inter-result term relationships which is an *improvement* on the result set. By showing prominent and important terms within the results, the value of the results is increased.

By displaying the conceptual neighbourhood, the various meanings or conceptual areas, the query can be used to aid *identification* of the relevant facets and guide further interaction meaningfully.

The interaction of query refinement – be it specialisation, generalisation or categorisation – is a *restructuring* of the information space. Although the restructuring is only visible locally, and not permanently committed to the search engine, the structure the user is exposed to is changed by these interactions.

*Search*, *improvement*, *identification* and *restructuring* are the primary tasks facilitated by **SearchSleuth**. Tasks such as *recognition* and *exploration* are also supported to a lesser degree.

Conceptual clusters surrounding the query concept, in particular specialisations, offer information confirming the query is *correct* in terms of user expectations. If a user recognises the space surrounding the query concept, then the space meets their expectations. This *recognition* implies that the search terms were closely coupled with the results.

*Exploration* appears to be enabled in this interface, but as it is performed over an ever changing landscape, it can not be considered real *exploration*. Each navigation step or query refinement changes the bounds of the information space.

*Decision*, *memorisation*, *analysis* and *investigation* are not supported by the interface.
8.3.1 Changes from MailSleuth, SurfMachine, DSift and ImageSleuth

The conceptual neighbourhood paradigm used in ImageSleuth is used in SearchSleuth. Unlike, ImageSleuth, the intent, as opposed to the extent, is displayed centrally. This is because the intentional focus of SearchSleuth; keywords are the more defining of the concept. The inclusion of incomparable siblings adds extra depth to the neighbourhood.

SearchSleuth does not make use of scales, perspectives or other types of attribute groups. The input is keywords and the context is constructed using terms derived from the web search results. All previously presented software application used some form of attribute grouping.

SearchSleuth concepts are never realised. All objects are displayed as a ranked list. The search engine uses algorithms based on the complete documents to produce each ranked list, so disregarding this ordering makes little sense.
9. CONCLUSION

The five software applications presented in this thesis each enable several of Wille’s 10 Tasks of Conceptual Information Systems. These tasks are facilitated by various aspects of the interfaces and the exposure of the conceptual structures which underlie each application. While using these five applications as the basis for connecting features of the interface to tasks may be statistically weak, some general implications between what an interface allows a user to do and what they can achieve can be drawn.

Each chapter has presented a different software application with a different task-oriented focus. These applications each offer a different form of access from the interface to the underlying conceptual structures used to process data. The exposure of these structures via the interface is discussed now.

9.1 Exposure of Conceptual Structures

Primarily, the conceptual structure exposure of an application can be classified using the following categories:

- *Lattice View/ Neighbourhood View*
- *Conceptual Scales*
- *Meaningful Concept Realisation*
- *Limited Attribute Vocabulary*
- *Intentional Focus*
- *Extensional Focus*
9.1. EXPOSURE OF CONCEPTUAL STRUCTURES

9.1.1 Lattice View / Neighbourhood View

Concepts can be shown as either a part of a lattice or as part of a neighbourhood. In a Lattice View, the interactions between all concepts are displayed. In a Neighbourhood View, a single concept’s relationship to other concepts is emphasised (that is, a ‘current’ concept’s ‘neighbourhood’ is shown).

MailSleuth, SurfMachine and DSift all use a Lattice View, showing all concepts arranged in a lattice diagram. This allows a complete view over the current information space. The size of the lattice is a limiting factor when visualising complete lattices – a large number of concepts can lead to long processing times and unreadable diagrams.

Conversely, ImageSleuth and SearchSleuth both use a Neighbourhood View, positioning the user at a single concept and only displaying the surrounding neighbourhood of that particular concept. This allows complex lattices to be used, with the constraint that only a small part of the entire lattice is presented at any one time. In the case of ImageSleuth, all upper and lower neighbours are displayed because the user can simplify or enhance the information space by modifying the current perspectives. SearchSleuth, however, uses such a complex lattice that the number of upper and lower neighbours displayed is limited to a predefined number. SearchSleuth also enriches the Neighbourhood View with the addition of sibling concepts.

9.1.2 Conceptual Scales

Conceptual Scales are named groups of attributes. Groups usually represent a facet of a context. For instance, looking at the Chapter 2 planets example (context shown in Table 2.1), three possible scales are Size, Distance and Moon.

By grouping attributes under logical headings, particular attributes are easier to find. This helps to focus the information space on an area of interest.

MailSleuth uses scales derived from email header fields to target querying – the queries become the attributes of the context. The attributes of each scale
are derived from the collection. **DSift** takes the same approach, this time using database schema to base scales on.

**SurfMachine** has a well defined domain, and therefore scaling is based on domain knowledge.

**ImageSleuth** uses scales (referred to as perspectives) in a very different manner, but derives them in much the same way as **DSift**. Attributes can be associated with any number of perspectives and the connection between an attribute and a perspective is decoupled. However, perspectives have the same purpose as scales, in that they are still used to control the focus of the information space.

**SearchSleuth**, on the other hand, has no scales. Attributes in **SearchSleuth** are terms derived from plain text, and as such are not able to be categorised.

### 9.1.3 Meaningful Concept Realisation

To realise a concept is to show its object set (extent or extent contingent). For a concept realisation to be meaningful, it should be displayed in a manner suiting the represented objects.

When a concept’s extent is displayed in **MailSleuth**, the objects are shown as emails of the collection, appearing in a folder in the same way as in any email application. This allows concepts to be browsed much like an ordinary email folder. **SurfMachine**’s objects are surfing locations, so displaying objects as locations on a geographic map allows the objects to be compared spatially. **ImageSleuth** uses thumbnails to display the current extent of the focal concept. This is an appropriate, established representation method for groups of images. **SearchSleuth** represents search result objects by title, URL and summary – another established and appropriate display method.

**DSift** is the exception and does not represent objects in a meaningful way. **DSift** allows input of any simple database, with objects as textual fields. As such, the objects do not necessarily have a ‘type’. Without knowing what the
textual label of an object represents, there is no way to realise it in any way other than a list.

9.1.4 Limited Attribute Vocabulary

A Limited Attribute Vocabulary allows users to select from a list of attributes known to be present in the data or domain. SurfMachine uses a Limited Attribute Vocabulary that is dictated by the restricted nature of the domain. There are only eight possible directions (points on the compass) for each of the two domain facets – wind and swell.

DSift uses a Limited Attribute Vocabulary for each scale, defined by the values in the database column corresponding to that scale. Similarly, ImageSleuth uses a vocabulary that is defined by the collection.

MailSleuth does not have a Limited Attribute Vocabulary. This is because keywords can be used to define queries, which in turn become attributes. A keyword that is not present in the collection as an attribute can be used, and could potentially become an attribute of the collection as the underlying mail collection changes.

Like MailSleuth, SearchSleuth uses keywords and as such it is infeasible to predetermine or calculate all possible keywords. Attributes of the context are not known until the user enters the attributes (search terms) that the context is based upon.

9.1.5 Intentional Focus

An interface with an Intentional Focus is geared towards interaction with the intent of concepts or attributes of the underlying data. For example, MailSleuth allows the creation of attribute hierarchies and devotes considerable attention to attribute query design. DSift also has a strong Intentional Focus, with two modalities for attributes affecting the current information space and automatically generated scales. SearchSleuth’s focus is also intentional, with the current information space and all neighbours represented as attributes.
SurfMachine and ImageSleuth have less focus on the intent of concepts, and instead place more emphasis on the display and information content of objects.

9.1.6 Extensional Focus

An interface with an Extensional Focus is geared towards interaction with the extent of concepts or objects of the underlying data. MailSleuth can be considered to have both an Intentional Focus and an Extensional Focus. Extensionally, MailSleuth uses meaningful realisation methods for objects, and is positioned within the object set itself (as a plug-in).

SurfMachine has an Extensional Focus, exemplified by the geographic realisation method which occupies a considerable amount of the interface. Likewise, ImageSleuth devotes the majority of its interface to the display of objects as image thumbnails. ImageSleuth’s use of query-by-example allows objects to be used as input to a search to find other objects.

DSift does not focus on the object set directly and only has a simple realisation method. SearchSleuth also devotes little attention to the object set, which in this case is displayed below the attribute-based conceptual neighbourhood of the query.

9.1.7 Interface Exposure

A breakdown of how the various applications presented in this thesis display their conceptual structures is shown in Table 9.1. This table is the basis for a formal context, the concept lattice of which is shown in Fig. 9.1. The concept lattice allows some insight into the relationships between these interface facets (at least in terms of this small sample of applications). From the concept lattice it can be seen that:

- Lattice View and Neighbourhood View are mutually exclusive
- Meaningful Concept Realisation is needed for an Extensional Focus
9.2. TEN TASKS

- *Meaningful Concept Realisation* is needed for a *Neighbourhood View*

- *Conceptual Scales* are needed for a *Lattice View*

- *Conceptual Scales* are needed for a *Limited Attribute Vocabulary*

- *Conceptual Scales* are needed for an *Extensional Focus*

- All presented applications represent different and non-inclusive groups of exposure types

While the lattice shown in Fig. 9.1 indicates that *Lattice View* and *Neighbourhood View* are mutually exclusive, it would not be impossible for an application to offer a combination of the two.

*Meaningful Concept Realisation* is needed for both *Extensional Focus* and *Neighbourhood View*. *Meaningful Concept Realisation* displays the object extent in a suitable manner, which forms the basis for an application with an *Extensional Focus*. The display method for a *Neighbourhood View* allows the current concept’s extent more screen area within the application in order to show the realised concepts. While this is not necessarily a strict rule, it is an unsurprising implication.

*Conceptual Scales* are needed for *Lattice View*, *Limited Attribute Vocabulary* and *Extensional Focus*. *Conceptual Scales* allow control over the complexity of the produced lattices (in most cases) and allow for semantic grouping of the attributes used to direct the creation of lattices used in a *Lattice View*. When there is a *Limited Attribute Vocabulary*, it is sensible to group attributes into *Conceptual Scales* (wherever possible). *Conceptual Scales* simplify the attribute selection process allowing more emphasis on the objects of the collection; and thus allowing an *Extensional Focus*.

9.2 Ten Tasks

The ten tasks presented by Wille are enabled by the various applications presented in this thesis. A summary (in the form of a context) of the primary tasks...
9. CONCLUSION 9.2. TEN TASKS

Lattice View
Neighbourhood View
Conceptual Scales
Meaningful Concept Realisation
Limited Attribute Vocabulary
Intentional Focus
Extensional Focus

<table>
<thead>
<tr>
<th></th>
<th>Lattice View</th>
<th>Neighbourhood View</th>
<th>Conceptual Scales</th>
<th>Meaningful Concept Realisation</th>
<th>Limited Attribute Vocabulary</th>
<th>Intentional Focus</th>
<th>Extensional Focus</th>
</tr>
</thead>
<tbody>
<tr>
<td>MailSleuth</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>SurfMachine</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DSift</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ImageSleuth</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>SearchSleuth</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Tab. 9.1: Breakdown of conceptual structure exposure types in the five applications presented.

Fig. 9.1: Concept lattice corresponding to the context shown in Table 9.1.
for each application is presented in Table 9.2. This context is visualised as a concept lattice in Fig. 9.2.

<table>
<thead>
<tr>
<th></th>
<th>exploration</th>
<th>search</th>
<th>recognition</th>
<th>identification</th>
<th>analysis</th>
<th>investigation</th>
<th>decision</th>
<th>improvement</th>
<th>restructuring</th>
<th>memorisation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MailSleuth</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
<td>×</td>
<td></td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>SurfMachine</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>×</td>
<td></td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>DSift</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
<td>×</td>
<td></td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>ImageSleuth</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>SearchSleuth</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
</tbody>
</table>

Tab. 9.2: Summary of the primary tasks of Conceptual Knowledge Processing enabled by the five applications presented.

All of the presented applications represent different and non-inclusive groups of tasks. This is good, as although the sample size is small, the applications all have different levels of conceptual structure exposure and enable different tasks, thus enabling a richness of data to draw implications from. Many implications
are possible from the lattice shown in Fig. 9.2, some of the more interesting are:

- **Search** requires **Identification**

- **Recognition** requires both **Exploration** and **Identification**

- **Recognition** is the culmination of **Exploration** and **Identification**

- **Analysis** requires **Recognition**

- **Decision** requires **Exploration**

- **Improvement** and **Restructuring** are equivalent

- **Memorisation** requires **Decision**

- All presented applications represent different and non-inclusive groups of tasks

Several natural implications are shown in Fig. 9.2. One of these is that **search** requires **identification**; **identification** is an inherent part of **search**, as **search** is the locating of a known item - if the item cannot be identified then it cannot be located. Another natural implications is that **recognition** is the culmination of **exploration** and **identification**. If the user can identify their location within the space and then move about (explore) within it – they should be able to recognise the relationship between those locations.

**Analysis** requires **recognition**, because analysing information cannot be performed without recognising the objects or concepts involved in the analysis. Similarly, **decision** has a requirement of **exploration**. This is to be expected because being able to explore various options allows the decision making task.

As well as expected implications, there are some unexpected ones. The equivalence between **improvement** and **restrictucturing** is one example. While these two tasks are similar, they should not be equivalent. This shows that the five applications presented do not distinguish between these two tasks. This is also true for the implication between **memorisation** and **decision**; **memorisation**
should not rely on decision. These three tasks (improvement, restructuring and memorisation) are only represented by one application each and this is the most likely reason for the unexpected relationships.

When the two contexts representing conceptual structure exposure and tasks of the applications are combined, the lattice shown in Fig. 9.3 is produced. From this lattice, some implications about what an interface allows a user to do and what they can achieve can be drawn.

Some notable implications from the lattice shown in Fig. 9.3 (ignoring those discussed previously) are:

- **Search** needs *Meaningful Concept Realisation*
- **Search** needs *Intentional Focus*
- **Decision** needs an *Extensional Focus*
- **Decision** needs a *Limited Attribute Vocabulary*
- **Analysis** needs a *Lattice View*
- **Analysis** needs an *Intentional Focus*
- **Recognition** needs *Conceptual Scales*
9. CONCLUSION 9.2. TEN TASKS

- Intentional Focus always allows identification
- Neighbourhood View always allows identification
- Conceptual Scales always allows exploration
- Limited Attribute Vocabulary always allows exploration
- Lattice View always allows exploration
- Extensional Focus always allows exploration

This reveals several interesting implications. Search requires Meaningful Concept Realisation because potential results should be easily accessible to the user. Search is also driven by being able to describe the item being searched for and therefore it is not surprising to see that search requires an Intentional Focus.

The Decision task requires both an Extensional Focus and a Limited Attribute Vocabulary. Extensional Focus emphasises the objects in the data, enabling comparison between options. Providing that the vocabulary in use is appropriately rich to enable a decision, a Limited Attribute Vocabulary reduces the possible complexity of the information space, allowing a choice between objects.

Analysis requires a Lattice View and Intentional Focus to be enabled in an application. This is logical as the Lattice View shows the relationships between concepts, where as a Neighbourhood View has a more limited scope of the information space. An Intentional Focus gives emphasis to the attributes which guide the analysis.

According to the lattice, Recognition requires interface support for Conceptual Scales. This may not be true in a general sense, and is an interesting implication which could indicate that organisation of attributes increases the semantic value of individual attributes allowing recognition.

By having an interface with Intentional Focus, the identification task is enabled. This is because an intentionally focussed application is geared towards
attributes which are used to describe objects, thus aiding identification. A Neighbourhood View also implies that identification is enabled. This is logical as the Neighbourhood View shows the current concept in terms of its surroundings, allowing its taxonomic position to be clearly shown in relation to neighbouring concepts.

Conceptual Scales, Lattice View, Limited Attribute Vocabulary and Extensive Focus all allow exploration. Exploration is a very general task and these four conceptual structure exposure types may not be the only ones that enable exploration. In fact, all exposure types would allow exploration if the SearchSleuth application was not included in this analysis. SearchSleuth does not facilitate exploration because any navigation step rebuilds all underlying information.

9.3 Conclusion

This thesis has described and analysed the conceptual knowledge tasks and how interface exposure to conceptual structures enables these tasks via case examples based on MailSleuth, SurfMachine, DSift, ImageSleuth and SearchSleuth. The trade off between exposure and ability to perform tasks has been described. This shows that when building Formal Concept Analysis-based applications there are high-level, interface design patterns which enable Conceptual Knowledge Processing tasks for the user.
APPENDIX
A. SOURCES

This thesis is the monograph of many published works by the author. This appendix acts as a guide to the source publications for the chapters of this thesis.

A.1 Chapter 2: Formal Concept Analysis

Discussion of lattice diagram layout, in particular what constitutes a good layout and heuristic layout algorithms are derived from:


A.2 Chapter 4: MailSleuth

Usability test results and evolution of the MailSleuth interface derived from:


A.3 Chapter 5: SurfMachine

SurfMachine is presented in:
A.4. Chapter 6: DSIFT

DSIFT is presented in:


A.5. Chapter 7: ImageSleuth

Sourced from the following conference papers and journal papers (which cover the various aspects of ImageSleuth):


A. SOURCES

A.6. CHAPTER 8: SEARCHSLEUTH

• Jon Ducrou, Bjorn Vormbrock, and Peter Werner Eklund. Browsing and searching mpeg-7 images using formal concept analysis. In International Conference on Artificial Intelligence and Applications. ACTAPress, 2006.


A.6 Chapter 8: SearchSleuth

SearchSleuth is presented in:

• SearchSleuth: The Conceptual Neighbourhood of an Internet Query: J. Ducrou and P. W. Eklund: in Proceedings of the Fifth International Conference on Concept Lattices and Their Applications (CLA’07), Springer, 2007, Accepted To Be Published.
The following is the formal context for the **SurfMachine**.

Note: Sharkies, Fisho’s and Lake Illawarra Entrance. The description for Fisho’s in the guide says “a right-hand reef break that breaks several times a decade” which tells us something but nothing about the Wind or Swell conditions. Likewise Sharkies, “short shallow left reef, break named after shark attack 30 years ago”. Lake Illawarra Entrance also contains no extractable information. This explains why there are no attributes for these objects and helps underline the difficulty of the information extraction problem in texts of this sort.

<table>
<thead>
<tr>
<th>Location</th>
<th>NW Wind</th>
<th>NE Wind</th>
<th>SW Wind</th>
<th>SE Wind</th>
<th>N Wind</th>
<th>NE Swell</th>
<th>SW Swell</th>
<th>E Swell</th>
<th>S Swell</th>
<th>NW Swell</th>
<th>NE Wind</th>
<th>SE Wind</th>
<th>E Wind</th>
</tr>
</thead>
<tbody>
<tr>
<td>Garie Beach</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Burning Palms</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Stanwell Park</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Coalcliff</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Scarborough</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Wombarra</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Coledale</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Sharkies</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Headlands</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Austinmer</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Thirroul</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Sandon Point</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Peggy’s Point</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Woonona</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bellambi</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Corrimal</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>East Corrimal</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>
## B. SURFMACHINE CONTEXT

<table>
<thead>
<tr>
<th>Location</th>
<th>Towradgi</th>
<th>Pucky's</th>
<th>North Wollongong</th>
<th>City Beach Wollongong</th>
<th>Bastian’s</th>
<th>Shitties</th>
<th>MM Reef</th>
<th>MM Beach</th>
<th>MM Bay</th>
<th>Fisho’s Reef</th>
<th>Stoney’s</th>
<th>Port Reef</th>
<th>Port Beach</th>
<th>Lake Illawarra Entrance</th>
<th>Sharkies (Windang)</th>
<th>Windang Island</th>
<th>Warilla Beach</th>
<th>Barrack Point</th>
<th>Madman’s</th>
<th>Suck Hole</th>
<th>Cowries</th>
<th>Pools / The Bombie</th>
<th>South Shellharbour</th>
<th>The Shallows</th>
<th>Redsands</th>
<th>The Farm</th>
<th>Minnamurra</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
C. ImageSleuth v1 QUESTIONS

C.1 Test Script

The test script consists of three sections the first two of which returned quantitative results and the third qualitative results. The first section included directions to be followed using Windows Explorer to browse the image collection. These tasks included finding particular items, finding items that matched certain criteria, observations of item groups with certain features and so on.

The second section of the test started with identical tasks to those in the first but now completed using ImageSleuth. As the study supervisors were not permitted to assist participants, the steps to be followed using ImageSleuth were ordered in such a way as to expose participants to the various functionality of ImageSleuth gradually. Designing the test script in this way assists the participant in learning the new navigation style without needing special training.

Tasks in the latter half of the second section of the test script were designed to make participants perform more complex interactions in order to solve problems. For example, showing a black and white image of an object in a setting at a different orientation and asking participants to identify its colour by finding the corresponding image in ImageSleuth. These tasks extended the test script and had no comparable task in the first section of the script. A complete list of the second sections tasks are shown below.

The first and second sections were issued in reverse order to half of the participants. This way, familiarity with the object set did not give an unfair advantage to ImageSleuth.
C.1. TEST SCRIPT

C.1.1 Part A: Windows Explorer

Using *Windows Explorer*, attempt to perform the following tasks. For each task specify whether the task was achievable.

1. Identify how many images are pink?
2. There are 2 beds with a high level of horizontal edges. What are they?
3. Are there more red, blue or brown images?
4. Find the name of the chair in the following image:

![Chair Image](image1.png)

Shown in colour.

5. Find the name of the bed in the following image:

![Bed Image](image2.png)

Shown in colour.
6. Find the name of statue in the following image. What colour hat does it have?:

![Statue](image)

Shown in black and white.

7. This object is not in The Sims 2\textsuperscript{TM} collection. Find one image that is similar.

![Similar Object](image)

Shown in colour.
C.1. TEST SCRIPT  C. IMAGESLEUTH V1 QUESTIONS

C.1.2  Part B: ImageSleuth

Using ImageSleuth, attempt to perform the following tasks. For each task specify whether the task was achievable. After you have completed each question, please click on the ImageSleuth Logo at the top of the page to return to the home page.

1. Identify how many images have the environment attribute.

2. Identify how many images have a price higher than 5000.

3. Search for all images that are Decorative. Then include the outside room type. How many images are there?

4. Identify how many images can be used in both the Dining Room and Kitchen.

5. Identify how many of the plumbing images also have an environment attribute. What types of objects are these?

6. How many images do not have a price at all?

7. Identify how many images are green using the colour properties.

8. There are 2 beds with a high level of horizontal edges. Which are they?

9. Are there more red, navy or green images?

10. Find the name of the chair in the following image:

![Chair Image](image-url)  
Shown in colour.
11. Find the name of the bed in the following image: (Note - The bedspread need not be the same)

![Bed Image](image1)

Shown in colour.

12. Find the name of the bath/shower in the following image. What colour is the curtain around it?

![Bath Image](image2)

Shown in black and white.

13. This object is not in The Sims 2™ collection.
It has attributes of: Comfort = 5, Energy = 4, Environment = 3.

Find one image that is similar in design and one that has similar attributes while not being expensive.

14. Find the object that can be used outside, builds logic and has a price between 1000 to 5000. How many images have similar colours?

15. Aspiration rewards and career rewards are special objects in The Sims 2\textsuperscript{TM}. Using ImageSleuth to browse the objects, what can you say about them, in 50 words or less?

The final section of the test script consisted of a “free exploration” of ImageSleuth: encouraging participants to discover features without any particular goal in mind. This allowed participants to gain an understanding of the features without explicit direction. Participants were subsequently asked to provide their positive and negative thoughts regarding the features of the program.
C.2 Survey

The survey asked participants questions on their personal background and experience with ImageSleuth. Background information included the faculty of study, experience with other image browsers/viewers and the methods of organisation used for personal image collections. Likert scales were used collect details of their experience with ImageSleuth and Windows Explorer. This was followed by a series of questions to assess the participants understanding of ImageSleuth and how it worked.

C.2.1 Participant Survey

• Short Response

  - Which faculty does your university degree belong?
  
  - Are you colour blind? If so, did you experience difficulty in completing the test script?
  
  - Have you used image management applications before? If so, which?
  
  - Do you already sort your images based on specific criteria (e.g. date, location, etc.)? If so, what?

• Likert Scale Statements (0 to 10, disagree to agree)

  - I am familiar with the PC Game The Sims 2™.
  
  - I found it easy to complete tasks with Windows Explorer
  
  - I found it easy to complete tasks with ImageSleuth
  
  - I feel that ImageSleuth has a strong advantage over Windows Explorer.
  
  - I feel that ImageSleuth has a strong advantage over other image browsers.
C.2. SURVEY

C. IMAGESLEUTH V1 QUESTIONS

- **ImageSleuth** allows me to recognise relationships between images that I may not have noticed previously.

- **ImageSleuth** is a tool that gives more power over searching and browsing catalogs of images.

- I found that the ***properties were accurate.
  * colour
  * Edge

- My overall experience with **ImageSleuth** was a positive one.

• Multiple Choice

- What features of **ImageSleuth** did you find assisted most when completing the tasks in the test script? (circle all that apply)
  * Including and removing attributes for searching
  * Ability to search images
  * Graphics and overall design of **ImageSleuth**
  * None of the above

- What features of **ImageSleuth** made it difficult to complete the tasks in the test script? (circle all that apply)
  * Including and removing attributes for searching
  * Ability to search images
  * Graphics and overall design of **ImageSleuth**
  * None of the above

• Long Response

- Which features of **ImageSleuth** could be used to improve the image browsing experience in the future, and why?
– In your own words, describe the 4 main components of the ImageSleuth interface and what they do. Name the 3 different types of searches and what they do?

– Do you understand what the Remove(up) and Include(Down) sections mean in ImageSleuth?

– In your own words, please describe what the Include and Remove sections allowed you to do, and comment on whether or not this tool helped you to complete the allocated tasks in the test script.

– Could you see this application being used in the real world? If so, where?

– Do you have any other comments?
<table>
<thead>
<tr>
<th>Reference</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>[BHS02]</td>
<td>Peter Becker, Joachim Hereth, and Gerd Stumme</td>
<td>ToscanaJ - an open source tool for qualitative data analysis. In *Advances in For-</td>
</tr>
</tbody>
</table>
mal Concept Analysis for Knowledge Discovery in Databases. Proc.
Workshop FCAKDD of the 15th European Conference on Artificial
Intelligence (ECAI 2002). Lyon, France., 2002.

1940. First edition, Colloquium Publications.

[CB04] Richard Jeffrey Cole and Peter Becker. Agreement contexts in
formal concept analysis. In Peter Eklund, editor, *Concept Latt-
tices: Proceedings of the Second International Conference on For-
mal Concept Analysis, ICFCA 2004*, LNAI 2961, pages 96–103,

[CB05] Richard Jeffrey Cole and Peter Becker. Navigation spaces for the
conceptual analysis of software structure. In Bernhard Ganter and
Robert Godin, editors, *Proc. of the 3rd Int. Conference on Formal
Concept Analysis*, LNAI. Springer-Verlag, 2005.

[CDE06] Richard Jeffrey Cole, Jon Ducrou, and Peter Werner Eklund. Au-
tomated layout of small lattices using layer diagrams. In *Proc.
of the 4th Int. Conference on Formal Concept Analysis*, volume

[CE99] Richard Jeffrey Cole and Peter Werner Eklund. Analyzing an email
collection using formal concept analysis. In *European Conf. on
Knowledge and Data Discovery, PKDD’99*, number 1704 in LNAI,

[CES00] Richard Cole, Peter Eklund, and Gerd Stumme. CEM - a program
for visualization and discovery in email. In *Principles of Data Min-
ing and Knowledge Discovery: 4th European Conference, PKDD
2000.


180
Bibliography


