

# A Web of Trails

by

Richard Wheeldon  
richard@dcs.bbk.ac.uk

A thesis submitted in conformity with the requirements  
for the degree of Doctor of Philosophy  
School of Computer Science and Information Systems  
Birkbeck College, University of London

# Contents

<b>I</b>	<b>The Navigation Problem and Related Issues</b>	<b>14</b>
<b>1</b>	<b>Introduction</b>	<b>15</b>
1.1	Abstract . . . . .	16
1.2	Motivation . . . . .	18
1.3	Outline of the Thesis . . . . .	19
1.4	Acknowledgements . . . . .	20
<b>2</b>	<b>Problems and Solutions</b>	<b>21</b>
2.1	Introduction . . . . .	22
2.2	A Brief History of Hypertext . . . . .	23
2.3	The World Wide Web . . . . .	29
2.3.1	Structure of the Web . . . . .	30
2.4	Resource Discovery . . . . .	33
2.5	Information Retrieval Techniques . . . . .	36
2.5.1	Document Representation . . . . .	36
2.5.2	Scoring Metrics . . . . .	37
2.5.3	Evaluation Metrics . . . . .	39
2.5.4	The Text REtrieval Conference . . . . .	40
2.6	Web Page Metrics . . . . .	43
2.6.1	HTML Tag Weighting . . . . .	43
2.6.2	Landmark Nodes . . . . .	44
2.6.3	Hubs and Authorities . . . . .	44
2.6.4	PageRank . . . . .	45
2.6.5	Combining Metrics . . . . .	46
2.7	Non-Linear Search . . . . .	47

2.7.1	Question Answering . . . . .	47
2.7.2	Category-based Clustering . . . . .	47
2.7.3	Link-based Clustering . . . . .	49
2.8	The Navigation Problem . . . . .	50
2.9	Navigation Aids . . . . .	51
2.9.1	Link Suggestion . . . . .	51
2.9.2	Site Maps . . . . .	51
2.9.3	Trees and Graphs . . . . .	57
2.10	Trail Recording . . . . .	62
2.11	Trail Finding . . . . .	64
2.12	Summary . . . . .	67
<b>II</b>	<b>Implementation of a Trail-Based Navigation Engine</b>	<b>68</b>
<b>3</b>	<b>The Best Trail Algorithm</b>	<b>69</b>
3.1	Introduction . . . . .	70
3.2	Graph Traversal and Path Finding . . . . .	73
3.3	The Best Trail Algorithm . . . . .	75
3.4	Auxillary Functions . . . . .	78
3.5	Scoring Trails . . . . .	80
3.6	Sorting and Filtering . . . . .	83
3.7	Implementation . . . . .	86
3.8	Complexity . . . . .	89
3.9	Performance Evaluation . . . . .	90
3.10	Concluding Remarks and Future Work . . . . .	96
<b>4</b>	<b>Navigability and Starting Point Selection</b>	<b>97</b>
4.1	Introduction . . . . .	98
4.2	Potential Gain and Related Metrics . . . . .	100
4.2.1	Potential Gain and Gain Rank . . . . .	100
4.2.2	Discounting Functions . . . . .	101
4.3	Computing Potential Gain . . . . .	102
4.4	Experiments . . . . .	104

4.4.1	Convergence . . . . .	104
4.4.2	Power Law Distributions . . . . .	104
4.5	Correlations between Ranking Metrics . . . . .	106
4.5.1	Experimental Methods . . . . .	106
4.5.2	Discussion . . . . .	107
4.6	Improving Starting Point Selection . . . . .	108
4.7	Concluding Remarks and Future Work . . . . .	112
4.7.1	Query Specific Potential Gain . . . . .	112
4.7.2	Multi-Metric Combinations . . . . .	112
4.7.3	Site-based Potential Gain . . . . .	112
<b>5</b>	<b>Architecture of a Navigation Engine</b>	<b>114</b>
5.1	Introduction . . . . .	115
5.2	Top level Overview . . . . .	116
5.3	Webcases . . . . .	118
5.4	An Extensible Component Architecture . . . . .	120
5.4.1	TrailAlgorithm subclasses . . . . .	121
5.4.2	Post-Processing and Index Creation . . . . .	124
5.5	Advanced Features . . . . .	131
5.6	Improving File-Type Recognition . . . . .	133
5.7	Web Page Summaries . . . . .	135
5.7.1	A Summarization Algorithm . . . . .	136
5.7.2	Examples . . . . .	137
5.7.3	Implementation . . . . .	138
5.7.4	Performance Analysis . . . . .	139
5.7.5	Titles and Short Titles . . . . .	143
5.8	Concluding Remarks and Future Work . . . . .	145
5.8.1	Summaries . . . . .	145
5.8.2	Multi-page Search . . . . .	145
5.8.3	Partial Collection Ranking . . . . .	145
5.8.4	Incremental Crawling and Merging Webcases . . . . .	146

<b>III Applications for Trail-Discovery</b>	<b>147</b>
<b>6 Navigating the Web</b>	<b>148</b>
6.1 Introduction . . . . .	149
6.2 Navigation Interfaces . . . . .	150
6.2.1 NavSearch . . . . .	150
6.2.2 TrailSearch . . . . .	153
6.2.3 VisualSearch . . . . .	154
6.3 Web Site Examples . . . . .	155
6.3.1 SleepyCat . . . . .	155
6.3.2 University College London . . . . .	158
6.3.3 Birkbeck . . . . .	159
6.4 Case Study – SCSIS . . . . .	161
6.5 Mat-Hassan and Levene’s User Study . . . . .	172
6.6 Comparative Testing . . . . .	173
6.6.1 Evaluation Philosophy . . . . .	173
6.6.2 Analysis of Queries . . . . .	174
6.6.3 Conclusions . . . . .	202
6.7 Scaling to the Web . . . . .	204
6.7.1 Splitting the Index . . . . .	204
6.7.2 Graph Partitioning . . . . .	205
6.7.3 Merging Results . . . . .	206
6.8 Concluding Remarks and Future Work . . . . .	209
<b>7 Search and Navigation in Database Systems</b>	<b>211</b>
7.1 Introduction . . . . .	212
7.2 Indexing Relational Databases . . . . .	215
7.2.1 Translating a Relation to a Full-Text Index . . . . .	215
7.2.2 Generating the Link Graph . . . . .	215
7.2.3 Computing Joins with Trails . . . . .	216
7.3 Extending the Navigation System . . . . .	217
7.4 Semi-Structured Data and XML . . . . .	219
7.5 Query Expressiveness . . . . .	220

7.6	Examples . . . . .	221
7.7	Evaluation . . . . .	224
7.8	Hierarchies, Taxonomies and Ontologies . . . . .	226
7.8.1	Generating the Link Graph . . . . .	227
7.8.2	Graph Construction Algorithms . . . . .	228
7.9	Related Work . . . . .	232
7.10	Future Work and Concluding Remarks . . . . .	234
7.10.1	Queries . . . . .	234
7.10.2	Presentation . . . . .	234
7.10.3	Security . . . . .	235
7.10.4	Closing the Loop . . . . .	235
7.10.5	Concluding Remarks . . . . .	236
<b>8</b>	<b>Trails and Program Comprehension</b>	<b>238</b>
8.1	Introduction . . . . .	239
8.2	OOP, Java and Object Coupling . . . . .	240
8.2.1	Object Oriented Programming . . . . .	240
8.2.2	Java . . . . .	240
8.2.3	Coupling . . . . .	240
8.2.4	Refactoring . . . . .	241
8.2.5	The Jakarta Project . . . . .	242
8.2.6	Java Documentation Systems . . . . .	242
8.3	AutoDoc . . . . .	244
8.4	AutoCode . . . . .	246
8.4.1	Architecture . . . . .	246
8.4.2	Source Code Display . . . . .	248
8.4.3	Examples . . . . .	248
8.5	Power Law Distributions in Class Relationships . . . . .	252
8.5.1	Related Studies . . . . .	252
8.5.2	Results . . . . .	253
8.6	Potential Gain as a Refactoring Metric . . . . .	259
8.7	Concluding Remarks and Future Work . . . . .	262
8.7.1	OO Coupling . . . . .	262

<b>9</b>	<b>Future Work and Concluding Remarks</b>	<b>264</b>
9.1	Summary of the Thesis . . . . .	265
9.2	Personalization . . . . .	267
9.3	Meta-Search . . . . .	268
9.4	The Software Navigation Problem . . . . .	270
9.5	Navigation in Virtual Environments . . . . .	272
9.6	Graph Characteristics . . . . .	273
9.7	Final Remarks . . . . .	274
<b>A</b>	<b>List of Abbreviations</b>	<b>275</b>
<b>B</b>	<b>List of Mathematical Symbols</b>	<b>279</b>
<b>C</b>	<b>Linear Correlation between Ranking Metrics</b>	<b>280</b>
<b>D</b>	<b>Non-Linear Correlation between Ranking Metrics</b>	<b>289</b>

# List of Figures

2.1	Artist's impression of Bush's memex. . . . .	24
2.2	The Xanadu Transcopyright model. . . . .	25
2.3	An expanded version of Conklin's list of hypertext systems. . . . .	28
2.4	The Bow-Tie model of the Web. . . . .	31
2.5	Power Law Relationships in the Web with exponents. Percentages denote confidence levels for intervals. The PageRank citation index is discussed in section 2.6. . . . .	32
2.6	Architecture of a typical search engine. . . . .	34
2.7	Bag of Words and Bag of $N$ -grams . . . . .	37
2.8	Korfhage's matrix for evaluation metrics . . . . .	39
2.9	Software evaluation metrics . . . . .	39
2.10	TREC WebTrack Collections . . . . .	40
2.11	HTML tag weighting schemes proposed by (a) Cutler et al and (b) Kim et al. . . . .	43
2.12	Vivisimo's clustered search results. . . . .	48
2.13	Lycos's sitemap. . . . .	52
2.14	ExPASy's sitemap. . . . .	53
2.15	Apple's sitemap. . . . .	54
2.16	An Unidentified Flying Sitemap! . . . . .	55
2.17	Sitemap showing information from various weather observation areas. . . . .	56
2.18	Sitemap from the Journals of the AMA . . . . .	57
2.19	The Cha-Cha interface. . . . .	58
2.20	The StarTree interface. . . . .	58
2.21	The VisIT interface. . . . .	59
2.22	The Nattoview interface. . . . .	60
2.23	The Kartoo interface. . . . .	61



2.24	Zin and Levene’s generic Web View algorithm. . . . .	65
3.1	An example Web topology. . . . .	72
3.2	The Best Trail algorithm. . . . .	75
3.3	An example navigation tree. . . . .	77
3.4	Table showing trail scores using Weighted Sum and Sum Distinct. . . . .	81
3.5	Results for the query “dotty” on the topology shown in figure 3.1. . . . .	82
3.6	Improvements of Average Trail Scores induced by Filtering . . . . .	84
3.7	Correlation between the increase in trail scores given by the Weighted Sum and the increase shown using Sum Unique. . . . .	85
3.8	Tree of Tips for the navigation tree shown in figure 3.3. The table representing this tree is shown in figure 3.9. . . . .	87
3.9	Table showing candidate tips for expansion. . . . .	88
3.10	Properties of the test corpora. . . . .	90
3.11	Increasing $I_{explore}$ increases score. . . . .	91
3.12	Increasing $I_{converge}$ increases score. . . . .	92
3.13	Increasing $I_{explore} : I_{converge}$ increases score with sum distinct. . . . .	93
3.14	Increasing $I_{explore} : I_{converge}$ decreases score with weighted sum. . . . .	94
3.15	Increasing the starting point count increases the average trail score. . . . .	95
4.1	An algorithm for computing Potential Gain . . . . .	102
4.2	A matrix-based algorithm for computing Potential Gain . . . . .	103
4.3	Potential Gain values converge rapidly in a few iterations. . . . .	104
4.4	Log-Log plots showing power law distributions in the values of Potential Gain for the web sites of (a) the DTI, (b) Birkbeck and (c) UCL and also for the pages of the TREC WT10g corpus. . . . .	105
4.5	Log-Log plots showing (a) that there is no power law in the distribution of Potential Gain values on the Sleepycat web site (b) that there is such a distribution in the values of potential gain for pages within the JDK 1.4 Javadocs. . . . .	106
4.6	Metrics used in tests of Potential Gain as a starting point selection metric. . . . .	108
4.7	Corpora used in tests of Potential Gain as a starting point selection metric. . . . .	108
4.8	Queries used in tests of Potential Gain as a starting point selection metric. . . . .	109

4.9	Percentage increases in trail score achieved by sorting by the given metrics in preference to $Found(p)$ after previous sorted by the number of keywords then by the relevance $\mu(p)$ . The two averages show the mean values taken over all <i>corpora</i> or webcases and over all <i>queries</i> . . . . .	110
4.10	Shows the increase in trail scores achieved by using various measures for starting point selection. . . . .	111
5.1	The navigation engine architecture. . . . .	116
5.2	Behind the facade. Structure of classes in the <code>core</code> package . . . . .	120
5.3	Two subclasses inherit from <code>TrailAlgorithm</code> , which relies heavily on <code>ActiveWebcase</code> . . . . .	121
5.4	Behind the facade. Interaction of core classes. . . . .	123
5.5	Algorithm for reading features and writing temporary files. . . . .	125
5.6	Algorithm to merge and aggregate files of <i>keyword</i> , <i>urlid</i> , <i>score</i> tuples. . . . .	125
5.7	Algorithm to build inverted file in B-tree. . . . .	126
5.8	Algorithm to run post-processing operations to generate webcase data. . . . .	126
5.9	Structure of the post-process classes. . . . .	128
5.10	Dependencies between post-process operations. . . . .	129
5.11	Advanced query syntax for search engines. . . . .	132
5.12	Algorithm for filtering documents. . . . .	133
5.13	Unix commands for extracting lists of files from archives. . . . .	134
5.14	Algorithm to Summarize Web Documents . . . . .	136
5.15	Time taken to load and generate an average summary, given a certain number of threads. . . . .	139
5.16	Time taken to load and generate an average summary, given a certain number of threads, where the number of threads is greater than one. . . . .	140
5.17	Time taken to load and generate an average summary, given a certain number of open file handles. . . . .	140
5.18	Time taken to load and generate an average summary, given a certain number of documents cached. . . . .	141
5.19	The time taken to load a document correlates strongly with its length. . . . .	142
5.20	The time taken to generate a summary for a document does not correlate strongly with its length. . . . .	143
5.21	Distribution in the times taken to load individual documents. . . . .	144
6.1	Results for the query “computers” on UCL’s Web site. . . . .	151

6.2	Results for the query “pitkow” on Xerox PARC’s Web site. . . . .	153
6.3	Results for the query “oxygen” on the Creoscitex Web site. . . . .	154
6.4	Results for the query “Dbt” on the Sleepycat Web site. . . . .	155
6.5	Trails found for the query “Dbt” on the Sleepycat Web site. . . . .	157
6.6	Results for the query “Cryptography” on UCL’s Web site. . . . .	158
6.7	Trails found for the query “hotel management” on the BBK site. . . . .	160
6.8	Trails found for the query “accomodation” on the SCSIS site. . . . .	162
6.9	Trails found for the query “accommodation” on the SCSIS site. . . . .	163
6.10	Trails found for the query “andrew” on the SCSIS site. . . . .	164
6.11	Trails found for the query “application form” on the SCSIS site. . . . .	165
6.12	Trails found for the query “birkbol programmes” on the SCSIS site. . . . .	166
6.13	Trails found for the query “c++ notes” on the SCSIS site. . . . .	167
6.14	Trails found for the query “exam papers” on the SCSIS site. . . . .	168
6.15	Trails found for the query “mark” on the SCSIS site. . . . .	169
6.16	Trails found for the query “neural network” on the SCSIS site. . . . .	170
6.17	Trails found for the query “xml” on the SCSIS site. . . . .	171
6.18	Queries used as the basis for authoring trails. . . . .	175
6.19	Statistics concerning the trails authored for the queries shown in figure 6.18.	176
6.20	Trails found for the query “access course” on the Birkbeck site. . . . .	177
6.21	Trails found for the query “birbbeck logo design” on the Birkbeck site. . . .	179
6.22	Trails found for the query “design postgraduate” on the Birkbeck site. . . .	180
6.23	Trails found for the query “exam papers” on the Birkbeck site. . . . .	181
6.24	Trails found for the query “international students” on the Birkbeck site. . . .	182
6.25	Trails found for the query “writing up phd” on the Birkbeck site. . . . .	183
6.26	Trails found for the query “a student gym” on the Birkbeck site. . . . .	185
6.27	Trails found for the query “beginners painting” on the Birkbeck site. . . . .	186
6.28	Trails found for the query “bsc programming” on the Birkbeck site. . . . .	188
6.29	Trails found for the query “dept of philosphy” on the Birkbeck site. . . . .	189
6.30	Trails found for the query “dept of philosophy” on the Birkbeck site. . . . .	190
6.31	Trails found for the query “distance learning” on the Birkbeck site. . . . .	192
6.32	Trails found for the query “mba” on the Birkbeck site. . . . .	193
6.33	Trails found for the query “access 97” on the Birkbeck site. . . . .	195

6.34	Trails found for the query “project management” on the Birkbeck site. . . .	196
6.35	Trails found for the query “part time” on the Birkbeck site. . . . .	197
6.36	Trails found for the query “research grants history” on the Birkbeck site. . .	198
6.37	Trails found for the query “social” on the Birkbeck site. . . . .	199
6.38	Trails found for the query “ba history” on the Birkbeck site. . . . .	201
6.39	Distribution of Trail Lengths. . . . .	203
6.40	Example graph for partitioning. . . . .	206
6.41	Example of the effect of various partitioning schemes on the graph shown in figure 6.40. . . . .	206
6.42	Algorithm to merge sets of trails from multiple sources. . . . .	208
7.1	Interaction between Microsoft CMS and SPS. . . . .	213
7.2	Architecture of DbSurfer. . . . .	218
7.3	Example XML entry extracted from the DBLP Schema. . . . .	219
7.4	UML diagram showing the DBLP Schema. . . . .	221
7.5	Example results using DbSurfer for the query “sergey anatomy”. . . . .	222
7.6	Example results using DbSurfer for the query “vannevar bush”. . . . .	223
7.7	Comparison of reciprocal rank and total time taken for 20 citation-seeking queries on DbSurfer, BANKS and CiteSeer. . . . .	225
7.8	Example of a taxonomy . . . . .	226
7.9	Example of documents mapped to the taxonomy shown in Figure 7.8 . . . . .	227
7.10	Conditions under which links are added to the Vertical Trailer graph. . . . .	227
7.11	Algorithm to create a directed graph from a document classification hierarchy. . . . .	228
7.12	Directed Acyclic Graph (DAG) formed from the union of the taxonomy and classification map. . . . .	229
7.13	Algorithm to add inter-document links to a classification hierarchy graph. . . . .	229
7.14	Algorithm to add parent-child links to a classification hierarchy graph. . . . .	230
7.15	Graph used for trail discovery. . . . .	231
7.16	Revised architecture of DbSurfer. . . . .	237
8.1	AutoDoc results for the query “Sql”. . . . .	245
8.2	Architecture of AutoCode. . . . .	246
8.3	Illustration of coupling types and their graph representations. . . . .	247
8.4	Results for the query “zip” on the JDK 1.4 source code. . . . .	249

8.5	Trails returned for the query “zip” on the JDK 1.4 source code. . . . .	250
8.6	Trails returned for the query “writer” on the JDK 1.4 source code. . . . .	251
8.7	Log-log plots showing power law distributions in the number of (a) fields, (b) methods and (c) constructors of classes in the JDK class libraries. . . . .	254
8.8	Log-log plots showing the relationships between (a) the number of fields and the number of constructors, (b) the number of methods and the number of constructors and (c) the number of methods and the number of fields for classes in the JDK. . . . .	255
8.9	Correlation matrix for class members in the JDK . . . . .	255
8.10	Log-Log plots showing power law distributions in (a) the number of subclasses of each class and (b) the number of interfaces implemented by classes, both based on data from the JDK class library. . . . .	256
8.11	Log-Log plots showing power law distributions in (a) the number of classes referenced as field variables and (b) in the number of classes which contain references to classes as field variables. . . . .	257
8.12	95% confidence intervals for power law exponents in JDK. . . . .	258
8.13	95% confidence intervals for power law exponents in Tomcat. . . . .	258
8.14	95% confidence intervals for power law exponents in Ant. . . . .	258
8.15	The fifteen classes with the highest reverse aggregation Potential Gain values: JDK . . . . .	260
8.16	The fifteen classes with the highest inheritance Potential Gain values: JDK . . . . .	260
9.1	Contributions of the thesis. . . . .	265
9.2	Mock-up of how the trail finding interface could be applied to the graph of potential user interactions at the operating system level. . . . .	271
C.1	Correlation between Web metrics on the Sleepycat webcase using Pearson’s product moment correlation coefficient . . . . .	281
C.2	Correlation between Web metrics on the SCSIS webcase using Pearson’s product moment correlation coefficient . . . . .	282
C.3	Correlation between Web metrics on the UCL webcase using Pearson’s product moment correlation coefficient . . . . .	283
C.4	Correlation between Web metrics on the UCL-CS webcase using Pearson’s product moment correlation coefficient . . . . .	284
C.5	Correlation between Web metrics on the Intel webcase using Pearson’s product moment correlation coefficient . . . . .	285
C.6	Correlation between Web metrics on the Birkbeck webcase using Pearson’s product moment correlation coefficient . . . . .	286

C.7	Correlation between Web metrics on the DTI webcase using Pearson's product moment correlation coefficient . . . . .	287
C.8	Correlation between Web metrics on the JDK 1.4 webcase using Pearson's product moment correlation coefficient . . . . .	288
D.1	Correlation between Web metrics on the Sleepycat webcase using Kendall's Tau Statistics . . . . .	290
D.2	Correlation between Web metrics on the Sleepycat webcase using Spearman's Rho . . . . .	290
D.3	Correlation between Web metrics on the SCSIS webcase using Kendall's Tau Statistics . . . . .	291
D.4	Correlation between Web metrics on the SCSIS webcase using Spearman's Rho . . . . .	291
D.5	Correlation between Web metrics on the JDK 1.4 webcase using Kendall's Tau Statistics . . . . .	292
D.6	Correlation between Web metrics on the JDK 1.4 webcase using Spearman's Rho . . . . .	292
D.7	Correlation between Web metrics on the UCL webcase using Spearman's Rho . . . . .	293
D.8	Correlation between Web metrics on the UCL-CS webcase using Spearman's Rho . . . . .	293
D.9	Correlation between Web metrics on the Intel webcase using Spearman's Rho . . . . .	294
D.10	Correlation between Web metrics on the Birkbeck webcase using Spearman's Rho . . . . .	294
D.11	Correlation between Web metrics on the DTI webcase using Spearman's Rho . . . . .	295

## Part I

# The Navigation Problem and Related Issues

# Chapter 1

## Introduction

A beginning is the time for taking the most delicate care that the balances are correct.

Herbert 1965

I will tell you the beginning, and, if it please your ladyships, you may see the end; for the best is yet to do; and here, where you are, they are coming to perform it.

Shakespeare 1599



## 1.1 Abstract

Pages returned by Web search engines are often used as starting points for further navigation. The hyperlinks which users follow from the start page form a trail. Despite this, navigation possibilities are not considered by conventional search engines. Nor do search engines provide any support in suggesting trails for users to follow.

Users experience the “Navigation Problem”, where they are said to be “Lost in Hyperspace”, whenever they are navigating the Web (or some other hypertext) and are either unsure of where they are relative to another page, unsure of which link to follow in order to find what they’re looking for, unsure of where they’ve been or unsure of where they will get to when they follow any given link.

This thesis describes a potential solution to the navigation problem – a “navigation engine”, developed to provide memex-like information trails in response to a user’s query. This unifies ideas from the information retrieval and hypertext communities. The main body of the thesis is presented in two parts, covering two major contributions – the implementation of this system and its application to the fields of web navigation, database search and program comprehension.

An existing algorithm, the Best Trail, has been refined and enhanced. The described implementation – the first effective, publically-accessible implementation of this approach to trail-finding – has been made possible by changes to the selection functions, the addition of new methods for removing redundant information and the introduction of a new link-based metric.

The Potential Gain metric improves the selection of starting points from which the Best Trail algorithm will find the trails by evaluating the potential of a page to provide future navigation opportunities. It is defined, for a given page, in terms of the fraction of trails of various lengths which start from that page. Algorithms for computing Potential Gain are shown along with techniques for using it to improve node selection. Experiments have been performed which show the effectiveness of the metric in increasing the likelihood of finding high scoring trails.

A comprehensive description of the architecture of the navigation system covers not only the basic components and the algorithm for index creation but also methods for handling extended query syntax, indexing content from non-standard file types and summarizing Web documents. It is shown how this system provides useful trails and enhances user navigation experiences on Web sites. The use of trails alleviates the navigation problem in two ways. Firstly, by semi-automating the navigation process, the user is able to follow a pre-determined path which can be assumed to be relevant. Secondly, by providing contextual information, the trails allow users to make more informed navigation decisions and hence avoid getting “lost”.

Building on this work, a tool called DbSurfer has been developed which provides an interface to relational databases. Data is extracted in the form of an inverted index and a graph of foreign key dependencies. Together, these can be used to construct trails of information, solving the join discovery problem and allowing free text search on the contents. The free text search and database navigation facilities can be used directly, or can be used as the foundation for a customised interface.

The navigation problem also exists in automatically-generated program documentation and in the source code from which such documentation is typically generated. The final contribution in this thesis is a pair of tools, AutoDoc and AutoCode, for indexing Javadocs and Java source code, respectively. AutoCode shows trails according to graphs of coupling relationships – graphs which are shown to be Web-like in their scale-free topology.

The development of the navigation engine called for solutions to several interesting problems and leaves a potential solution for many more. This represents an important step on the path to Bush's *Web of Trails*.

## 1.2 Motivation

In Vannevar Bush's seminal paper "As We May Think" (Bush 1945) he suggested a future machine called a *memex*, which would help the user build a "web of trails". In doing so, he introduced the world to the concept of linked documents, which would later be known as *hypertext*, and of the *trail* – a sequence of linked pages. The work was continued by Nelson, Engelbart and Tim Berners-Lee who introduced the World Wide Web (Berners-Lee 1999) – the largest and the most successful hypertext system ever developed. The ubiquitous Web has dominated hypertext research in recent years and is likely to continue to do so, but two key usability problems remain unsolved.

The first major problem to be addressed is the *resource discovery problem*, that of finding a given document on the web, finding a resource which answers a given question, or which provides information about a given subject. Solutions to this problem have included directories such as Yahoo! or the Open Directory Project (ODP) and automated web search engines such as AltaVista and Google.

The second major problem on the web is the *navigation problem* of avoiding situations where people get "lost in hyperspace?" (Levene and Loizou 1999). A person is said to be "lost in hyperspace" if they are navigating (or browsing) the web (or some other hypertext) and are either unsure of where they are relative to another page, unsure of which link to follow in order to find what they're looking for, unsure of where they've been or unsure of where they will get to when they follow a given link. Search engines help to solve the problem by providing a means for people to find what they're looking for, but neither provide support for users once they have their initial results nor show those results in the context of the pages around them. Search engines also contribute to the problem by directing users to the middle points of a web site with no reference to important points, such as home pages, etc.

The main contribution of this thesis is to describe a potential solution to the navigation problem. In order to tackle the navigation problem, a *navigation engine* has been developed which builds information trails in response to a user's query. This unifies ideas from the information retrieval and hypertext communities. It is shown how this system enhances user navigation experiences on web sites and how the system has wider applicability in the fields of databases and program comprehension.

### 1.3 Outline of the Thesis

This thesis describes the implementation of the system and its application in relation to web navigation, databases and program comprehension. It is organized as follows:

**Chapter 2** describes the resource discovery and navigation problems, the relationship between them and proposed solutions. This includes related work in information retrieval systems, search engines, link analysis and Web navigation aids.

**Chapter 3** describes the *Best Trail Algorithm* and its implementation. This is the key algorithm used to compute the trails. The algorithm was originally described in Levene and Zin 2001 but this chapter describes its first *effective* implementation. This chapter expands on work presented in Wheeldon and Levene 2003.

**Chapter 4** describes a new metric called *Potential Gain* which evaluates the potential of a page to provide future navigation opportunities. The potential gain of a node is formally defined for as the sum for all  $l > 0$  of the product of the fraction of trails of length  $l$  which start at that node and a discounting factor. This chapter provides an analysis of the metrics utility and also covers work introduced in Wheeldon and Levene 2003.

**Chapter 5** describes the architecture of the navigation system. A basic description of the architecture was presented in Levene and Wheeldon 2001, and this extends it to show not only the basic components and the algorithm for index creation, but also methods for handling extended query syntax, indexing content from non-standard file types and summarizing web documents.

**Chapter 6** discusses the application of this work to solve the navigation problem in the World Wide Web. The user interfaces presented here have been previously covered in Levene and Wheeldon 2001 and Wheeldon, Levene, and Zin 2002. An overview of the work is provided in Levene and Wheeldon 2003.

**Chapter 7** expands on work described in Wheeldon, Levene, and Keenoy 2003 to show how the trail finding approach can be used to provide effective free-text search with navigation facilities in relational databases. This is extended with new work concerning data stored in document, information and content management systems with hierarchical document classification schemes.

**Chapter 8** expands on work presented in Wheeldon, Levene, and Zin 2002 to show how the system can be used to index program documentation. It is also possible to index program source code and show trails which aid its comprehension as described in Wheeldon, Counsell, and Keenoy 2003. This is improved by computing trails on multiple graphs and combining the results. Analysis on these graphs reveals a scale-free topology as described in Wheeldon and Counsell 2003b. Use of the Potential Gain metric on these graphs yields interesting results concerning refactoring as described in Wheeldon and Counsell 2003a.

**Chapter 9** concludes the thesis with ideas for future research.

## 1.4 Acknowledgements

The *navigation engine* described in chapter 5 was constructed over a period of about 3 years, and served as the basis for NavigationZone's trail-finding technology. The design and construction of the system would not have been possible without the help of those involved with NavigationZone, in particular Nadav Zin and Sean Greenan for their ideas concerning algorithms, hierarchies and java documentation, James Skene for the design and implementation of the second web robot and Jon Bitmead for the implementation of the user interface.

The author would like to thank Mazlita Mat-Hassan for the user study described in chapter 6. Her work has helped greatly in validating the approach taken in this thesis. The work of Nadav Zin and Mark Levene in the initial development of the Best Trail Algorithm is also greatly appreciated.

Some of the illustrations and figures in this thesis have been copied from other sources. In particular, the illustration of Memex (figure 2.1) is accredited to Ian Adelman and Paul Kahn of Dynamic diagrams and is taken from the Association of Computing Machinery (ACM) reprint of Bush 1945; the transcopyright diagram (figure 2.2) is taken from the xanadu.com web site; the illustration of Nattoview (figure 2.22) is taken from Shiozawa, Nishiyama, and Matsushita 2001; and the picture showing Microsoft Content Management server and SharePoint Server (figure 7.1) is taken from Microsoft's web site.

Similarly the navigation system has relied on the development work of others. In particular, thanks go to Jason Shattu, for the development of Java2HTML; Stephen North and all those involved in the GraphViz project and all members of the Jakarta project and Sun's Java development team.

Finally, thanks go to all those who have helped in the ideas, concepts in this thesis and the endless proof-reading. In particular Steve Counsell, Bryn Reeves, John Wheeldon, Connie Bottel, Kevin Keenoy and most of all my supervisor, Mark Levene, without whom none of this would have been possible.

## Chapter 2

# Problems and Solutions

In Xanadu did Kubla Khan,  
A stately pleasure-dome decree :  
Where Alph, the sacred river, ran  
Through caverns measureless to man  
Down to a sunless sea.

So twice five miles of fertile ground  
With walls and towers were girdled round:  
And there were gardens bright with sinuous rills,  
Where blossomed many an incense-bearing tree;  
And here were forests ancient as the hills,  
Enfolding sunny spots of greenery.

Coleridge 1816

## 2.1 Introduction

This chapter introduces and describes the *navigation problem* in hypertext. The relationship between the navigation and *resource discovery* problems is explored. Measures are described for solving each of these problems.

The rest of this chapter is organized as follows:

**Section 2.2** presents a brief history of the early developments in hypertext, starting with Bush's memex, through Engelbart's oN-Line System (NLS) and Nelson's Xanadu to the hypertext boom in the late 80s and early 90s.

**Section 2.3** describes the *World Wide Web* – briefly discussing its creation before focussing on the properties and structure of the Web.

**Section 2.4** describes the resource discovery problem and the proposed solutions including search engines and meta-search engines.

**Section 2.5** describes classic *information retrieval* techniques for identifying sets of *relevant* documents. The techniques used mainly involve *keyword* based metrics.

**Section 2.6** shows how these techniques have been extended using metrics designed specifically for the Web.

**Section 2.7** describes search techniques which go beyond returning ordered sets of documents, and return direct answers or complex structures.

**Section 2.8** describes the navigation problem and the differences and similarities between the resource discovery and navigation problems.

**Section 2.9** describes navigation aids designed to tackle this problem, including link suggestion tools and site maps.

**Section 2.10** explores systems for manually defining trails or *guided tours* through Web sites as a method of solving the navigation problem. Such systems were commonly used in early hypertext systems and were often found to be highly effective.

**Section 2.11** describes how the process of automating the construction or discovery of these trails has progressed to date.

**Section 2.12** summarizes the main points of the chapter and sets to scene for the rest of the thesis.

## 2.2 A Brief History of Hypertext

The history of hypertext begins with Vannevar Bush's seminal 1945 paper "As We May Think" (Bush 1945) in which he suggested a future machine called a memex, which would help the user build a "web of trails". Bush had been Franklin D. Roosevelt's scientific advisor during the war, working on the Manhattan Project and leading-edge computing technology in the work to crack the Japanese cipher code-named *Ultra* (Zachary 1999; Burke 1994). This placed him in a unique position to offer a vision for the future. He had previously designed a machine called the *Rapid Selector* which would store documents and retrieve them at will, and he extended this in his design for the memex. The machine would be a cabinet-like box into which the user could store documents and images using a rapid-action camera (figure 2.1). A sequence of such documents could then be annotated, and linked together to form a trail. A large *web of trails* could be constructed via a memex. Levers would allow the user to quickly move backwards and forwards through these trails and recall them when needed. It is noteworthy that in this case, it is the reader who creates the links, whereas modern hypertext systems tend to leave this responsibility to the document's authors. Although no memex was ever built, the importance of the vision lived on. Bush's contribution to science has been analysed by many people (Nyce and Kahn 1991) and even the name "memex" lives on in modern hypertext research (Chakrabarti, Srivastava, Subramanyam, and Tiwari 2000). The linking of documents was the first suggestion of hypertext, although the name had not yet been invented.

The term "hypertext" was coined by Ted Nelson, famous for his visions of the *Xanadu* project (Nelson 1993). He described hypertext as "a body of written or pictorial material interconnected in such a complex way that it could not conveniently be presented or represented on paper" (Nelson 1965). In the Xanadu model, content portions are published as separate entities (with certain permissions) in a content *pool*. Documents are then constructed as virtual documents with references to this content. The underlying content never changes, so links never break, but documents can be re-written by altering the references and adding new content to the pool. Software developers might see similarities between this and the *flyweight* (Gamma, Helm, Johnson, and Vlissides 1995) pattern for shared objects. In the Xanadu model the shared objects would be the content text. Xanadu offered the possibility of two-way, unbreaking links which could (like those in the memex) be created by any user between any document. *Transclusive* links (Nelson 1999) would allow user's to connect any excerpt to its original, implicitly providing deep version management whilst the *transcopy-right* model (see figure 2.2) would allow authors control over and royalty payments for these excerpts. Users could also be offered a side-by-side intercomparison of connected documents. The front-end would show the differences between versions in a style of presentation similar to that of a graphical *diff* tool (Myers 1990).

Unlike memex, which was intended as a vision for the future, Xanadu was a project whose implementation was attempted more than once, with two versions now available under an open-source license as Udanax Gold and Udanax Green<sup>1</sup>. Despite this, many of the goals of the Xanadu project were never implemented in any hypertext system and it is worth a brief investigation into the possible reasons for Xanadu's lack of adoption.

Some authors have been somewhat uncharitable in their criticism of Xanadu's failings. One

---

<sup>1</sup> <http://www.udanax.com/>



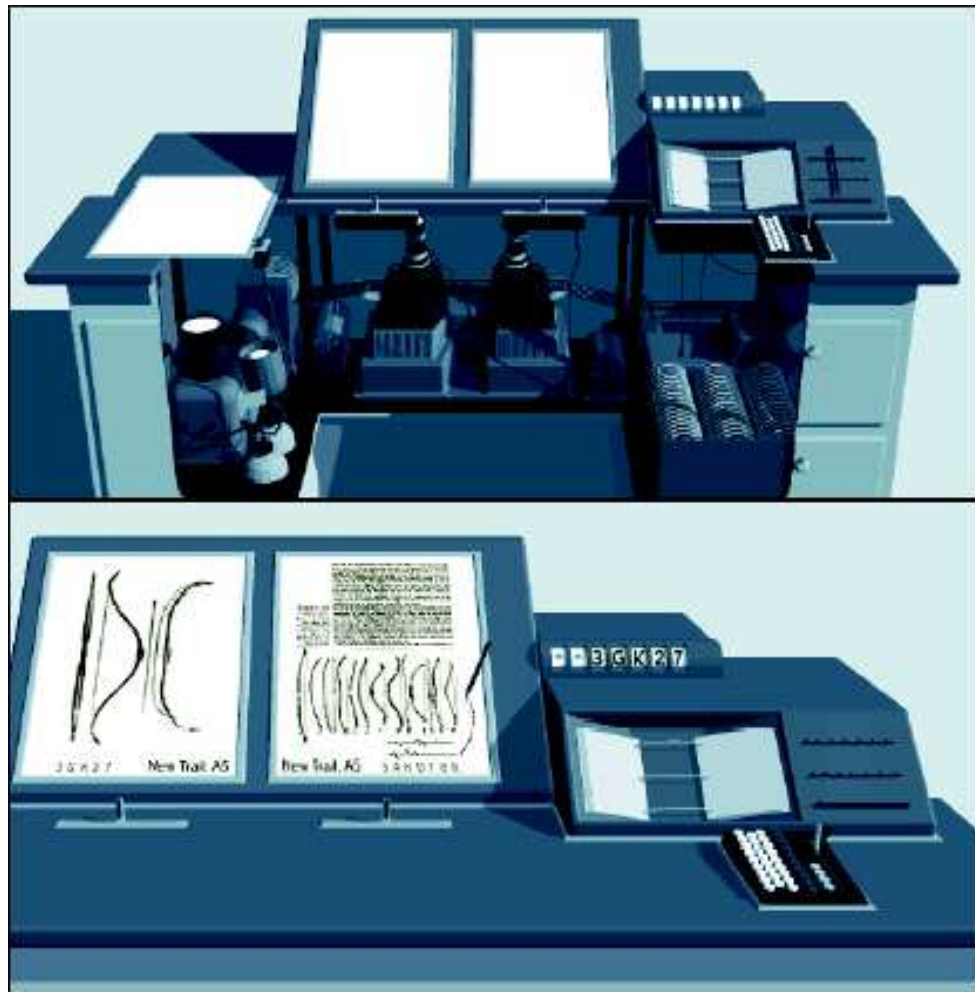


Figure 2.1: Artist's impression of Bush's memex.

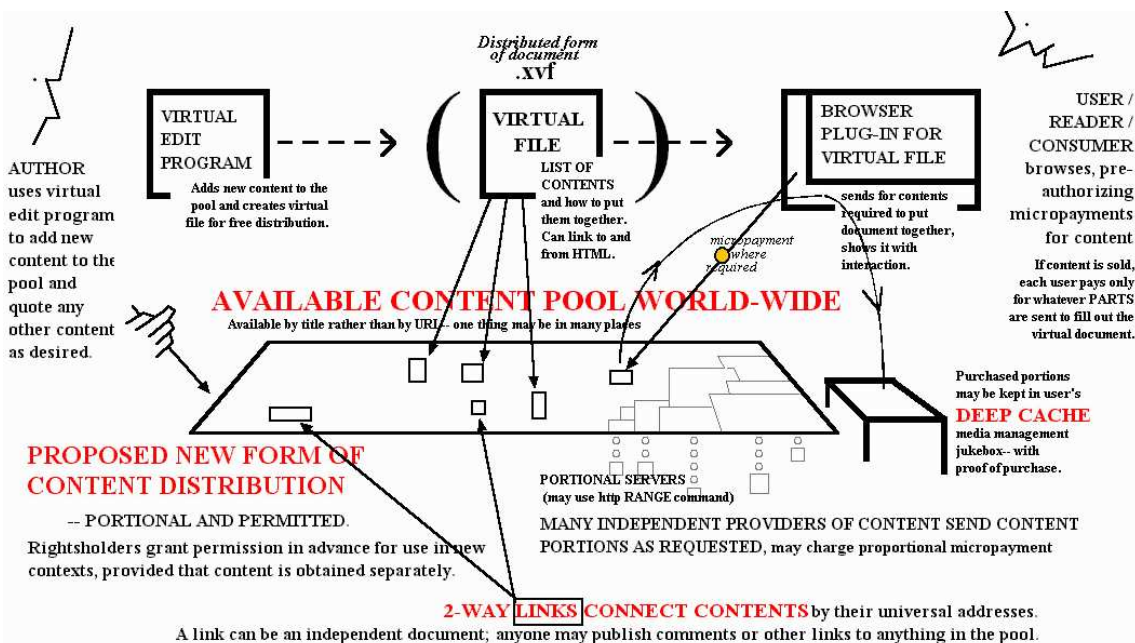


Figure 2.2: The Xanadu Transcopyright model.

example, the Wired article, “Curse of Xanadu” provoked heavy criticism from Nelson himself for its claims that “Xanadu couldn’t be done” and its misstatement and misrepresentation of transclusion and enfiladics (Wolf 1995; Nelson 1995). If Wolf’s suggestions of technical impossibility and ignorance are too difficult to believe, perhaps a simpler reason can be found in Eric Drexler’s account of hypertext publishing (Drexler 1987). Drexler defines the essential characteristics of a *hypertext publishing system* as being a hypertext system which has support for bidirectional, fine-grained, filtered linking. However Drexler continues that to state that “The most important [characteristic] is that the system be public; the difference between using a small, private system and using a large, public system will be like the difference between using a typewriter and filing cabinet and using a publisher and a major library”. Xanadu’s failure to address the needs of users to control the software in their own environments and yet maintain the power to link to documents on other servers may have been its undoing; as may the failure to set open standards so that developers could extend and enhance the original ideas.

Douglas Engelbart was another early believer in Bush’s ideas. He dreamt of a machine that would aid human cognition, and felt that Bush’s trails represented a “beautiful example of a new capability in symbol structuring” (Engelbart 1962). He told the Advanced Research Projects Agency (ARPA) that it was “quite feasible to develop a unit record system . . . with automatic trail establishment and trail following facility” and set out to prove it. In 1963, ARPA gave Engelbart the funds to start the Augmentation Research Centre (ARC), at the Stanford Research Institute (SRI). This was a research laboratory dedicated to finding better ways to use technology for research and communication and which, at its peak, grew to 47 people.

Engelbart, along with colleagues at the SRI such as William English and John Rulifson, de-

veloped the NLS. A revolutionary system for its day, it introduced the concepts of hypertext linking, tele-conferencing, word processing, e-mail, the mouse and multiple windowing systems. In the Spring of 1967, it was announced that the thirteen ARPA-sponsored computer research labs would be networked. The ARC became the second host on the ARPANet. Engelbart saw the NLS as the ideal tool for this environment. At the 1968 Fall Joint Computer Conference in San Francisco, Engelbart, operating the computer from the stage via a home-made modem, used NLS to demonstrate the system and illustrate his ideas to the audience<sup>2</sup>.

The hypertext nodes in NLS were organised into files. Reference links would help a user to move from one file to another and paths could be specified using linked lists of links. The NLS files were structured into a hierarchy of segments called statements. Each statement was limited to 3000 characters in length and tagged according to its level in the hierarchy. Links referencing these tags could be established between any of these statements regardless of which file they were in.

Other notable hypertext systems include the Hypertext Editing System (HES), Document Examiner, Intermedia, Notecards ZOG, and the Knowledge Management System (KMS):

**HES** was developed by Andries van Dam at Brown University. It supported branching text arranged in menus and was designed to run on an IBM/360 mainframe.

**Symbolics Document Examiner** was developed around 1985 and used for exploring the set of Symbolics Lisp manuals (Walker 1987).

**Intermedia** was developed by Norman Meyrowitz at Brown University between 1985 and 1991 (Meyrowitz 1986). It was an educational hypertext system, implemented for Apple's version of UNIX. It was a window-based system which supported bi-directional links between arbitrary strings of text and included a text editor, graphics editor, image viewer and timeline editor.

**NoteCards** was designed at Xerox's Palo Alto Research Center (PARC) and was designed to help authors, researchers and designers work with ideas (Halasz 1988). Each card represented a node between which typed links could be made. A "browser card" provided a structural overview diagram of the notecards and links.

**ZOG** was a system developed at Carnegie Mellon University (CMU) which supported multiple users working on a large time-sharing system.

**KMS** was a direct descendant of ZOG and was developed as a commercial product in the early 1980s (Acksyn, McCracken, and Yoder 1988). Each node in KMS is a frame, of which only two could appear on screen at one time. Two types of links were supported: *tree* and *annotation*. Tree links referenced material at lower levels in a hierarchy whilst annotations referenced peripheral material, such as comments. An index could be formed by listing all levels in the hierarchy. All these links were followed by clicking with the mouse on a segment of anchor text, an idiom followed by many hypertext systems.

---

<sup>2</sup> See <http://sloan.stanford.edu/mousesite/1968Demo.html> for archive video footage of the demonstration.

Conklin 1987 discusses the features of many hypertext systems. Table 2.3 extends Conklin's list of hypertext systems and provides a comparison between some of the systems. The table includes the World Wide Web (discussed in the following section) and other hypertext systems such as HyperPad, LinkWay, Folio Views and Black Magic – developed in the late 1980s and early 1990s to run on International Business Machines (IBM) compatible Personal Computers (PCs) (Fairhead 1990). The following features of hypertext systems are listed:

**Hierarchy** Is there specific support for a hierarchical structure?

**Graph-based** Does the system support nonhierarchical (cross-reference) links?

**Link types** Can links have types?

**Attributes** Can user-designated attribute/value pairs be associated with nodes or links?

**Trails** Can many links be strung together into a single persistent object? Paths in Intermedia can be user programmed. AmigaGuide supports a single guided tour controlled by the behaviour of the “Browse” button. The support for paths, trails and tours will be discussed in sections 2.10 and 2.11.

**Versions** Can nodes or links have more than a single version?

**Code** Can arbitrary executable procedures be attached to events at nodes or links? Client-side programs can be added to web pages using JavaScript, ActiveX, Flash, Java but support may be limited to certain browsers or operating systems. Server-side programs can be written using any language.

**String search** Can the hyperdocument be searched for strings (including keywords) ? Search engines such as Google, AltaVista, etc. provide partial search facilities for the Web. There is no centralized service covering all possible information.

**Text editor** Which editor is used to create and modify the contents of nodes?

**MultiUsers** Can several users edit the hyperdocument at the same time?

**GFX** Is some form of pictorial or graphical information supported in addition to text?

**Graphical Browser** Is there a browser which graphically presents the nodes and links in the hyperdocument?

It is impossible to cover all the hypertext systems ever developed but it is possible to give examples and show the growing influence that hypertext was having on modern systems. By the early 1990s, no respectable computer system was complete unless it shipped with its own hypertext documentation. Microsoft had the Windows Help system, Commodore had AmigaGuide (Junod 1992) and Silicon Graphics had Insight (Muchowski and Smith 1994), but a revolution was coming . . .

System	Hierarchy	Graph based	Link types	Attr.	Trails	Versions	Code	Search	Text Editor	Multi-user	GFX	Graphical browser
AmigaGuide	No	Yes	No	No	One	No	Yes	No	Any	No	Yes	No
Black Magic	No	Yes	No	No	No	No	Yes	No	Custom	No	Yes	No
Boxer	Yes	Yes	Fixed	No	No	No	Yes	Yes	Emacs	No	Yes	Yes
CREF	Yes	Yes	Yes	No	No	By link	No	Yes	Zmacs	No	Yes	No
Emacs INFO	Yes	No	No	No	No	No	Yes	No	Emacs	No	No	No
Folio Views	No	Yes	Fixed	No	No	No	No	Full-Text	Custom	No	No	No
Guide	No	Yes	Fixed	No	Yes	No	Yes	Yes	Custom	No	Yes	No
HyperCard	No	Yes	No	No	Yes	No	Yes	Yes	Any	No	Yes	No
HyperPad	No	Yes	No	No	No	No	Yes	Yes	Custom	No	No	No
IBIS	Yes	Yes	Yes	No	No	By link	No	No	Basic	Yes	No	No
Insight	Yes	Yes	No	No	No	No	No	Yes	None	No	Yes	No
Itermidia	Yes	Yes	Yes	Yes	No	No	No	Yes	Custom	Yes	Yes	Yes
KMS	Multiple	Yes	Fixed	No	No	Yes	Yes	Yes	WYSIWYG	Yes	Yes	No
Linkway	No	Yes	Fixed	No	No	No	Yes	Yes	Custom	No	Yes	No
Microsoft Help	Yes	Yes	No	No	No	No	Yes	Yes	None	No	Yes	No
neptune	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Smalltalk	Yes	Yes	Yes
NLS/Augment	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Custom	Yes	Yes	No
Notecards	Multiple	Yes	Yes	Nodes	No	No	Yes	Yes	Interlisp	Yes	Yes	Yes
Outline Processors	Yes	No	No	No	No	No	No	Yes	Various	No	No	No
PlaneText	Unix FS.	Yes	No	No	No	No	No	Unix/grep	SunView	Yes	Yes	Yes
Symbolics D.E.	Yes	Yes	No	No	Yes	No	No	Yes	None	No	No	No
SYNVIEW	Yes	No	No	No	No	No	No	No	Unix	No	No	No
textnet	Multiple	Yes	Yes	Yes	Yes	No	No	Keyword	Any	No	No	No
hyperties	No	Yes	No	No	No	No	No	No	Basic	No	Yes	No
WE	Yes	Yes	No	fixed	No	No	No	No	Smalltalk	No	Yes	Yes
World Wide Web	No	Yes	No	No	No	No	Yes	Partial	Any	Yes	Yes	No
Xanadu	No	Yes	Yes	Yes	Yes	Yes	No	No	Any	No	Yes	No
ZOG	Yes	No	No	No	No	No	Yes	Full Text	Spec Pur	Yes	No	No

Figure 2.3: An expanded version of Conklin's list of hypertext systems.

## 2.3 The World Wide Web

Whilst working for Consel Européen pour la Recherche Nucleaire (CERN) in 1980, Tim Berners-Lee had written a small notebook program, “Enquire Within Upon Everything” (shortened to *Enquire*), which allowed links to be made between arbitrary nodes. In March 1989 a proposal circulated (Cailliau 1995) for a new hypertext information management system to organize the complex sets of information provided by various groups of physicists. The article was recirculated the following May. Finally, in September 1990, Mike Sendall, Berners-Lee’s supervisor at the time, gave Berners-Lee a NeXT Cube (Thompson and Baran 1988) and permission to start his work.

Berners-Lee’s proposal was for a hypertext management system in which the information storage software could be both logically and physically separated from the information display software, with a well defined interface between them (Berners-Lee 1989). This would meet CERN’s requirements of remote data access across networks, heterogeneity and operating system independence, access to existing data and decentralized control so that anyone could create new content. Specifically not included in the requirements were mechanisms for handling copyright enforcement and data security as these were “of secondary importance at CERN, where information exchange is still more important than secrecy”. This removed the complexity of Nelson’s transcopyright model. The proposal for private annotated links seems to have been neglected.

A *browser-editor* called *WorldWideWeb* was developed (later it was renamed to *Nexus*) which ran under NeXTStep. It allowed the user to browse hypertext pages specifically designed for it, as well as Usenet groups, File Transfer Protocol (FTP) sites and the local file system, but only local files could be edited. Berners-Lee was fortunate to be working in an environment where the work could be freely published and reported to everyone. Thus, the protocols for the Web were made public, the designs were made public and the source-code for both the server and client was made public. This allowed for adoption by the masses and its meteoric success has been well-publicized (Berners-Lee 1999). The World Wide Web was born!

The foundations for the success of the Web were laid in three of Berners-Lee’s inventions. Firstly, that of the HyperText Markup Language (HTML), used to provide meaning and styling to the majority of web pages. Secondly, that of the HyperText Transfer Protocol (HTTP) (Berners-Lee 1996) and thirdly that of the Uniform Resource Locator (URL) (Berners-Lee 1994). Open standards were published for all of these protocols. The maintenance of these standards is now the responsibility of the World Wide Web Consortium (W3C), of which Tim Berners-Lee is overall director.

A language not without its flaws (Greenspun 1994), HTML may well be replaced with the eXtensible Markup Language (XML) (Harold and Means 2001) which has already found uses in many situations where HTML would be totally unsuitable. XML is a meta-markup language for text documents which is extensible in the sense that new markup tags can be defined within a document. Both HTML and XML documents use embedded tags which can be nested to many levels. The style of both formats is based upon that of the Standard Generalized Markup Language (SGML).

HTTP is a stateless protocol used to transmit Web pages. It is build on top of the Transfer Control Protocol/Internet Protocol (TCP/IP) – the fundamental protocols of all internet

communication. By choosing an open standard for internet communication, Berners-Lee created the first open, distributed hypertext system. Parallels may be drawn between the effects of moving to the Web from close hypertext systems such as Xanadu and Notecards and the effects of open source in the development of Unix (Raymond 1998; Raymond 2001). The use of open source and protocols allowed the creation of new browsers (clients) and servers. Most notable is the graphical browser, Mosaic. Mosaic was developed by the National Center for Supercomputing Applications (NCSA) located at the University of Illinois at Urbana-Champaign (UIUC) and was released in 1993. It subsequently formed the basis of both Netscape and Internet Explorer (IE).

The invention of the URL enabled the location of almost any internet content to be described by a single string. This allowed Web pages to start immediately linking to existing FTP and Usenet resources. URLs all have two sections. The first denotes the protocol and the second is protocol specific. For Web pages, the protocol-specific section specifies a server name and a resource on that server. Typically, this resource description denotes a path to a file, or an instruction to a server based program.

Evaluating the Web with respect to Conklin's survey paper (Conklin 1987) is interesting. Of note are statements such as the assertion that an "essential characteristic of hypertext is the speed with which the system responds to referencing requests. Only the briefest delay should occur (one or two seconds at most)". Also, Conklin's arguments that "window systems have no single underlying database, and therefore lack the database aspect of hypertext" and that DataBase Management Systems (DBMSs) "lack the single coherent interface to the database which is the hallmark of hypertext" seem weak when given the example of the Web, where data is pooled from multiple sources and where pages may take several seconds to return.

Of the features of previous hypertext systems which the Web is missing, Yahoo! and the ODP go some way to providing global hierarchical support, the work by the W3C on XLink may go some way to providing support for link types and arbitrary attributes, and there have been many attempts at producing graph-based visualization tools for subsets of the Web. There has been some work on navigation paths which will be discussed in sections 2.10 and 2.11.

### 2.3.1 Structure of the Web

The rising popularity of the Web has led many people to add their own servers and content and to link to new and existing resources, forming a huge graph of links and pages.

Much research has gone into understanding the structure of the Web and the behaviour of large groups of users. A popular model for describing the structure of the Web is the *Bow-Tie* model (Broder et al. 2000) shown in figure 2.4. Broder et al. showed, by performing an analysis of an AltaVista crawl of approximately 200 million pages and 1.5 billion links, that the structure of the Web looks like a bow-tie, with a central Strongly Connected Component (SCC) of around 27% of Web pages (around 56 million in the AltaVista crawl). An SCC is a set of nodes in a graph in which any node is reachable from any other. By traversing the graph or navigating the Web from any one of these pages, a further 21% of the Web pages (around 43 million) can be reached, in the *OUT* group. Another group comprising 21% of Web pages (43 million) from which any page in the SCC can be reached, but which cannot themselves be reached by traversing from pages within the SCC is called *IN*. Another group

of *TENDRILS* contains around 22% (44 million) pages which can be reached from pages in the *IN* group or from which pages in the *OUT* group can be reached. The fifth and final group contains *DISCONNECTED COMPONENTS* which cannot be reached from any of the other four groups. This group contains the remaining 9% (17 million) of pages.

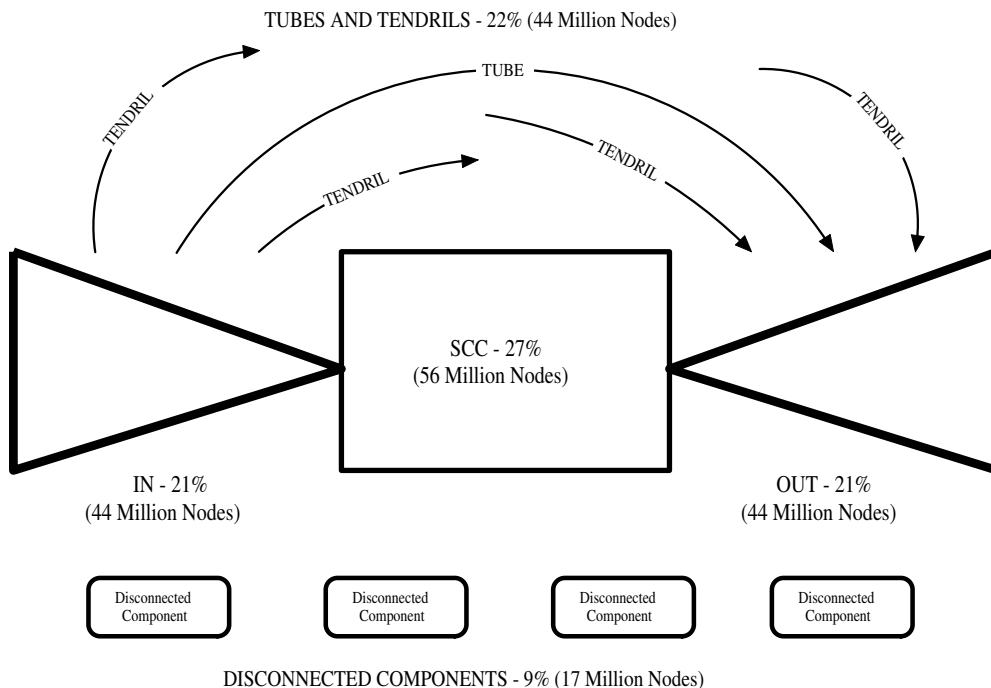


Figure 2.4: The Bow-Tie model of the Web.

The large SCC at the center of the bow-tie model is not the only such component to be found in the graph of nodes and edges which represents Web pages and their associated hyperlinks. There are a large number of such components and the distribution of their sizes follows a power law (Adamic 2000). A power law implies that small components are extremely common, whereas large components are extremely rare. Specifically, the number of components,  $y$ , of a given number of pages,  $x$  is given by the equation  $y = Cx^{-a}$ . Many man made and naturally occurring phenomena, including city sizes, incomes, word frequencies, and earthquake magnitudes, are distributed according to power law distributions.

The power law distribution of SCC sizes can be easily verified. Firstly a graph of the Web can be constructed using the crawler and graph representation systems described in chapter 5. Next, the SCCs must be found. This can be achieved using a simple algorithm consisting of two Depth First Search (DFS) operations (Aho, Hopcroft, and Ullman 1983). The component sizes can then be plotted against the frequencies on a log-log plot. The relationship  $y = Cx^{-a}$  implies that  $\log(y) = \log(C) - a \log(x)$  so the power law can be identified as a straight line with slope  $-a$ . Because of significant clustering of data points near the  $x$ -axis, regression on these plots can lead to skewed results. To prevent this, the values must be grouped into buckets of exponentially increasing sizes (Adamic 2002). From the subsequent regression an exponent value of approximately 2.5 can be obtained.



SCCs are not the only phenomena to follow a power law distribution though. Inlinks, outlinks, frequency of hits, the number of pages in a Web site and the size of Weakly Connected Components (WCCs) all follow similar relationships. The exponent values for all these power laws are shown in figure 2.5.

Feature	Exponent	Source
SCC sizes	$a \approx 2.54$	Broder et al. 2000
WCC sizes	$a \approx 2.54$	Broder et al. 2000
Indegree of Web Pages	$a \approx 2.09$	Broder et al. 2000
Outdegree of Web Pages	$a \approx 2.72$	Broder et al. 2000
Inlinks to Web sites	$a \approx 2.0$	Adamic 2002
Outlinks from Web sites	$a \approx 2.0$	Adamic 2002
PageRank of Web Pages	$a \approx 2.1$	Pandurangan et al. 2002
Pages in Web Sites	$1.647 < a < 1.853$ (95%)	Adamic 2002
Users visiting Web sites	$a \approx 2.07$	Adamic 2002
AOL Users visiting adult sites	$a \approx 1.65$	Adamic 2002
AOL Users visiting .edu sites	$a \approx 1.45$	Adamic 2002

Figure 2.5: Power Law Relationships in the Web with exponents. Percentages denote confidence levels for intervals. The PageRank citation index is discussed in section 2.6.

In order to explain these properties of the Web, new models for its growth and evolution have emerged. The original random graph models of Erdős and Rényi have proven unsatisfactory in this regard. The key to the new models is a process known as *preferential attachment* (Albert, Barabási, and Jeong 2000) in which pages which have a high indegree are more likely to be referred to by new links. This can be explained by considering a page with higher indegree as being more popular more important and better connected. It is thus more likely to be visited by a user who may then also choose to link to that page.

However, the model proposed by Albert et al. does not fit exactly with the empirical studies of real-world data. Research is ongoing to find methods to improve the model – for example, by combining preferential and non-preferential attachment (Levene, Fenner, Loizou, and Wheeldon 2002).

## 2.4 Resource Discovery

The Web is not the ultimate hypertext system that Nelson had envisaged. Nor is it the trail-blazing vision of Bush. It is however both the largest and the most successful hypertext system ever developed and it continues to grow exponentially. The ubiquitous Web has dominated hypertext research in recent years and is likely to continue to do so, and it is important to address the usability problems which arise (Nielsen 2000; Nelson 1999). The two central problems are the *resource discovery problem* and the *navigation problem*.

The resource discovery problem is that of finding a given document on the Web or finding a resource which answers a given question or which provides information about a given subject. As the Web expanded so did the difficulty of finding information within it. Early solutions involved pages of links to useful sites. One of the most popular of which evolved into the Yahoo! directory (Yahoo! 2001). The construction of directories helped but did not solve the problem and the need for an automated search solution was clear.

Search solutions developed along three lines. Directories, like Yahoo!, are still a major source of information. The largest of these is the ODP or Directory Mozilla (DMOZ) started by Netscape Communications Corporation and maintained by a large number of volunteers. These directories use human judgements to categorize pages in a large hierarchy of categories. Chapter 7 discusses how similar categorization schemes, used in information management systems, can provide hints for trail construction.

The second solution came in the form of automated Web search engines (now known simply as search engines), which found pages using extensions of information retrieval techniques described in section 2.5. Early examples of which include WebCrawler (Pinkerton 1994; Pinkerton 2002), Lycos (Mauldin 1997), Excite, Hotbot and AltaVista. Google (Brin and Page 1998) appeared shortly after, having being developed as a Stanford project and is now the most popular search engine. Fast's AllTheWeb.com (Risvik and Michelsen 2002) also gained popularity and a large influence. Further information on search engines can be found at Search Engine Watch<sup>3</sup> and Search Engine Showdown<sup>4</sup>.

These search engines have often had difficulty providing sufficiently comprehensive coverage of the Web. To provide better coverage results from several search engines may be merged into a single result. This technique is known as *Meta-Search*. Popular examples of meta-search engines have included Metacrawler, SavvySearch and Ask Jeeves, which was started as a meta-search engine, but which has subsequently acquired the Teoma search engine.

Figure 2.6 shows the architecture of a typical Web search engine consisting of a separate crawler (or robot), indexer and query engine. Various components can be combined, but are most commonly separated. For example, although it is possible to combine the crawler and indexer, search engines often separate the two to improve fault tolerance (Brin and Page 1998; Pinkerton 2002).

The crawler downloads pages, parses them for the required text content and URLs (outlinks) and adds the outlinks to a queue. The text data is stored on disk and ultimately converted to an inverted file by the indexer. The inverted file maps keyword entries to lists of matching

---

<sup>3</sup><http://searchenginewatch.com/>

<sup>4</sup> <http://www.searchengineshowdown.com/>

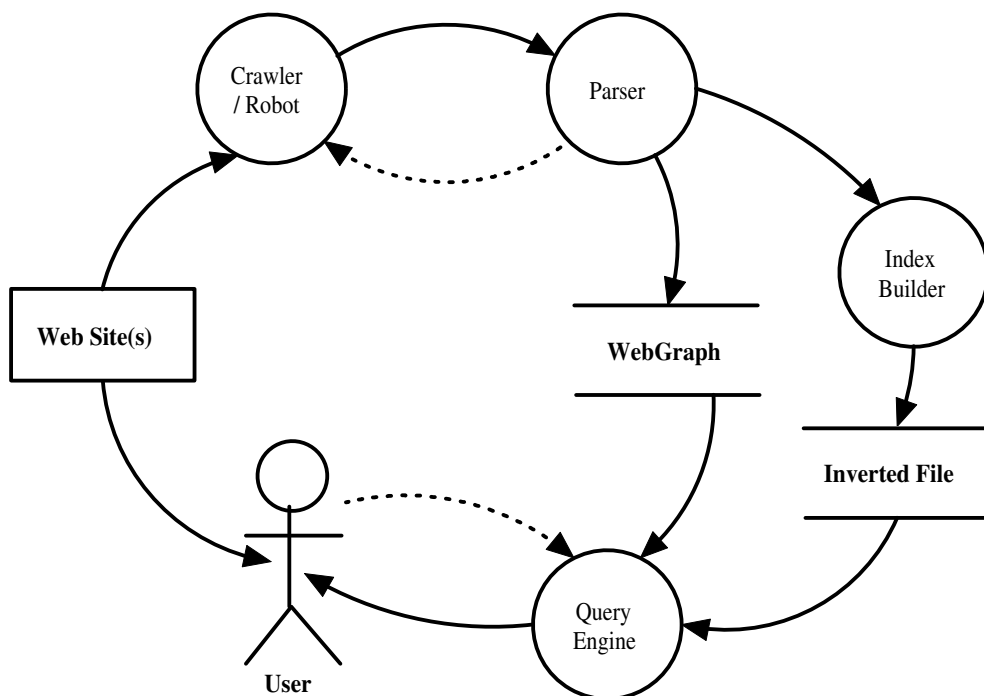


Figure 2.6: Architecture of a typical search engine.

documents known as posting lists (Harman, Fox, Baeza-Yates, and Lee 1992; Baeza-Yates, Ribeiro-Neto, and Navarro 1999). The query engine takes a user's query from a Web page form and returns a ranked list of pages. Each of the terms in the user's query is used to retrieve a posting list. Several posting lists are combined to give a ranked list of pages which is returned to the user. This basic structure of a search engine has changed little since the early engines and is unlikely to change soon, although some research has been conducted in the use of Peer to Peer (P2P) technologies for resource discovery.

Crawling pages presents several issues. Ongoing research into the order in which to crawl URLs has particular relevance to the Best Trail algorithm discussed in chapter 3. Crawling pages using a breadth-first traversal leads to good pages, but crawling pages in order of PageRank leads to better quality pages, but at a high cost (Najork and Wiener 2001; Cho, Garcia-Molina, and Page 1998). Traversal strategies are made more complicated for crawlers by the need to distribute crawling across a number of machines, to keep the rate of hits to individual servers to a minimum, to obey the instructions of webmasters and to cover a broad array of subjects and servers (Koster 1994). WebCrawler balanced these considerations by using an adapted breadth-first search across servers where pages were sorted using a heuristic algorithm (Pinkerton 2002). The Java-based Mercator robot handled distribution by splitting the crawling such that each Web site was handled by a single machine. Thus, all URLs with a common domain would then be kept within the same crawler's queue. The prevalence of links within Web sites improves the efficiency of this strategy and reduces interprocess communication (Najork and Heydon 2001; Heydon and Najork 1999a).

The Web Graph is stored separately from the inverted file, and can be stored in a relational database (Pinkerton 2002) or a customized structure (Randall, Stata, Wickremesinghe, and Wiener 2002; Guillaume, Latapy, and Viennot 2002). This data is used for link analysis to improve the results, as described in section 2.6. The parsers typically have support for HTML and plain text files and split the documents into keywords. In some systems this is being augmented by customized *filters* which provide support for the Portable Document Format (PDF), Postscript, Microsoft Office and many other document formats (Raghavan 2001). Techniques for writing such filters are discussed in section 5.6. The query engine uses information retrieval techniques to determine the documents most relevant to the query.

## 2.5 Information Retrieval Techniques

The term Information Retrieval (IR) refers to the retrieval of documents matching some given search request. IR systems have typically relied on techniques which fall into one of two distinct research areas. Natural Language Processing (NLP) solutions model the structure of the documents and queries, in an attempt to understand the text. Whilst this sounds a sensible approach, only one major search engine, Ask Jeeves, claims to use any NLP techniques and these are used only for query processing. One reason for avoiding complex NLP solutions is cost. Scalability is of fundamental importance for Web search, so most search engines rely on techniques based upon statistical properties of the text. Whilst it might appear that statistical properties should be less effective, research shows that NLP solutions rarely outperform systems using simpler statistical methods (Fagan 1987; Voorhees 1999; Cleverdon 1997).

### 2.5.1 Document Representation

In order to analyse its statistical properties, a document must be split into manageable segments or *terms*. Most IR systems split a document at the word or *keyword* level and many treat these keyword terms as independent. This leads to a popular document representation known as the “bag of words”, in which all occurrences of keywords are assumed to be independent and where ordering is assumed to be unimportant. This model is still important despite moves by all the major search engines to incorporate proximity in the ranking measures.

An alternative representation is the  $n$ -gram model. An  $n$ -gram is a sequence of  $n$  characters (including spaces) taken from the document or query text.  $N$ -grams may then be treated in similar ways to keywords. None of the major Web search engines rely on this technique as the number of  $n$ -grams in any string of text is usually much greater than the number of keywords and the cost of processing is thus much higher.  $N$ -grams have been used in the Telltale system (Miller, Shen, Liu, and Nicholas 2000) and in applications for the Department of Defense (Damashek 1995), both of which use a value of  $n = 5$ . Figure 2.7 shows the bag of words and bag of  $n$ -grams representations.

$n$ -grams have several advantages over term-based indexing. They are considered more robust against a high error rate which is useful in situations where spelling mistakes are likely, such as in corpora resulting from documents scanned using Optical Character Recognition (OCR) techniques.  $n$ -grams are also considered better for language agnostic solutions, where there is also a reduced need for language specific operations such as identifying the separation of words. Whilst identifying terms is an operation generally considered trivial for English, it is considerably more complex for languages such as Japanese and Chinese which do not use spaces between words in the same manner. Similarly,  $n$ -grams can reduce the need for language-specific stemming algorithms, such as Porter’s algorithm (Mayfield and McNamee 2003; Cavnar and Trenkle 1994; Porter 1980).

It is considered by some that  $n$ -grams are better suited when the query strings are long and that keyword-based approaches are more suitable for short queries (Mayfield and McNamee 1997; Mayfield and McNamee 1999). Given the short length of Web queries (Silverstein, Henzinger, Marais, and Moricz 1999), this may seem like a good reason to adopt keyword-

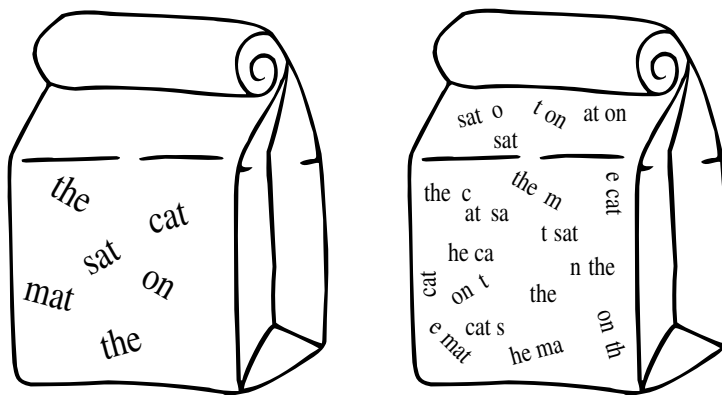


Figure 2.7: Bag of words and bag of  $n$ -grams representations. Terms are independent, and ordering is not preserved.

based approaches. However, users have been shown to adapt to the abilities of search engines<sup>5</sup>. If internet search engines performed better on verbose natural language descriptions, users might be more inclined to use them.

Thus far, the term  $n$ -grams has been used to describe character tuples. It is also possible to use systems with word  $n$ -grams – sequences of  $n$  words taken from the document or query. Attempts have been made to extend the bag of words representation by incorporating word  $n$ -grams and by incorporating topic and categorization measures (Mladenic 1998) although there seems to have been no work in combining character  $n$ -grams in the measures. The IR system developed and used for the experiments described in this thesis, is a keyword-based system and is described in greater detail in chapters 3 and 5.

## 2.5.2 Scoring Metrics

Having described a theoretical model for the representation of the documents (and queries) the measures which are used to calculate the score, or *relevance*, of a document must now be described. This is performed by computing the *similarity* of a document to the query. One of the most popular models used to achieve this is the Vector Space Model (VSM). In this model, all documents and queries are modeled as vectors in  $n$ -dimensional space, where  $n$  is the number of keywords common to both the query and the document. Incorporating words which are in either one or the other but not both leads to zero values in the resulting equations which can be simply cancelled out. The similarity is then represented as the cosine of the angle between the vectors. In practice this can be computed very efficiently by computing the sum of the normalized scores for all the query keyword weights,  $w_{qk}$  and the document keyword weights,  $w_{dk}$ . The similarity between a document  $D$  and a query  $Q$  can be computed

<sup>5</sup> One such example of this behaviour, as suggested by Nick Craswell of CSIRO, is of people who now use Google instead of using bookmarks or favourites for identifying commonly visited sites. Before Google, links to such sites would be stored. Now, due to improved home-page finding ability on the part of the search engine, people can trust that they will be directed them to their requested page, encouraging an increase in home-page finding queries. A classic example of this can be seen from the number of queries for sites such as CNN in the aftermath of the September 11th attacks.

as :

$$\text{Similarity}(Q, D) = \sum_{k=1}^t w_{qk} \cdot w_{dk}$$

Weighting schemes for both the document and query terms can now be defined. The most basic metric used is the *Term Frequency*,  $tf$ , defined as the number of times a query term appears in the document. Almost all modern information retrieval systems use this metric in some form. Some common variations are  $tf$ ,  $\log(tf)$ ,  $\log(tf + 1)$  and  $\log(tf) + 1$  (Grossman and Frieder 1998). The reasoning behind this measure is simple – the more times a document refers to a given word or phrase, the more likely it is that the document is relevant to discussion of that subject.

However, not all words are equal. Some words are more useful than others for describing a document, or for discriminating between documents. Two schemes are used for dealing with this discrepancy. Firstly, a list of “stop words” may be constructed. Stop words, for example “the”, “and” and “which”, appear in many documents and are considered particularly poor at discriminating between documents. Any occurrences of these words are removed from the query<sup>6</sup>. Secondly, document and query terms are weighted according to the frequency with which they occur in the corpus. The Inverse Document Frequency,  $idf$ , is typically defined as

$$idf_w = \log \frac{N}{df_w}$$

where  $N$  is the number of pages in the corpus and  $df_w$  is the number of pages in the corpus in which the term  $w$  appears. The variations  $\frac{N}{df_w}$  and  $\log \frac{N}{df_w + 1}$  are also used (Grossman and Frieder 1998). It is also possible for  $df_w$  to be defined as the total number of occurrences of  $w$  in the corpus.

The product of  $tf$  and  $idf$  forms the commonly used  $tf.idf$  measure. However,  $tf$  is still biased towards long documents, as long documents are inevitably more likely to contain a given term than short documents. The solution to this is to normalize the document term weights against some global document weight. One such normalized  $tf.idf$  system is that presented in Salton and Buckley 1998 and described as the “best fully weighted system”. This prescribes a document term weight of

$$\frac{tf \cdot \log \frac{N}{df_w}}{\sqrt{\sum_{vector} tf \cdot \log \frac{N}{df_w}}}$$

and a query term weight of

$$\left(0.5 + \frac{0.5tf}{tf_{max}}\right) \cdot \log \frac{N}{df_w}$$

the products of which are summed when the posting lists are read, to conform to the VSM definition.

---

<sup>6</sup> Stop words are not always useless. They may be considered ineffective for discriminating between documents, but may still be useful in phrases, such as “to be or not to be”, or in language identification (Wood 1998).

A popular variant of the *tf.idf* measure is the Okapi measure (Robertson and Walker 1999; Singhal 2001) in which the similarity between document and query is given by

$$\sum_{t \in Q, D} \log \frac{N - df_w + 0.5}{df_w + 0.5} \cdot \frac{(k_1 + 1)tf}{(k_1((1 - b) + b\frac{dl}{avdl})) + tf} \cdot \frac{(k_3 + 1)qtf}{k_3 + qtf}$$

where  $1.0 < k_1 < 2.0$ ,  $b \approx 0.75$  and  $0 < k_3 < 1000$  are constants, and where *dl* and *avdl* are the document length and average document length respectively. Both of these use normalization techniques to reduce the unfair selection of long documents. The methods proposed by Salton and Buckley sometimes over-compensate, creating a bias towards short documents. Singal et al. propose a solution to this problem based upon a technique called “pivoted document length normalization” (Singhal, Buckley, and Mitra 1996).

A complete alternative to the *tf.idf* variants is the probabilistic IR model popularized by Spark-Jones and Van Rijsbergen amongst others. In this model, documents are returned in the order of probable usefulness. The basis for computing these probabilities lies in earlier work by Bayes on probability. For further information see Van Rijsbergen 1979; Crestani, Lalmas, Rijsbergen, and Campbell 1998; Baeza-Yates and Ribeiro-Neto 1999; Jones, Walker, and Robertson 2000.

### 2.5.3 Evaluation Metrics

Having defined methods for retrieving a set of documents, methods for evaluating the quality of the results are now described. A corpus of  $N$  documents will be considered, of which  $n_1$  documents have been judged as relevant to a query,  $q$ , by the human user or assessor and from which  $n_2$  are retrieved by an information retrieval system. Figure 2.8 shows the relationship between the sets from which the metrics shown in figure 2.9 can be defined. These are related by the formula  $RG(1 - P) = FP(1 - G)$ .

	Returned	Not Returned	
Relevant	$w$	$x$	$n_1 = w + x$
Not Relevant	$y$	$z$	
	$n_2 = w + y$		$N = w + x + y + z$

Figure 2.8: Korfhage’s matrix for evaluation metrics

$$\begin{aligned} Resolution &= \frac{n_2}{N} & Elimination &= \frac{N - n_2}{N} \\ Noise &= \frac{n_2 - w}{n_2} = \frac{y}{n_2} & Omission &= \frac{n_1 - w}{n_1} = \frac{x}{n_1} \\ F = Fallout &= \frac{y}{N - n_1} & G = Generality &= \frac{n_1}{N} \\ P = Precision &= \frac{w}{n_2} & R = Recall &= \frac{w}{n_1} \end{aligned}$$

Figure 2.9: Software evaluation metrics

Precision and recall are considered the most useful metrics for evaluation. Unfortunately there is always a trade-off between these two metrics. It is trivial to obtain high precision (and low noise) – by returning a single document, at the cost of low recall. It is equally trivial



to obtain high recall – by returning all documents, at the cost of low precision. By plotting the values of precision against recall at various points we obtain a saw-toothed *recall-precision curve* for an IR system returning an ordered set of results. However, it is often inconvenient to assess the precision and recall at each point, so values are plotted at fixed percentages of the returned results. Using the values at 25%, 50% and 75% of the returned results gives *3 point precision*. 11 point precision is determined using the points 0%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90% and 100%. An *average 11-point precision* is the average of these values.

A simpler alternative is to list only the precision at a fixed point. This is a common technique in evaluating Web search engines, where the top- $n$  precision is given. This is the precision for the top  $n$  documents as found by the search engine. Common values for  $n$  are 10, 20 or 100. In evaluating search engines, recall is often ignored as the number of documents is often huge, and users rarely examine more than the first few pages of results (Silverstein, Henzinger, Marais, and Moricz 1999; Wolfram, Spink, Jansen, and Saracevic 2001). In this situation, coverage is deemed more important and search engines consistently compete for larger indexes (Sullivan 2001; Lawrence and Giles 1998; Lawrence and Giles 1999).

#### 2.5.4 The Text REtrieval Conference

The Text REtrieval Conference (TREC) was set up by the American National Institute of Standards and Technology (NIST) to provide a common forum for developers of IR systems to compare results. It has been held annually at Gaithersburg, Maryland since November 1992. The TREC WebTrack was introduced at the 8th conference (TREC-8) and has subsequently replaced both the Ad-Hoc and Very Large Collection (VLC) tracks.

Four test collections are currently distributed by the Commonwealth Scientific and Industrial Research Organisation (CSIRO) for the use in this conference track and for other research<sup>7</sup>. The WT100g was formerly known as VLC2 and represents a large Web crawl. WT10g is a subset of WT100g and WT2g is a subset of WT10g. The .GOV test corpus was used in both tasks (named-page finding and topic distillation) in the 11th conference (TREC-2002) (Craswell and Hawking 2002). Figure 2.10 shows the relative sizes of these corpora.

Corpus	Size	Documents
WT100g	100 gB	18,500,000
.GOV	18 gB	Unknown
WT10g	10 gB	1,690,000
WT2g	2 gB	247,491

Figure 2.10: TREC WebTrack Collections

The TREC corpora provided are a useful resource for evaluating search techniques. Several sets of queries are provided for testing and many of the documents which are likely to be retrieved have been evaluated by the NIST assessors. However, the corpora are not without substantial limitations:

<sup>7</sup> See [http://www.ted.cmis.csiro.au/TRECWeb/access\\_to\\_data.html](http://www.ted.cmis.csiro.au/TRECWeb/access_to_data.html) for details of distribution. All data is supplied on CD or tape.

## Size and Scalability

The size of the TREC corpus is given by many people as a major problem. CSIRO currently offer four test collections. WT100g, WT10g and WT2g are the 100Gb, 10Gb and 2Gb collections used on earlier tests. A fourth collection, based on a crawl of the .gov domain comes in at 18Gb. This seems like a lot of data, until the terabytes of information indexed by commercial search engines are considered – along with the fact that TREC tests have only used the 10Gb and 18Gb collections. There is a real and genuine concern that results on these collections may not indicate viable solutions for the Web.

However, interesting problems can still be asked of a smaller corpus - particularly relating to the area of site search. There are millions of Web sites which are large enough to require separate search facilities. Usability expert, Jakob Nielsen has stated the importance of search facilities on Web sites and the behaviour of users in expecting to be able to search sites (Nielsen 1997; Nielsen 2000). Web sites are a good example of corpora which are of a similar (or usually much smaller) size to the TREC collections and where good search facilities are essential.

There is an often-ignored issue with larger engines scaling down. Economies of scale may be lost and the cost-per-megabyte of indexing may increase. Moreover, a lack of data may affect the quality of results. For example, link text on the Web provides a strong indicator of a sites content (Brin and Page 1998; Cutler, Deng, Maniccam, and Meng 1999). With smaller corpora, there are fewer links and the quantity of link text is naturally reduced. A system which returns 1000 pages from the Web in answer to a query might only return a very few pages on a site. More resources may be granted to some site searches because of the smaller corpus size and the mission-critical nature of the Web site. For example, a user failing to find a Web site with Google may be inconvenienced. A user failing to find a product at Amazon may mean lost revenue. This increased importance should imply an increase in search quality, but this can only be achieved with proper evaluation. What is needed is a mix of sizes. The WT2g and WT100g collections provide this. Unfortunately, they do not have the same depth of query information and relevance judgements, so analysis is restricted to the WT10g and .GOV collections.

## Language

The Web is a multilingual resource. Google's Zeitgeist<sup>8</sup> shows that almost half of the queries submitted were in non-english text and that the proportion of non-english queries is growing. Web search facilities (even for site search) need to be able to work with documents in multiple languages. The TREC corpora are all heavily biased towards English with relevance judgements only given for English language questions.

## Relevance Judgements and Ranking

All TREC documents are assumed to be irrelevant, relevant or highly relevant to a given query, and are judged only on the text content. In practice there are many different levels of

---

<sup>8</sup><http://www.google.com/press/zeitgeist.html>

importance and relevance which are not considered. It has been shown that there are considerable discrepancies between the performance of Web search systems and those algorithms which perform best under TREC conditions. TREC algorithms which take no account of linkage information have still been able to achieve very high scores with respect to TREC measures.

The most important issue with regard to evaluation of the results in this thesis is that TREC judgements fail to take navigation possibilities into account. A page may not contain the answer to a query, but may link to several which do. Such a page would be considered non-relevant in TREC, but might be useful to someone who was willing to investigate further. The TREC corpora are also limited in the information they provide. CSIRO does not provide any data on query history, navigation history, bookmarks or other information which might be available to either a search engine or client-side search tool for a real user. Hence testing of personalization is beyond the scope of the TREC experiments.

## 2.6 Web Page Metrics

The size, structure and markup of the Web leads to a number of possible improvements which can enhance the utility of the search results. Some of these will be briefly discussed. Alternative descriptions of most of the metrics may be found in Baeza-Yates and Ribeiro-Neto 1999. Many of these metrics can be expressed in alternative forms, such as the result of computing principal eigen vectors on a matrix describing the Web's structure.

### 2.6.1 HTML Tag Weighting

With HTML documents, further term weighting is possible based upon the position of the term in the HTML tags. Two studies have been conducted into the effects of tag weighting schemes. The first was conducted using a small corpus and a set of 20 questions and incorporated anchor text from pages linking to the given page, a commonly used measure also suggested in Brin and Page 1998. The first results were presented in Cutler, Shih, and Meng 1997. The experiments were subsequently repeated using a Genetic Algorithm (GA) and presented in Cutler, Deng, Maniccam, and Meng 1999. The second study, also refers to a GA used to learn HTML tag weights and was conducted using a larger corpus (the TREC WT2g corpus) but did not consider text on inlinks (Kim and Zhang 2000). Figure 2.11 shows the schemes proposed in both of these papers.

Tag Type	Weights
Plain tags (no html markup)	1
H1 and H2	6
H2 ... H7	1
STRONG, B, EM, I, U, DL, OL, UL	8
TITLE	4
Inlink Text	8

(a) Cutler et al.

HTML Tag	Weights
Title	$0.5584 \pm 0.2822$
Header	$2.3425 \pm 0.2614$
Bold	$0.7060 \pm 0.2061$
Italic	$1.0192 \pm 0.3128$
Anchor	$1.7634 \pm 0.1306$

(b) Kim et al.

Figure 2.11: HTML tag weighting schemes proposed by (a) Cutler et al and (b) Kim et al.

Thus  $tf$  can be redefined as the weighted sum of the number of equal terms in the body of the document, the number of equal terms in the headings and titles of the document and the number of equal terms defined in links referring to the document. Equality in this case may be case-sensitive or case-insensitive. The  $tf.idf$  values given as document term weights are re-used as the term weights for the summarization algorithm described in chapter 5.

Traditional IR approaches work well for flat-text systems where the authors can be relied on to follow consistent style, write metadata comments that would be helpful to the user and be non-competitive. However, Web search engines are heavily “spammed” to influence results. Search engine spamming involves manipulating meta-tags, the insertion of text which is invisible to the user and the creation of “nepotistic links” (Davison 2000). The quality of the data is low and text alone cannot be relied on for evaluation. One solution to this is link analysis – the science of extracting information from the hyperlinks available and using this information to improve results. The roots of this research stem from much earlier work in

citation analysis and typically stem from the idea that a link conveys approval of the cited page.

### 2.6.2 Landmark Nodes

An early attempt at using link structure was described in Mukherjea and Foley 1995. *Landmark nodes* were defined as those nodes in the top 10% of those in the graph when scored using the formula

$$importance = (I + O) * wt_1 + (SOC + BSOC) * wt_2$$

where  $wt_1 + wt_2 = 1.0$ ,  $I$  is the number of inlinks,  $O$  is the number of outlinks,  $SOC$  is the number of second-order-outlinks (outlinks of outlink) and  $BSOC$  is the number of back-second-order-outlinks. Suggested values are  $wt_1 = 0.4$  and  $wt_2 = 0.6$ .

This importance metric was later extended to incorporate access frequency and depth. Using the importance measure above as a measure of “structural importance”, Mukherjea proposed the formula

$$importance = k_1 \frac{S}{max(S)} k_2 \frac{A}{max(A)} k_3 \frac{D}{max(D)}$$

where  $S$  is the importance measure given above,  $D$  is the depth of a page in a Web site,  $A$  is the frequency with which this page is accessed and  $k_1$ ,  $k_2$  and  $k_3$  are constants. Once the landmark nodes are identified they can be used to provide context for each page the user visits as he navigates through the site (Mukherjea and Hara 1997).

### 2.6.3 Hubs and Authorities

Kleinberg 1998 proposed an algorithm for Hyperlink-Induced Topic Search (HITS) which gives two scores for each page with respect to a given query. These are a *hub* score and an *authority* score. It was proposed that the best hubs and authorities would be returned as results for any given query.

The algorithm, which formed the basis for IBM’s *Clever* project, starts by submitting a query to a search engine, which returns a small initial set, of around 100 pages. This set is expanded by examining each node and adding all the pages to which the node links and some or all of the pages linking to this node. This gives a new set which should contain the best pages for that query. The next stage of the algorithm is to compute the best hubs and authorities, using the scores:

$$H(p) = \sum_{l \in outlinks(p)} A(l)$$

which is the hub score and

$$A(p) = \sum_{l \in inlinks(p)} H(l)$$

which is the authority score. One major problem with this implementation is the need for a large seed set from which the returned results will be selected.

### 2.6.4 PageRank

A simpler metric was PageRank – used by the Google search engine (Page, Brin, Motwani, and Winograd 1998; Page 1998). A PageRank is a single number assigned to each page in the Web graph. The higher the number, the greater the supposed importance of the page. PageRank was originally defined as

$$P(j) = \sum_{i \in B_j} \frac{P(i)}{|F_i|}$$

where  $B_j$  denotes the set of page linking to  $j$  (backlinks) and  $F_i$  denotes the set of pages linked to by  $i$  (forward links). This was altered to

$$P(j) = \frac{(1 - \beta)}{N} + \beta \sum_{i \in B_j} \frac{P(i)}{|F_i|}$$

in an attempt to handle the problem of “rank sinks” where rank is “lost” due to the lack of forward links.  $\beta$  is referred to as a damping factor and is typically set to values of around 0.85. There are several ways to view this rank, but one of the simplest is that PageRank is the probability that a “Random Surfer” will be at a given page at any given time.

The papers by Kleinberg and Page et al. are amongst the most well-known and well-cited in the field and have subsequently formed the basis for a great deal of further research. It is interesting to note that the system using PageRank (Google) has become the dominant search engine whilst that using the Hubs and Authorities model (Clever) has failed to achieve the same level of success. It is the author’s belief that the primary reason for this is the complexity of the calculations. Research has shown that response time is the biggest factor in Web usability (Nielsen 2000), and that sub-second response time is required by search engines under heavy load. The complexity of Kleinberg’s algorithm leads to an unacceptable cost in resources.

### Topic-Specific PageRank

There have been several attempts at introducing a topic or query sensitive variant of PageRank. A topic sensitive PageRank was introduced in Haveliwala 1999 in which individual PageRank vectors were computed for each of the top-level categories in the ODP – giving 16 topic specific rank vectors.

Richardson and Domingos went further and proposed a system in which a term-dependent PageRank is computed for each unique term in the corpus, potentially doubling the inverted index size. This score is then combined with an IR metric at query time (Richardson and Domingos 2002). The proposed scheme extends the PageRank formula to incorporate page relevance with respect to a query term  $q$  to give

$$P_q(j) = (1 - \beta)P'_q(j) + \beta \sum_{i \in B_j} P_q(i)P_q(i \rightarrow j)$$

where  $P'_q(j)$  specifies the probability of a surfer visiting page  $j$  when not following links and  $P_q(i \rightarrow j)$  is the probability of a transition to  $j$  given that the surfer is at  $i$ .  $P_q(i \rightarrow j)$  is

defined, using the relevance  $R_q(i)$  of a page  $i$  with respect to a query  $q$ , by

$$P_q(i \rightarrow j) = \frac{R_q(j)}{\sum_{k \in F_i} R_q(k)}$$

This formula is used to compute PageRank for all matching nodes for each given term.

Lempel showed that both PageRank and Kleinberg's HITS metrics could be influenced by a phenomenon called the Tightly Knit Community (TKC) effect (Lempel and Moran 2000). The mutually reinforcing nature of the Hubs and Authorities metric can be compromised by small highly interlinked groups and these will dominate search results regardless of quality. Lempel's work on the Stochastic Approach for Link-Structure Analysis (SALSA) lead to an extension of HITS that was less vulnerable to the effect.

An interesting tangent to the work on PageRank was provided by SimRank (Jeh and Widom 2001) which applied the principles of PageRank to the problem of similarity measurement. They used the concept of a node pair graph, a graph  $G^2 = (N^2, E^2)$  constructed from an original graph  $G = (N, E)$  in such a way that a node  $(x, y)$  exists in  $N^2$  for each  $x$  and  $y$  in  $N$  and a link  $((x, y), (v, w))$  exists in  $E^2$  if and only if there exists two links  $(x, v)$  and  $(y, w)$  in  $E$ . Upon this graph is computed a SimRank equivalent to the original PageRank, giving a similarity measure for each document pair. They also considered a similarity measure for bipartite graphs based upon Kleinberg's HITS algorithm.

### 2.6.5 Combining Metrics

To be useful in IR systems, the PageRank, Hubs and Authorities score must be combined with existing text-based metrics. All the major search engines combine text-based retrieval techniques with link-based ranking measures, but there is very little information on how these metrics are combined. WebCrawler (Pinkerton 2002) combines *external* page scores, generated from link-analysis (in this case the number of distinct servers linking to a page), using the formula

$$score(doc) = a.relevance(doc, query) + (1 - a).external(doc)$$

where

$$a = \max\left(\frac{2}{3}, 1 - \frac{1}{20} \log(C)\right)$$

where  $C$  is the number of documents matching the query. This allows pages for queries where the keywords discriminate between pages well to be scored predominantly on the IR score. For queries where keywords fail to discriminate between documents, the pages will be scored to a greater extent on the citation weight.

## 2.7 Non-Linear Search

Having established the importance of link analysis in improving the quality of Web search results, and having a partial solution to the resource discovery problem, search engine research has recently progressed to finding better ways to present results which show context, metadata and link structure. Traditional search engines have focused on returning a simple ordered set of pages. By removing this constraint, many possibilities for research present themselves.

### 2.7.1 Question Answering

AskJeeves is a popular search engine which takes natural language queries and attempts to provide Web pages which supply the answer. There are many situations in which a single answer to a question is all that is required. Three systems have been developed to provide the answers:

**Mulder** (Kwok, Etzioni, and Weld 2001) takes a question and returns a list of possible answers with the confidence of each answers validity. The question is given in a natural language (English) and the corresponding parse tree is generated from it. This parse tree is used by a *classifier* to identify the type of answer which should be expected. A *query formulator* translates the parse tree into a series of search engine queries, which are issued in parallel to the search engine. The relevant pages are downloaded and parsed by an *answer extraction module* to generate small extracts, or *summaries*, which are used to select candidate answers. These candidates are then scored and ranked by an *answer selector* before being presented to the user. Kwok et al. claimed superior results to Google but subsequently withdrew the engine due to performance concerns.

**AnswerBus** (Zheng 2002) is a similar system, with improved multi-lingual support – questions can be posed in English, German, French, Spanish, Italian and Portuguese, but only answered in English. Relevant pages are downloaded and parsed from five search engines. Zheng claims that the “current rate of correct answers to TREC-8’s 200 questions is 70.5% with the average response time to the questions being seven seconds”.

**NSIR** performs the same basic operations, with probabilistic techniques for passage and sentence extraction, and answer ranking (Radev, Fan, Qi, Wu, and Grewal 2002). It is claimed that the algorithm, Probabilistic Phrase Reranking (PPR) “achieves a total reciprocal document rank of .20 on the TREC 8 corpus”.

### 2.7.2 Category-based Clustering

A simpler alternative is to re-order the resulting pages, not in order of relevance, but grouped or *clustered* according to the subject or topic of the returned pages.

Mase reported on experiments using a simple keyword-based dot-product categorization algorithm (Mase 1998). Normalized keyword frequency vectors were stored in a “knowledge base”, which was reduced in size using a choice of filtering techniques.

Pratt et al. used a domain-specific knowledge base in the development of Dynacat, a system for dynamically categorizing search results (Pratt, Hearst, and Fagan 1999). Using a medical



database as an example, they classified queries by type, such as “problem–diagnoses” or “treatment–problems”. The knowledge base held these query types, which could be selected by the user, acceptable category areas and a mapping between terms and their semantic types. Processing involved taking a document and comparing its terms against a set of criteria. The matching criteria determined the category headings. The final results were then grouped under these categories.

Hearst, et al. developed a tool called Scatter/Gather, which also clustered search results. These clusters could then be selected or deselected before reclustering, thus eliminating the need to view large numbers of documents (Hearst, Pedersen, and Karger 1995; Hearst and Pedersen 1996; Hearst 1997).

Grouper (Zamir and Etzioni 1999) is an extension of the HuskySearch meta-search engine which uses Suffix Tree Clustering (STC) to cluster search results based on the content of the document summaries. The STC algorithm works by stemming the document keywords and removing surplus markup and punctuation, using a suffix tree to organise the documents and create base clusters then combining these base clusters into the clusters to be shown to the user.

Vivisimo is another meta-search engine which organizes results into a classification hierarchy or tree. This classification tree is displayed on the left hand side alongside the main results, as shown in figure 2.12. Users can access search results through the normal sorted list or navigate through this hierarchy.

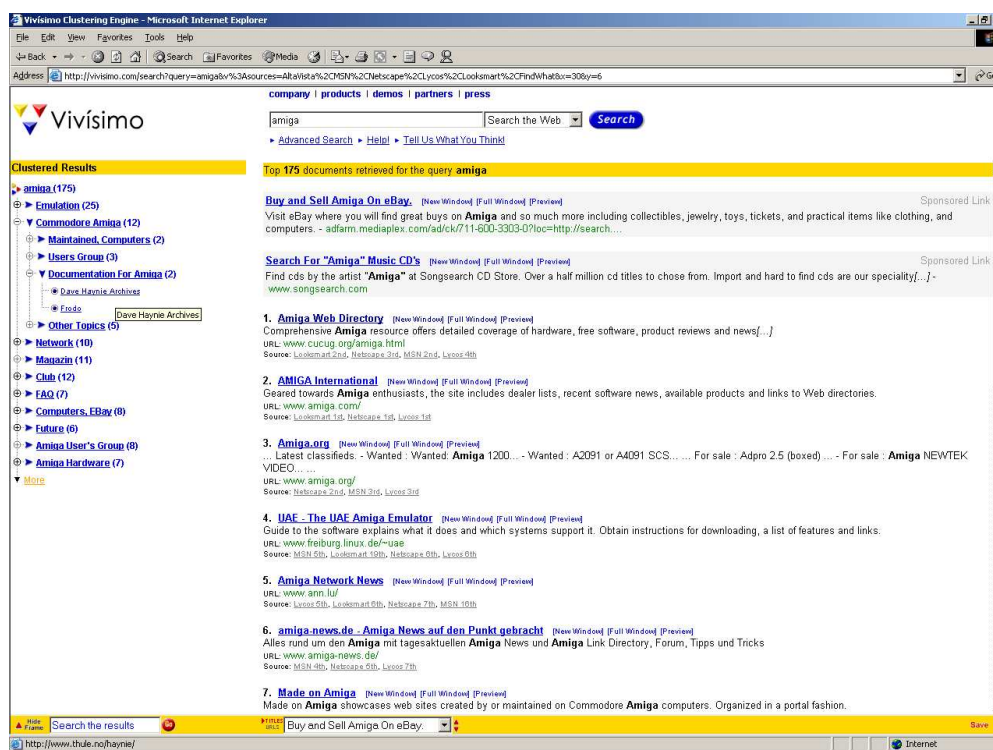


Figure 2.12: Vivisimo’s clustered search results.

One view of the clustering problem is that it represents the task of finding the  $k$  centroids in a set of  $n > k$  points such that the sum of the euclidean distances to these points is minimized. Drineas, Frieze, Kannan, Vempala, and Vinay 1999 took this definition, relaxed it and attempted to find efficient solutions to the relaxed problem using an algorithm based on the Singular Value Decomposition (SVD) of the adjacency matrix. Their work at Yale university formed the basis for the Manjara clustering engine which was widely cited but which has subsequently been withdrawn from service.

### 2.7.3 Link-based Clustering

An alternative method of grouping pages is using clusters of pages formed by analysing the link topology.

Flake et al. define a *Web community* as a set of nodes which contains more links (both inlinks and outlinks) to nodes which are members of that set than to nodes which are not members (Flake, Giles, and Lawrence 2000; Flake, Lawrence, Giles, and Coetzee 2002). They identify such communities using a variation of the Max-Flow-Min-Cut algorithm. It should be noted that, unlike those described in Kleinberg, Gibson, and Raghavan 1998, these communities are independent of query or topic and can be computed a-priori.

Li et al. proposed the idea of an *information unit* – a small set or cluster of connected pages returned in answer to a user’s query (Li, Candan, Vu, and Agrawal 2001). Specifically, an information unit is a weighted subgraph or Steiner tree extracted from the complete Web graph. The problem of computing even a single optimal Steiner tree is NP-hard, so the algorithm given computes  $k$  suboptimal information units of pages which together contain all the query keywords. Li et al. also show extensions to the algorithm to deal with clusters which contain less than all the query terms. What they do not discuss is any mechanism for displaying the information units to the user or for dealing with the inevitable volume of redundant information which will arise from deploying an information-unit system in a real-world environment.

## 2.8 The Navigation Problem

The second major problem on the Web is the *navigation problem* (Levene and Loizou 1999), which can be expressed simply as “How can we stop people from getting lost in hyperspace?” A person is said to be “lost in hyperspace” if they are navigating (or browsing) the Web (or some other hypertext) and are either unsure of where they are relative to another page, unsure of which link to follow in order to find what they’re looking for, unsure of where they’ve been or unsure of where they will get to when they follow a given link.

The navigation problem is distinct from the resource discovery problem, but the difference is subtle. One way to demonstrate the difference is to consider the questions a user might ask when faced with each problem. If the user is asking “where can I find this ?” it’s a resource discovery problem. If a user is asking “how do I get to that ?” it’s a navigation problem. Typically the user’s need can be satisfied by a URL in the first case or by a set of links or a tour in the second. Users who are given different tasks may face either or both of these problems, and react accordingly. This was noticed in early hypertexts where users exploring the corpus to learn about a subject might make heavy use of guided tour facilities, whereas those trying to answer specific questions would tend to use index or search facilities (Hammond and Allinson 1989a; Nielsen 1989).

Whilst search engines help to solve the navigation problem by providing facilities to help people find what they’re looking for and start the navigation process, they fail in that they do not provide support for users once they have their initial results, and in that they do not show results in the context of the pages around them. They also contribute to the problem by directing users to the middle points of a Web site or hypertext with no reference to important points, such as home pages, etc.

## 2.9 Navigation Aids

There have been many attempts to help solve the navigation problem through the construction of navigation aids, many of which attempt to suggest single links for users to follow or provide visualization strategies for showing the relationships between large numbers of pages.

### 2.9.1 Link Suggestion

Autonomy developed the Kenjin system for suggesting “interesting” links, based on Bayesian probability theory. The Adaptive Probabilistic Concept Modelling (APCM) used by Autonomy is designed to analyse “correlation between features found in documents relevant to an agent profile” (Autonomy 2003). WebWatcher is a system for helping user navigate by suggesting single links from pages, using a proxy to redirect browsing activity via a server (Joachims, Mitchell, and Freitag 1995; Joachims, Freitag, and Mitchell 1997). Gori, Maggini, and Martinelli 1999 describes a system to Navigate AUTonomously and Target Interesting Links for USers (NAUTILUS). NAUTILUS is a proxy-based navigation aid which utilises the interesting technique of representing HTML Web pages as graphs and using these as inputs to a neural net.

An alternative is not to suggest links but to try and provide more information about the links being selected. TileBars are small images which can be displayed next to a link to show the relative position in the document of each of the query terms (Hearst 1995). These can be placed next to search results, where a score or PageRank might otherwise be placed.

Weinreich and Lamersdorf describe a proxy-based system which adds elements to pages to give highly descriptive pop-up windows for each link (Weinreich and Lamersdorf 2000). The pop-ups display metadata such as the title, author, size and Multipurpose Internet Mail Extensions (MIME) type (Freed and Borenstein 1996) of each page linked to. Geisler describes a similar pop-up system, but with separate subsections for “Preview”, “Overview” and “History”, each of which is activated by rolling over the section title with the mouse pointer. Their system displays similar information, including the number of outlinks from each page and a small thumbnail preview (Geisler 2000).

### 2.9.2 Site Maps

Site Maps present an overview of an entire Web site from which visitors can reach all parts of the site very quickly and provide a great deal of context. Tabular formats are very popular, being used by Lycos (as shown in figure 2.13), Cnet<sup>9</sup>, Intel<sup>10</sup>, Google<sup>11</sup> and Yahoo!<sup>12</sup> amongst others. Circular layouts are less popular. In these layouts, pages radiate from a central hub. Figure 2.14 shows an example of a circular layout combined with other elements. Apple have previously experimented with tabular and circular layouts and are now presenting results in a long list, as shown in figure 2.15. This is another very popular format. Single column lists

---

<sup>9</sup> <http://www.cnet.com/sitemap/0-2253447.html>

<sup>10</sup> <http://www.intel.com/intel/nav/sitemaps.htm>

<sup>11</sup> <http://www.google.com/dirhp>

<sup>12</sup> <http://www.yahoo.com/>

are used by ESPN and CNN. Two-column lists (technically these are also tables) are used on sites belonging to organizations such as NASA, New Scientist<sup>13</sup> and the IMF<sup>14</sup>. Typically lists are either clustered together or sorted. Clustering may be done by topic, subject or department whilst lists may be sorted in alphabetical or chronological order.

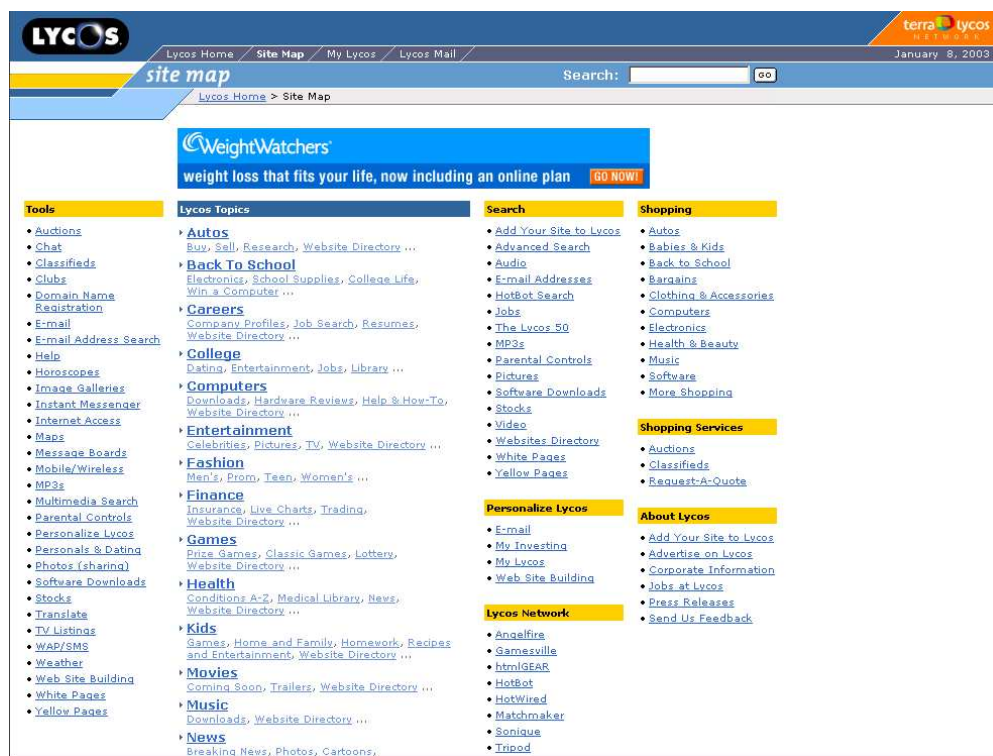


Figure 2.13: Lycos's sitemap. The pages are shown in tabular format.

Some site maps use metaphors to display the results. Figure 2.16 shows a sitemap from Jam Design which uses a spaceship metaphor. Other metaphors used in Web sites include circuit boards, solar systems and company products (Kahn and Lenk 2001). Site maps can also mirror geographical maps, as in the example from the American Federal Aviation Administration (FAA) shown in figure 2.17. Other examples including Peter Burden's map of UK Universities<sup>15</sup> and London Underground's TubeGuru<sup>16</sup> – a clickable Tube map showing entertainment options in close proximity to each station.

<sup>13</sup> <http://www.newscientist.com/thesite/tssitemap.jsp>

<sup>14</sup> <http://www.imf.org/external/map.htm>

<sup>15</sup> <http://www.scit.wlv.ac.uk/ukinfo/uk.map.html>

<sup>16</sup> <http://www.tube.tfl.gov.uk/guru/index.asp>

## ExpPASy site map

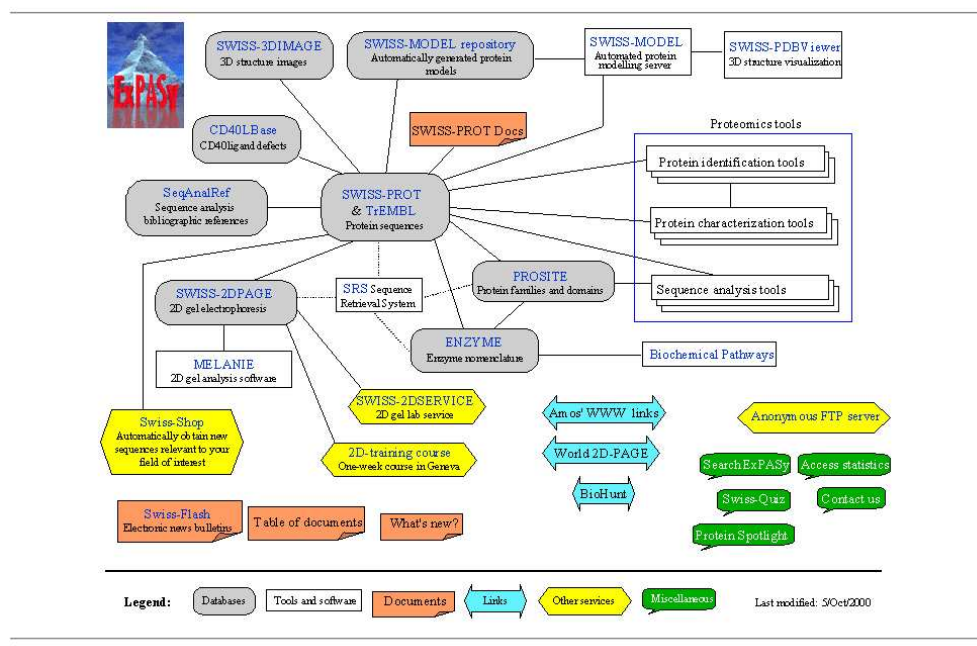


Figure 2.14: ExpPASy's sitemap.

Some sitemaps attempt to use form elements for navigation. Figure 2.18 shows a sitemap from the Journals of the American Medical Association (AMA) where HTML `select` elements are used to select areas of the site. The sitemap for the american Small Business Administration (SBA)<sup>17</sup> takes this to further extremes where every option is embedded in such a list. Merriam-Webster<sup>18</sup> presents the hierarchy in an interactive tree which hides some of the complexity of the site. However, these designs all make navigation more difficult, require more clicks (typically one to select an element and one to submit unless JavaScript is used) and the links are less likely to be followed by robots. Many site maps also fail in conveying multiple levels of the site's structure and usability tests have shown that users often overlook site maps. It is vital that maps aid the navigation process – not provide further challenges (Nielsen 2002).

<sup>17</sup> <http://www.sba.gov/map.html>

<sup>18</sup> [http://www.m-w.com/map\\_new.htm](http://www.m-w.com/map_new.htm)

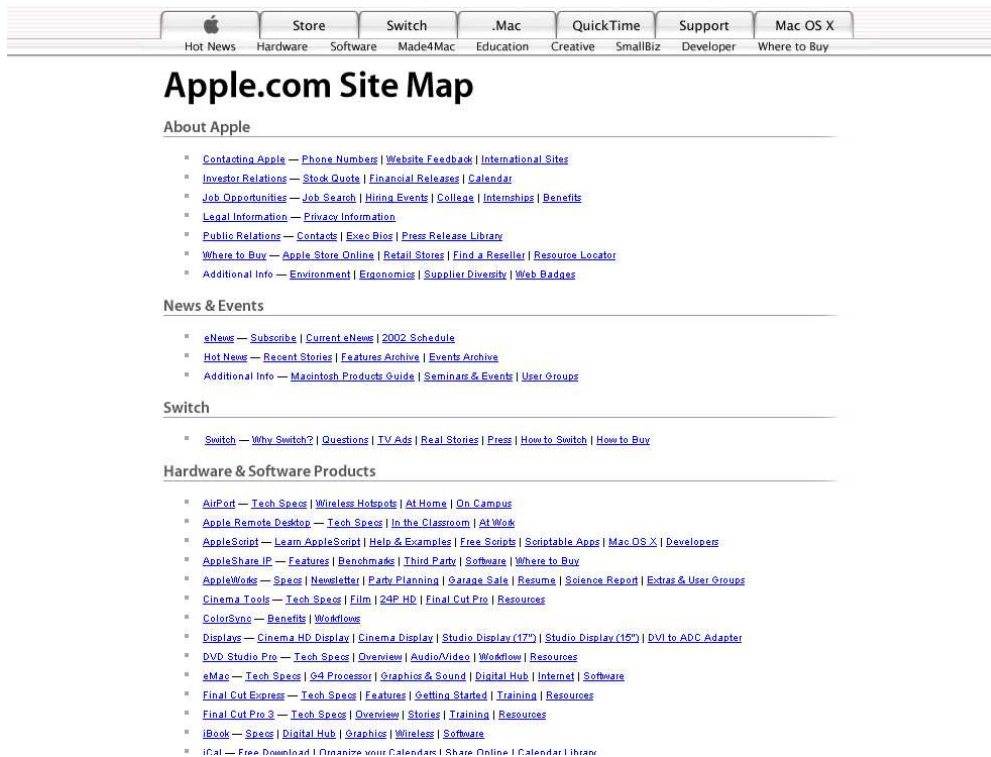
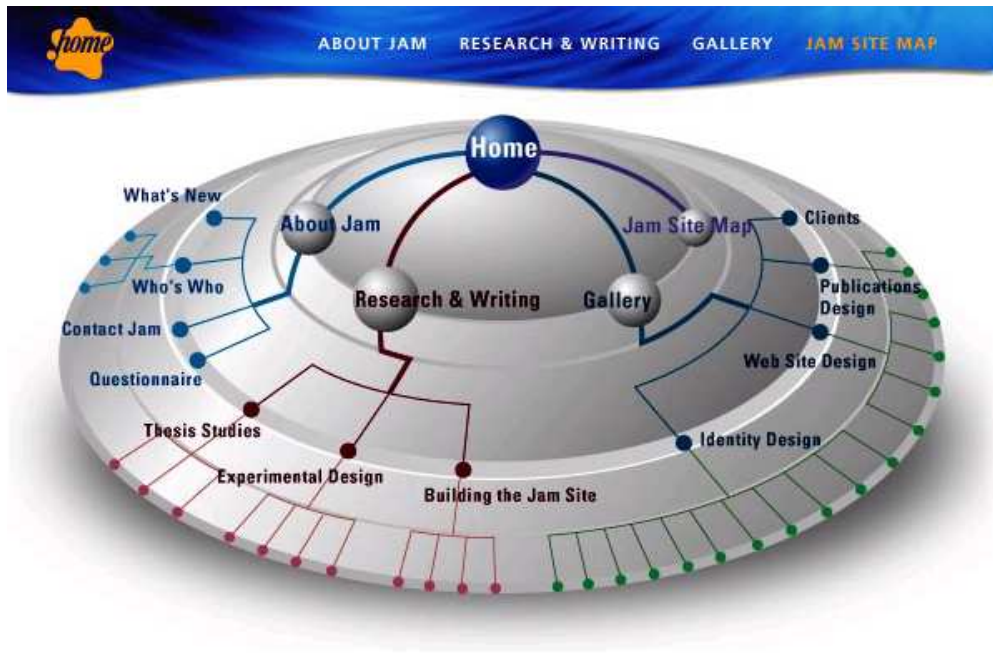


Figure 2.15: Apple's sitemap.



Copyright 2002 Jam Design Inc. 617.247.9465

Figure 2.16: An Unidentified Flying Sitemap!





Figure 2.17: Sitemap showing information from various weather observation areas.

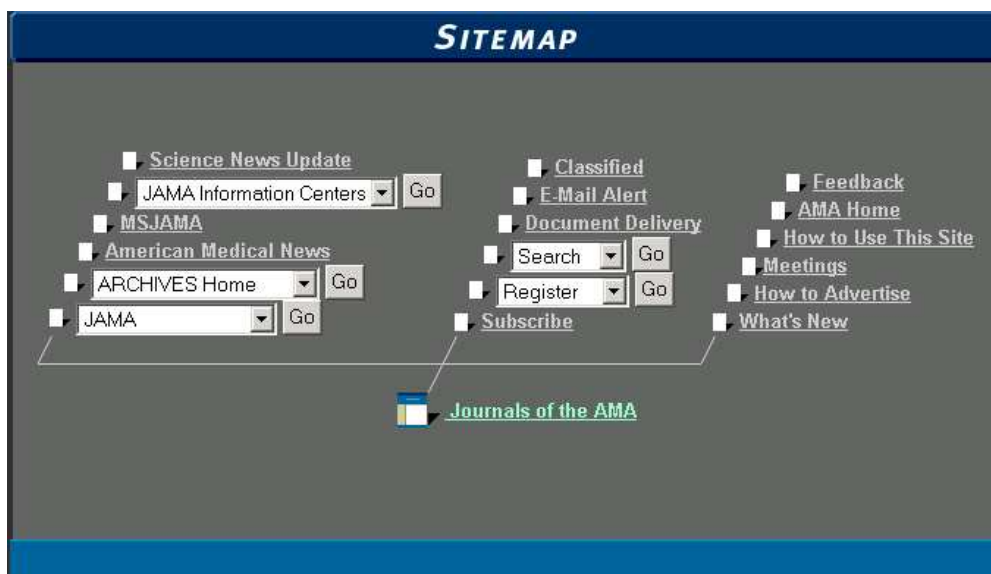


Figure 2.18: Sitemap from the Journals of the AMA. Form elements make navigation more difficult.

### 2.9.3 Trees and Graphs

Jam Design’s site map<sup>19</sup> (figure 2.16) displays the pages within the site in a tree stretched out over the shell of the spaceship. Trees and hierarchies are useful systems for organizing information and their construction can be automated. Chen et al. presented the Cha-Cha<sup>20</sup> concept as an attempt to show search results in context using a tree shaped list of results (Chen, Hearst, Hong, and Lin 1999). The user interface is shown in figure 2.19. Each leaf node represents a relevant page in the Web site and the tree shows the shortest paths back to a starting point, such as the home page.

An alternative approach also uses the idea of a tree combined with the idea of a *fish-eye* view. The “focus+context” technique (Lamping, Roa, and Pirolli 1995) is demonstrated by Inxight’s hyperbolic *StarTree* shown in figure 2.20. Nodes can be clicked and dragged towards the centre to bring them into focus. When this happens new nodes appear at the edges of the view.

The *VisIT* interface developed at UIUC shows each page as a rectangle in a grid. Links between the rectangles indicate hyperlinks between the pages. The program acts as a meta-search engine – requesting information from search engines, downloading each page in the returned results, parsing the contents for link information and constructing the graph shown in figure 2.21. For this reason, it is more bandwidth intensive than many search applications which rely on information from a single server.

Natto is a kind of Japanese food made of soybeans. As it is very sticky, if a bean is picked up, those nearby follow it, these cause more beans to follow, and so on. Figure 2.22 shows

<sup>19</sup> [http://www.jamdesign.com/html/jam\\_site\\_map.html](http://www.jamdesign.com/html/jam_site_map.html)

<sup>20</sup> <http://cha-cha.berkeley.edu/>

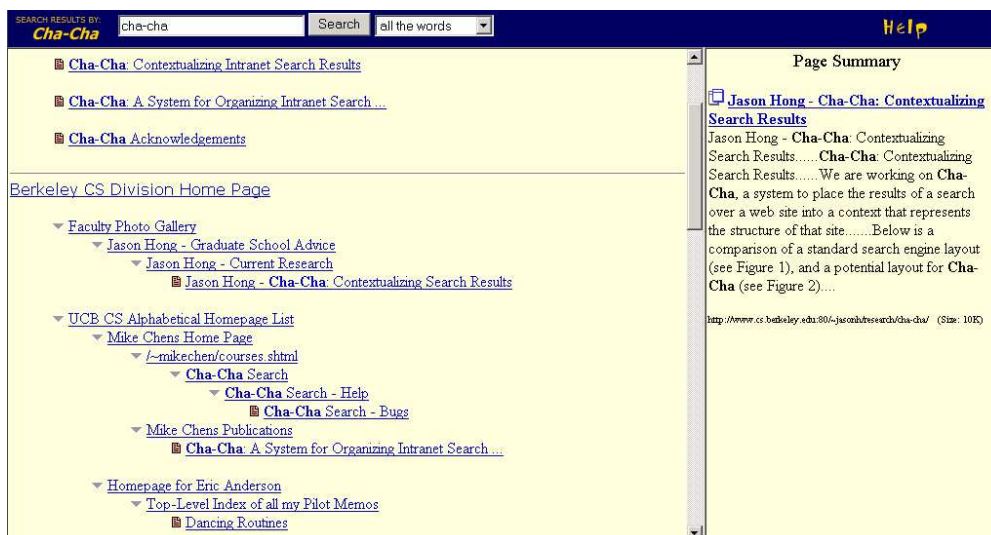


Figure 2.19: The Cha-Cha interface developed at Berkeley.

### Site Map

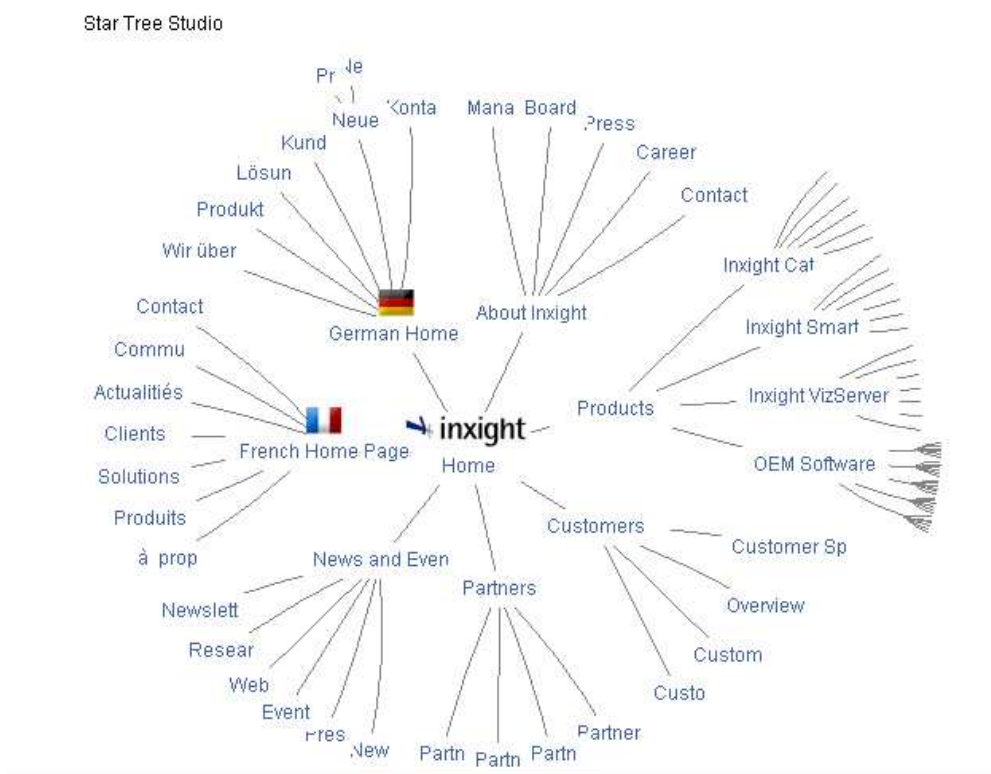


Figure 2.20: The StarTree interface developed by Xerox and available from Inxight.

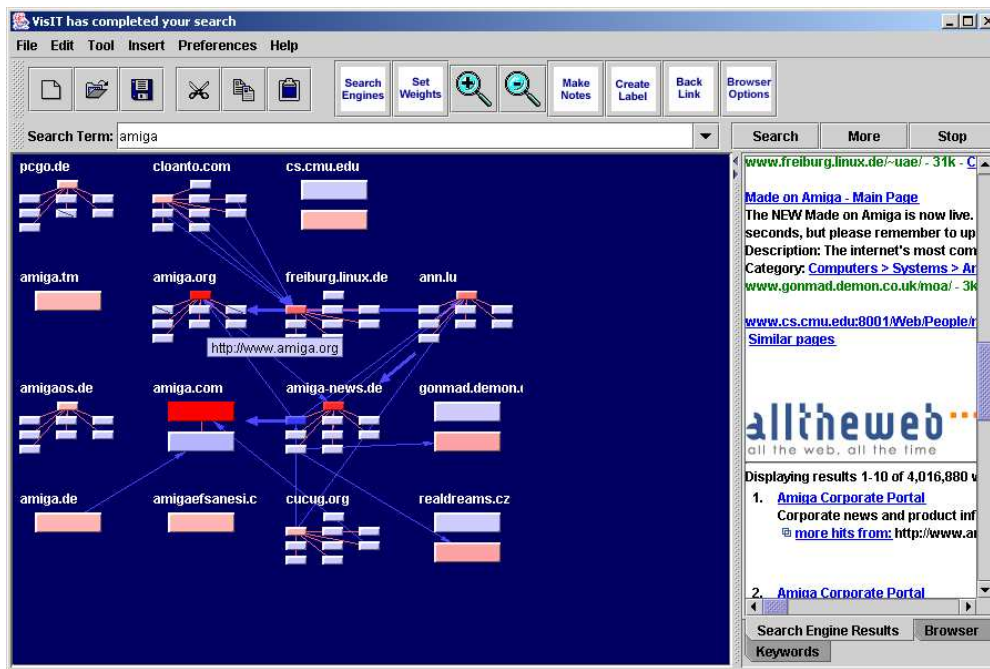


Figure 2.21: The VisIT interface developed by UIUC.

Nattoview – a 3D graph visualization tool based on the metaphor of natto, in which nodes in a Web graph can be dragged and moved like beans (Shiozawa, Nishiyama, and Matsushita 2001).

Cat-a-cone (Hearst and Karadi 1997) presents pages on-screen in a 3D environment. Groups of categories are presented in a barrel-shaped list which can change and revolve whenever a category or page is selected. The categories are stored in a complex hierarchy. The combination of circular lists and tree levels gives rise to the conical structure. The major problem with interfaces such as these is that too much information may be obscured at any one time.

Kartoo is a meta-search engine which presents its results in an interactive graph. Figure 2.23 shows the novel, flash-based interface. Each site in the chosen results is associated with an annotated “ball” connected via “topic” descriptions or keywords. Whenever the mouse pointer is moved over one of the balls, the description of the corresponding site is displayed on the left hand side. When the mouse pointer moves over a topic, the user is presented with plus and minus symbols which alter the query and regenerate the map.

Two other systems which employ graphs for personal use are Internet Cartographer and Mapuccino. Inventix Software’s Internet Cartographer is a personal browsing assistant that uses a graph model to structure the display of sites previously visited by the user. Mapuccino is a program which allows users to create small graphs based on web site content and structure.

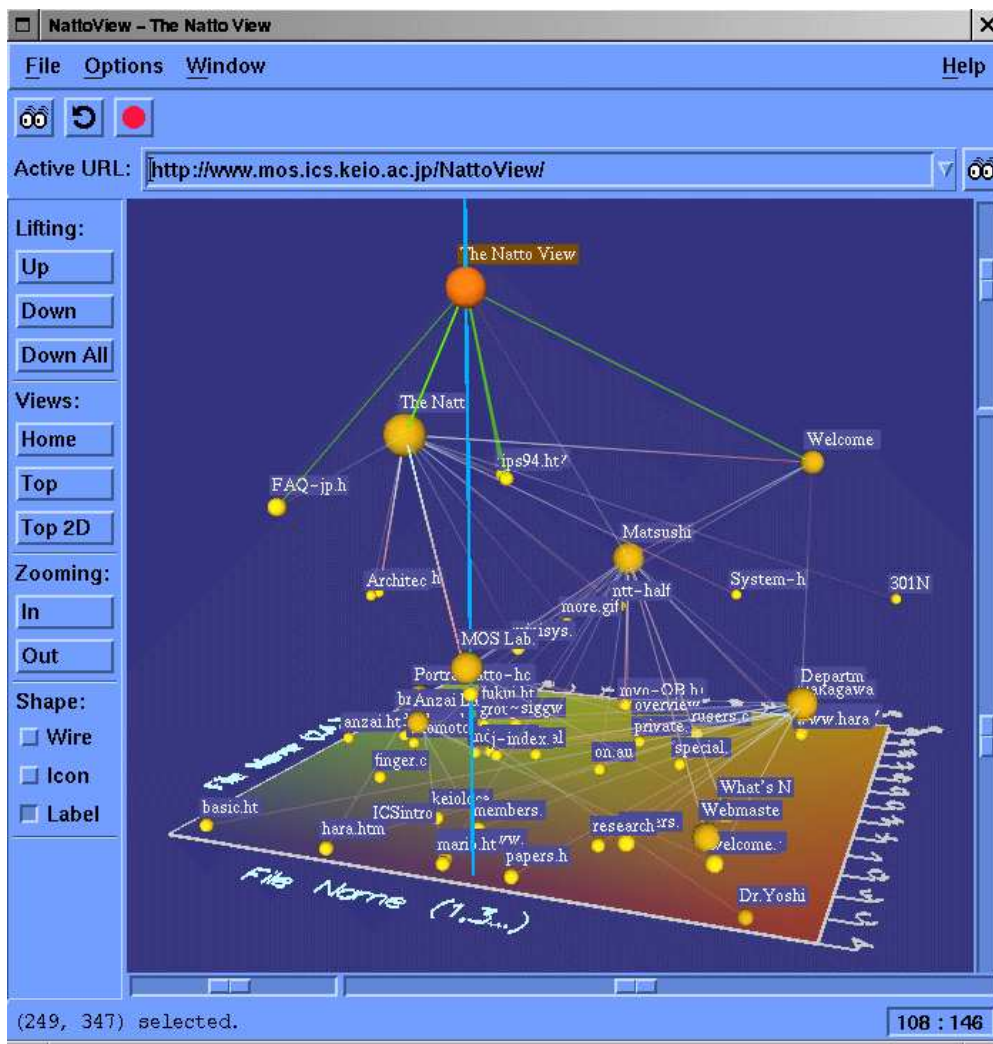


Figure 2.22: The Nattoview interface.

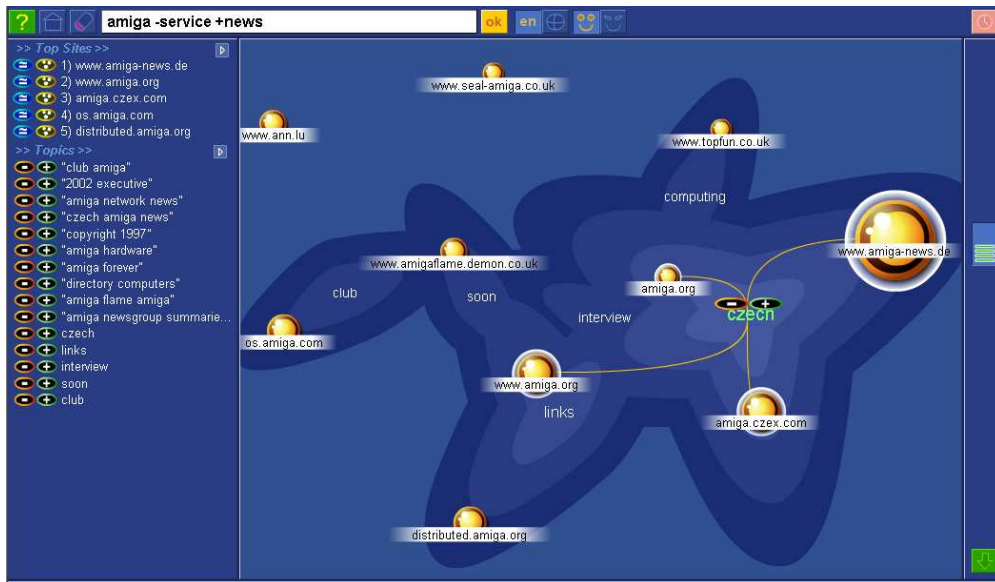


Figure 2.23: The Kartoo interface.

## 2.10 Trail Recording

A trail is a sequence of pages, connected by links. The first use of the term trail in this context occurred in Bush 1945, but the concept has often been used since. When referring to the paths that users follow, the terms “navigation path”, “click-path” or “click-trail” are common. The trails that users follow can be extracted using data mining techniques (Borges 2000).

The idea of presenting optimal trails to the user as a means of solving the navigation problem is also not new. The terms “navigation path”, “chain” or “guided tour” are sometimes used in this context.

Engelbart’s NLS, discussed in section 2.2, was the first system to allow users to construct arbitrary trails. Engelbart’s demo showed how the hierarchical structure of his shopping list example could be moulded to create a trail for his journey home. Textnet supported linear paths using a similar structure (Trigg and Weiser 1986).

Xanadu defines paths in the form of “suggested-threading links”. In the Xanadu model all documents and links are referenced by *tumblers* – a complex tree-like numbering scheme (Nelson 1993). Links are formed with four tumbler references – to the link’s “home” document, the source and destination and an arbitrary type. As all links in the *docuverse* can be referenced using the same tumbler scheme, a chain can be created by joining links together. Nelson defines suggested-threading links as “chainable links which propose a pathway through a corpus of material”.

Hammond and Allinson’s hypertext system for students made use of maps, keyword search and guided tours. The tours also supported sub-tours called “excursions” after their “travel holiday” metaphor (Hammond and Allinson 1988). When questioned on their usage of the system, 91% of students made use of the guided tour facility as opposed to 74% who used the maps and 79% who searched with keywords. Guided tours were the most popular method for studying unfamiliar material and joint favourite for revising known material. The choice of a guided tour mechanism has been given as one of the most influential factors in a user’s ability to navigate hypertext effectively (Hammond and Allinson 1989b; Nielsen 1989).

The NoteCards system relied heavily on the manipulation of various types of card – each of which were handled as first-class objects. Guided tours were supported through the use of *tabletop* cards which captured the session state. Trigg defines the tabletop as “a snapshot which records the list of cards, the shapes of their windows, their positions on the screen, the scrolled locations (vertically and possibly horizontally) of the windows’ contents, and the order in which to open the windows so that the original (possibly) overlapping arrangement can be preserved” (Trigg 1988). Guided tours were also defined as cards, each of which contained several tabletops in a path or network of paths. This network could be traversed by the system, with the user being prompted whenever multiple destinations were possible.

Stotts and Furata 1989 describes a model for hypertext based on Petri Nets. This model allows the possibility for authors to “specify the creation and deletion of multiple concurrent browsing paths” which can be synchronized so that “concurrently displayed information elements do not become unrelated to each other”.

Zellweger 1989 presents a hypertext system, called “Scripted Documents” which uses trails or

paths as the primary linking mechanism. It allows the user to program *scripts* which describe the path. Each script consists of a set of documents, a set of *script entries*, associating locations with actions and a *path specification*, in the form of a Cedar (Swinehart, ZellWeger, Beach, and Hagmann 1986) program.

Sillitoe et al. proposed a system for manipulating trails, complete with forks and subtrails (Sillitoe, Rossiter, and Heather 1990). They discussed a database backed scheme for storing and retrieving the information. This allows users to record trails and play them back.

Furuta et al. developed a system for authoring, modifying and re-using *Walden's paths* – guided tours on the Web, which could be used in a teaching environment (Furuta, Shipman III, Marshall, Brenner, and Hsieh 1997).



## 2.11 Trail Finding

As has been shown, the concept of presenting trails is well established in the hypertext community, as is the idea of using the computer to extract a user's trails using data mining techniques. The logical extension of this is to allow the computer to predict useful trails and present these to the user. However, there have been few attempts to apply this principle.

WebWatcher (Joachims, Mitchell, and Freitag 1995) advises users on navigation possibilities by highlighting links as they browse. This forms a trail over time, but the link-at-a-time approach does not allow the user to see the context initially. Joachims stated that "in many cases only a sequence of pages and the knowledge about how they relate to each other can satisfy the user's information need" (Joachims, Freitag, and Mitchell 1997). The *navigation engine* extends this to compute and show complete sequences in advance.

The Cha-Cha system (Chen, Hearst, Hong, and Lin 1999) presents results in a similar manner to the NavSearch interface described in chapter 6. These results are shown in context, but the scoring is only conducted at the page level, the trails leading to the page are chosen as the shortest paths, not those with informative content.

Mizuuchi and Tajima 1999 describes a system for finding paths to specific nodes, based upon link-analysis and the directory structures implied by URLs. These paths are assumed to be the paths most likely to be followed by the user and therefore contain keywords not present in the pages themselves. The content of pages on these paths can be used to augment indexes. Yet again, these paths are not judged on the total information content, nor are they suggested to users.

Bernstein describes a "shallow apprentice" for link suggestion. The system "may also construct hypertext paths or tours" by getting the author to "choose an interesting starting point and ask the apprentice to construct a path through related material" (Bernstein 1990). The tours are constructed via a best-first page finding scheme using document similarity measures. The intended audience is hypertext authors who can verify and modify the tours. In this example, the tours are based upon IR metrics, but ignore pre-existing link structure.

Guinan and Smeaton 1992 describes a system for computing query-specific guided tours, in a similar way to that outlined by Bernstein. The algorithm to compute the tours is based on a simple ordering using link types and *tf.idf*. All relevant nodes with *tf.idf* values above a certain fraction of the maximum value are incorporated. However, as with Bernstein's system, the algorithm doesn't take advantage of user-defined links. It also relies heavily on user-defined or system-defined link types, neither of which are applicable to the Web.

Two algorithms which appear more promising are the *Best Trail* algorithm and a more complex reinforcement algorithm for constructing *Web Probabilistic Views* based on a sample-credit-update cycle (Zin and Levene 1999). The Best Trail algorithm was enhanced and used as the basis for this work. The current version is discussed in detail in chapter 3.

An outline sketch of the second algorithm, which generates trails in the form of *Web Probabilistic Views*, is shown in figure 2.24. The views are weighted subgraphs in which the good trails can be found. The algorithm represents the views in the form of a Hypertext Probabilistic Grammar (HPG),  $G = (V_N, V_T, R, P, S)$ , in which  $V_N$  and  $V_T$  represent sets of state (node) and transition symbols,  $R$  represents a set of transition rules,  $P$  assigns probabilities

to these rules and  $S$  defines a start state. The *sample* function, updates a sample of trails,  $T$ . A vector of credit scores,  $C$ , are assigned in proportion to the relevance of these trails. The probabilities controlling the grammar are then updated. The rate at which this rather complex process operates is controlled by a parameter,  $0 \leq \alpha < 1$ . It is the cost of this lengthy computation which makes the Web View algorithm less suitable for subsequent development than the Best Trail algorithm.

**Algorithm 1** (`web_view`( $\langle V_N, V_T, R, S \rangle, n, Q, \alpha$ ))

1. **begin**
2.    $p \leftarrow \textit{initialise}(R)$ ;
3.   **repeat**
4.      $G \leftarrow \langle V_N, V_T, R, P, S \rangle$ ;
5.      $T \leftarrow \textit{sample}(G, n)$ ;
6.      $C \leftarrow \textit{credit}(T, Q)$ ;
7.      $P' \leftarrow \textit{normalize}(C)$ ;
8.      $P \leftarrow \textit{update}(P, P', \alpha)$ ;
9.   **until** the expected trail relevance converges to a fixpoint;
10. **return**  $\langle V_N, V_T, R, P, S \rangle$ ;
11. **end.**

Figure 2.24: Zin and Levene's generic Web View algorithm

The Web view concept is grounded in grammar and automata theory. Finite automata are used as models of computer power, in the study of languages and as the basis for regular expression matchers (Hopcroft and Ullman 1979). Regular expressions play an important part in powerful tools such as `grep`, `sed` and `perl` (Friedl 2002; Dougherty and Robbins 1997; Larry Wall 2000).

A Web Graph  $G = (N, E)$  can be modelled as a *hypertext automata*, where the set of nodes,  $N$ , maps onto the set of possible states,  $S$ , and the set of edges,  $E$ , maps onto the set of state transitions,  $T$  (Borges 2000). In such an automata, a trail through the Web Graph is described by a sentence accepted by the automata, and the set of possible trails through the Webgraph maps to the *language* accepted by the automata (the set of all possible sentences accepted by the automata). It is also possible to define a Web View as a subset of this language, and to construct a hypertext grammar describing this language.

A Hypertext Probabilistic Automata (HPA), is the probabilistic formulation of the hypertext automata where any link between two states,  $s_1$  and  $s_2$ , is associated with a probability,  $P(s_1 \rightarrow s_2)$ . In this context, the term *support* denotes the acceptance of trails whose initial probabilities are greater than some value  $\theta$ , *confidence* denotes the acceptance of trails whose total probability is greater than a value  $\delta$  and a *cut-point* denotes the acceptance of trails whose total probabilities are greater than  $\theta \cdot \delta$  (Borges 2000).

The probabilistic automata model maps directly to the Markov chain model where the transitions are represented by a *process* in which the probability of a transition is based solely upon the previous step or steps. This leads directly to the *random surfer* model which has proved popular with researchers and has been used as a model for explaining the HITS and

PageRank metrics (Kleinberg 1998; Page, Brin, Motwani, and Winograd 1998; Henzinger, Heydon, Mitzenmacher, and Najork 1999; Fagin, Karlin, Kleinberg, Raghavan, Rajogopalan, Rubinfeld, Sudan, and Tomkins 2000; Levene and Loizou 2002).

These model allow complex reasoning about hypertext documents. For example, they can be used in analysis to verify the structure of the document or to prove the complexity of operations (Stotts, Furuta, and Ruiz 1992; Levene and Loizou 1999; Moreau and Hall 1998).

## 2.12 Summary

The navigation problem in hypertext – the problem of helping users find their way and stopping them from getting “Lost in Hyperspace” has been described. This is related to, yet distinct from, the problem of resource discovery.

The information retrieval community has provided document retrieval mechanisms based on statistical properties of the corpus and link graph. Models such as the VSM allow complex document structures to be simplified into simple independent terms; measures such as *tf.idf* allow the effective selection of such documents; and practical techniques such as inverted files allow the rapid retrieval of documents according to that selection.

The hypertext community has provided mechanisms for storing, representing and manipulating trails. Similarly, these mechanisms are based upon sound mathematical models using graph theory, Petri nets or automata. They have been widely implemented in many environments, and proven to be popular and successful.

The Web community has contributed techniques for managing the World’s largest repository of knowledge. The Web is composed of sites, which can be difficult to navigate. The site can be made more navigable through the use of maps and cues suggesting relevant links to follow.

In the following chapters many of these techniques will be incorporated into an automated system for constructing and returning trails based on properties of the corpus.

## Part II

# Implementation of a Trail-Based Navigation Engine

## Chapter 3

# The Best Trail Algorithm

Two roads diverged in a yellow wood,  
And sorry I could not travel both  
And be one traveler, long I stood  
And looked down one as far as I could  
To where it bent in the undergrowth;

Then took the other, as just as fair,  
And having perhaps the better claim  
Because it was grassy and wanted wear;  
Though as for that, the passing there  
Had worn them really about the same,

And both that morning equally lay  
In leaves no step had trodden black.  
Oh, I marked the first for another day!  
Yet knowing how way leads on to way  
I doubted if I should ever come back.

I shall be telling this with a sigh  
Somewhere ages and ages hence:  
Two roads diverged in a wood, and I,  
I took the one less traveled by,  
And that has made all the difference.

Frost 1875

### 3.1 Introduction

This chapter describes the implementation of an enhanced version of the Best Trail algorithm (Levene and Zin 2001). The original algorithm was developed for finding trails between linked pages in the Web. Unfortunately, the algorithm had previously been implemented only once and the performance of this implementation was never adequately assessed. The code has been completely re-written and substantial changes have been made to both the algorithm and the auxiliary functions whilst maintaining the overall structure. The principle of combined exploration and convergence operations to manipulate a navigation tree remains but the probability distribution has changed to improve efficiency. The outer loops are also merged to allow the return of a set number of trails from each starting point. The selection functions, scoring mechanisms and trail ordering functions were never adequately defined and have all been changed. Finally, post-computation filters and sorting methods have been developed to remove redundant information and improve the quality of results. In order to avoid continual references to the previous version, the revised algorithm will be presented in full as described in Wheeldon and Levene 2003.

The Web is a massive global hypertext system in which documents or pages can be found on almost every subject imaginable. A Web *site* represents a collection of pages with some common element. The common factor may be the subject or topic of the pages, a common author or a shared organization or community. Web sites are often associated with physical servers, but this may not always be the case. A large Web site may be spread across servers (e.g. `sun.com`, `ibm.com`, etc.) and a single server or cluster may host many individual Web sites (e.g. `geocities.com`). The vast majority of links on the Web are between pages within a common Web site.

The objective of the Best Trail algorithm is to quickly and efficiently identify memex-like trails in Web-like graphs. A trail in this context consists of a series of linked pages. The algorithm works by performing a probabilistic expansion of a set of *navigation trees* from chosen starting points. The original intention was to apply this to large crawls of the Web. However, the huge numbers of servers and required levels of support have unfortunately removed full-scale Web search from the realms of academic research. The algorithm is better applied to the realm of *Web sites* and this application will be explored in chapter 6.

A Web site is modelled as a hypertext system  $H$  having two components – a directed graph  $G = (N, E)$ , having finite sets of nodes and arcs  $N$  and  $E$ , respectively, and a scoring function  $\mu$  which is a function from  $N$  to the set of non-negative real numbers. The directed graph  $G$  defines the Web site topology and is referred to as the *Web Graph*; the nodes in  $N$  represent the Web *pages* and the arcs in  $E$  represent hyperlinks (or simply *links*) between *anchor* and *destination* nodes. The terms node, page and URL are used interchangeably. The score,  $\mu(m)$  of a Web page  $m \in N$ , is interpreted as a measure of how relevant  $m$  is with respect to a given query, where the query is viewed as the goal of the navigation session. In this interpretation  $\mu$  can be viewed as the scoring function of a search engine with respect to a given query. Another interpretation of  $\mu$  is that it denotes the relevance to the user. In this sense the hypertext system would be personalised and  $\mu$  would be different for distinct users. In both interpretations of  $\mu$ , the user initiating a navigation session would like to maximise the relevance (or suitability) of the trail to the query. The Best Trail algorithm attempts to find relevant and useful trails by computing trails which score highly according to some

function of these page scores.

Figure 3.1 shows an example Web graph, representing a complete crawl of the GraphViz Web site<sup>1</sup>. The numbers denote unique identifiers assigned to all URLs. For example, the root of the site, / , is assigned an ID of 4 and is shown left-of-centre. The gaps in the sequence are due to URLs, referenced by the pages in the Web site, of pages elsewhere on the Web. Two hubs, `download.html` and `refs.html`, link to the program executables and perl scripts and to the papers and technical manuals respectively. The numbers in parentheses denote relevance scores for the query “dotty”, which is the one of the main GraphViz programs – the one used to edit directed graphs. This Web graph will be used as a running example throughout this chapter, the remainder of which is organized as follows:

**Section 3.2** discusses related algorithms for graph traversal and finding different kinds of paths, and shows why these algorithms are not optimal for the task of finding relevant trails.

**Section 3.3** describes the Best Trail algorithm. The algorithm is a probabilistic best-first traversal algorithm with two stages.

**Section 3.4** describes auxillary functions used by the algorithm. Functions are required to manipulate the navigation trees – selecting tips to expand and adding the new tips to the tree.

**Section 3.5** describes scoring metrics used as objective functions. Two functions, referred to as *sum distinct* and *weighted sum*, are defined in terms of the scores of individual pages.

**Section 3.6** describes heuristics for filtering redundant information from navigation trails.

**Section 3.7** describes the latest implementation of the Best Trail algorithm. All tip selection is done via a table-based binary tree.

**Section 3.8** describes the computational complexity of this implementation. The complexity is specified entirely by the parameters of the algorithm and the characteristics of the Web graph.

**Section 3.9** shows the results of experiments into the behaviour and performance of the algorithm.

**Section 3.10** gives concluding remarks and directions for future research.

---

<sup>1</sup> <http://www.research.att.com/sw/tools/graphviz/>



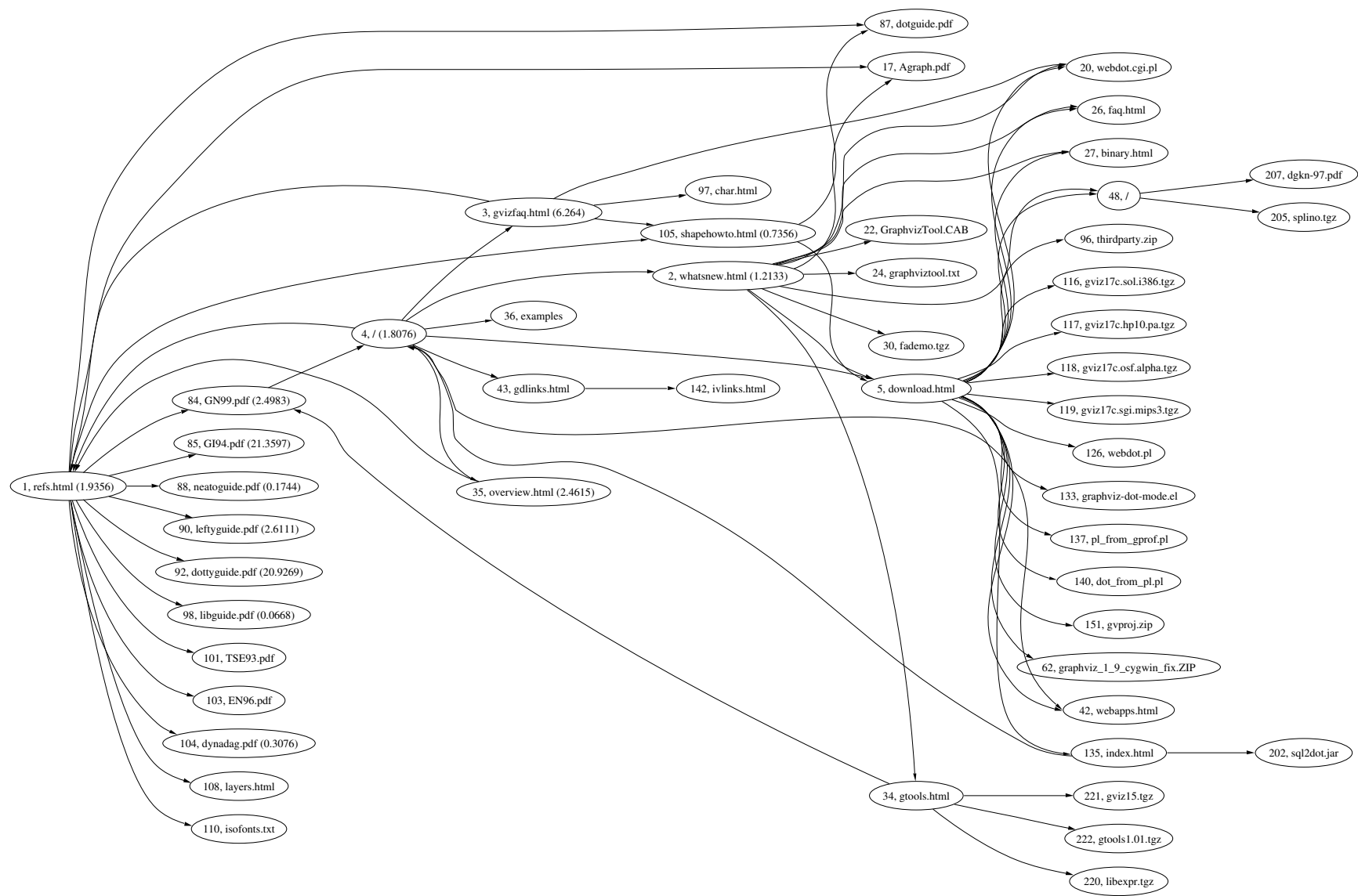


Figure 3.1: An example Web topology.

## 3.2 Graph Traversal and Path Finding

Many graph traversal and path-finding algorithms have been developed over the last 50 years and it is not unreasonable to question the development of a new one. The effects of a few of them will be considered:

**Depth-first** search (DFS) is unsuitable for trail finding as it may tend towards “black-holes” from which there is no escape. It is considered unsuitable for crawling for similar reasons.

**Breadth-first** search (BFS) is non-viable for anything other than very short trails, due to the exponential growth of the tree.

**Best-first** search is possible but will struggle in situations where the best pages are separated by content which is less relevant – exactly the situations where automated navigation is most needed! It should also be noted that a best-first search is a degenerate case of the Best Trail algorithm. Even a simple implementation of a best-first algorithm will require a priority queuing system similar to that used by the Best Trail, so there is little saving in complexity.

**Dijkstra’s** shortest path algorithm is designed to find exact matches for short paths. The algorithm is of too high a complexity,  $O(N^3)$ , to be suitable for real time use in Web graphs. In addition the shortest trail may not always be the best and an algorithm such as Dijkstra’s is difficult to adapt to arbitrary functions.

**Elastic Net** methods and similar algorithms (R.Durbin and Willshaw 1987) are useful for finding close-to-optimal solutions to the Travelling Salesman Problem (TSP). However, these algorithms often make simplifying assumptions concerning the instance of TSP. Some common assumptions which are that the graph represents a real-world terrain, that the distance from point  $a$  to point  $b$  is always less than the sum of the distances between from points  $a$  and  $c$  and point  $c$  and  $b$  for any  $a$ ,  $b$  and  $c$  or that the distance from  $a$  to  $b$  is equal to the distance from  $b$  to  $a$ . These assumptions do not always hold in the Web environment.

**Ant Colony** optimization is another approach that has been used effectively for computing solutions to TSP (Dorigo, Maniezzo, and Colorni 1996). Each “ant” is an agent which uses a greedy heuristic to follow a trail based upon the weight of links and the presence of a “pheromone”. This pheromone is laid by ants following a path, based upon the length of the final result. Our own experiments have provided anecdotal evidence that the Best Trail algorithm out-performs the ant colony optimization approach for Web site trail finding, although the ant colony system appears to out-perform the Best Trail in finding solutions to TSP.

**Fractal Traversal** is a technique designed for enumerating values in cryptographic systems (Jakobsson 2002). The annotated, acyclic graphs and other data structures, such as Merkel trees, over which fractal traversal has proved successful do not easily map to Web graphs and therefore the technique is somewhat unsuitable.

What is required is an algorithm that can quickly return solutions, yet will be able to find relevant documents beyond less relevant ones. The Best Trail algorithm performs well in this regard.

### 3.3 The Best Trail Algorithm

The pseudo-code of the algorithm is shown as algorithm 2 (figure 3.2). It takes, as input, a set of starting URLs,  $S$ , and a parameter,  $M \geq 1$ , which specifies the number of repetitions of the algorithm for each input URL. When the algorithm terminates it outputs a set of trails,  $B$ . For each URL in  $S$ , there will be  $M$  trails in  $B$ . Each trail is the highest ranking trail contained within the *navigation tree* expanded from a single starting node. Manipulating this set has a filtering effect on the set of starting points, reducing the rank of nodes which are isolated from other relevant documents and from which navigation is problematic. Returning trails from separate trees also has the effect of removing highly similar trails before further filtering is required.

**Algorithm 2 (Best\_Trail( $S, M$ ))**

```

1.  begin
2.    foreach  $u \in S$ 
3.      for  $i = 0$  to  $M$  do
4.         $D \leftarrow \{u\}$ ;
5.        for  $j = 0$  to  $I_{explore}$  do
6.           $t \leftarrow select(D)$ ;
7.           $D \leftarrow expand(D, t)$ ;
8.        end for
9.        for  $j = 0$  to  $I_{converge}$  do
10.          $t \leftarrow select(D, df, j)$ ;
11.          $D \leftarrow expand(D, t)$ ;
12.        end for
13.         $B \leftarrow B \cup \{best(D)\}$ 
14.      end for
15.    end foreach
16.  return  $B$ 
17. end.
```

Figure 3.2: The Best Trail algorithm.

Starting from each node in  $S$ , the algorithm follows links from anchor to destination according to the topology of the Web site, building a set of navigation trees. A navigation tree is a finite subtree of the possibly infinite tree which could be generated by traversing through  $G$ , the root of which is a member of the set of starting points. At each stage of the traversal, one of the *tips* (the leaf nodes of the navigation tree) is chosen for *expansion*. The destination node of each outlink whose source is represented by the chosen tip is assigned a new tip which is added to the navigation tree, along with a computed trail score. Previously visited nodes in the Web graph will result in distinct nodes in the navigation tree, with identical page scores but different trail scores.

Figure 3.3 shows an example navigation tree based on the Web topology shown in figure 3.1. Each node is annotated with a unique tip number, and the URLid and URL shown previously. Red ellipses denote candidate tips for expansion. The tip numbers are assigned in sequence

during the iteration of the algorithm. In this example, the tips numbered 1, 3, 9, 5 and 24 were expanded.

The algorithm has a main outer loop which computes the best trail for each URL. The second loop recomputes the Best Trail  $M$  times. As the tip selection is probabilistic, the same results may not occur for each navigation tree. The two innermost loops comprise the exploration and convergence stages of the algorithm, both of which expand the navigation tree from which the Best Trail is selected by the *best()* function. The number of iterations in the exploration phase is set by  $I_{explore}$ , whilst the number of iterations in the convergence phase is set by  $I_{converge}$ .

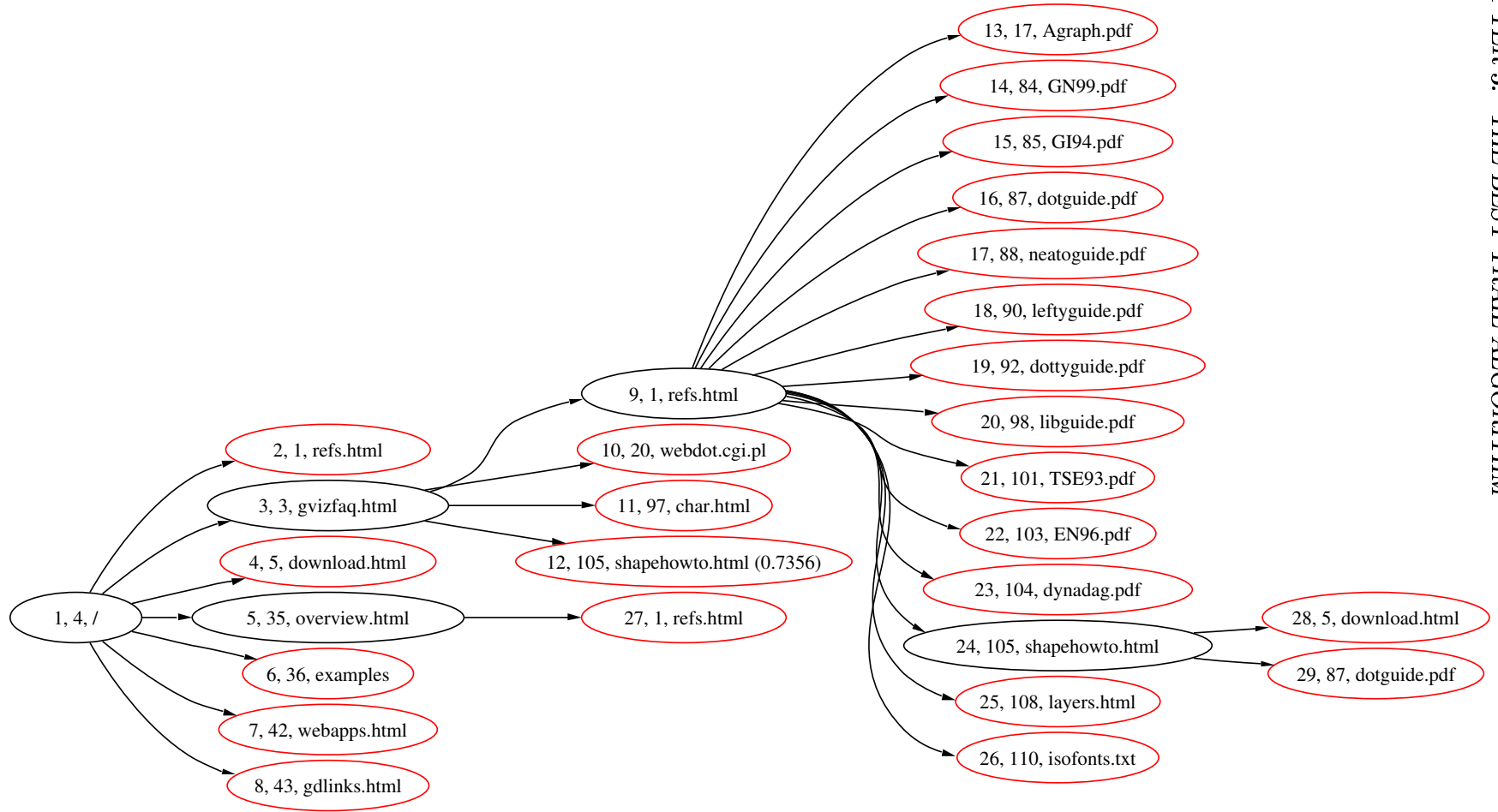


Figure 3.3: An example navigation tree.

### 3.4 Auxillary Functions

Definitions of the auxiliary functions and parameters used in the Best Trail algorithm will now be expanded on:

$I_{explore}$  is the number of iterations during the exploration stage of the algorithm.  $I_{explore} \geq 0$

$I_{converge}$  is the number of iterations during the convergence stage of the algorithm.  $I_{converge} \geq 0$

$df$  is a *discrimination factor* – a parameter, in the range  $[0, 1]$ , which allows us to discriminate between “good” trails and “bad” trails by reducing the influence of trails which perform badly. Thus during the convergence stage “better” trails get assigned exponentially higher probability. Setting  $df$  equal to one would imply a uniform random selection, whilst as  $df$  tends towards zero, the behaviour of the algorithm tends towards that of a best-first approach. The degenerate case of the Best Trail algorithm where  $df = 0$ ,  $I_{explore} = 0$  and  $I_{converge} > 0$  is equivalent to the best-first algorithm.

$\rho(t_k)$  represents the score of trail  $t_k$ . This is defined as a function of the score of the constituent nodes where the scores for a node  $n$  is given by  $\mu(n)$ . Possible functions are discussed in section 3.5.

$\tau(t)$  denotes the *rank* of a tip,  $t$ , (or of the trail leading to it). This is determined by the tip’s position within the ordered set of candidate tips. The position of  $t$  is determined by comparing nodes based upon

1. The number of query terms matched by the trail ending at  $t$ .
2. The maximum number of query terms matched by any single page in the trail.
3. The trail score,  $\rho(t_k)$ .

In the current implementation, no two trails may share the same rank even if they are equally matched with regard to each of these measures. This allows the *select()* operation to be performed in  $O(\log N)$  time, as described in section 3.8.

It has been argued that the number of keywords in a query that are matched by a document should take precedence over other scoring mechanisms, and that the terms for a query may be spread across several pages (Anh and Moffat 2002; Li, Candan, Vu, and Agrawal 2001; Joachims, Freitag, and Mitchell 1997). Ranking the trails first upon the number of keywords that are matched, incorporates both of these ideas and improves relevance.

*expand*( $D_i, t$ ) returns a navigation tree resulting from the expansion of  $D_i$  with tip node  $t$ . That is, the navigation tree resulting from adding new tip nodes for each outlink from the node in the Web graph which is represented by  $t$ .

*select*( $D_i, \alpha, j$ ) is a function which selects a candidate tip to expand.  $\alpha$  is either 1 or  $df$  and  $j$  denotes either an exploration or convergence step and  $D_i$  is the navigation tree. The tip is chosen at random according to the probability distribution function  $P$ . During the exploration phase, the *select()* function selects a tip to expand where the probability of a tip  $t$  being selected is given by

$$P(D_i, t) = \frac{\rho(t)}{\sum_{k=1}^n \rho(t_k)}$$

making the probability of any node being selected directly proportional to its score. During the convergence phase, the probability of a node  $t$  being selected is dependant only on its relative rank,  $\tau(t)$ , in the ordered set of candidate tips, and is given by

$$P(D_i, t, df, j) = \frac{df^{\tau(t)j}}{\sum_{k=1}^n df^{\tau(t_k)j}}$$

where  $j$  is the number of completed convergence iterations. These distributions allow the *select()* function to be implemented efficiently.

*best(D<sub>i</sub>)* returns the highest ranked trail induced by the set of tip nodes of  $D_i$ .



### 3.5 Scoring Trails

The relevance or *score* of a trail,  $T = U_1, U_2, \dots, U_n$ , is defined by a function,  $\rho$ , of the scores of the individual Web pages of the trail. A function is needed which will encourage non-trivial trails whilst discouraging redundant nodes. The following functions perform well in this regard:

**Discounted Sum** The sum of the discounted scores of the URLs in the trail, where the discounted score of  $U_i$ , the URL in the  $i$ th position in the trail, is the score of  $U_i$  with respect to the query multiplied by  $\gamma$  raised to the power of  $i - 1$ , where  $0 < \gamma < 1$  is the discount factor. This encourages trails in which the most relevant content appears early and fits well with observations about the length of trails which users naturally follow (Silverstein, Henzinger, Marais, and Moricz 1999). The discounted score of  $T$  is given by

$$\rho(T) = \text{discount}(T) = \sum_{i=1}^n \mu(U_i) \gamma^{i-1}.$$

**Weighted Sum** of discounted scores, where the additional weighting is achieved by discounting each URL according to its previous number of occurrences within the trail. The weighted score of  $T$  is given by

$$\rho(T) = \text{weighted}(T) = \sum_{i=1}^n \mu(U_i) \gamma^{i-1} \delta^{c(i)}$$

where  $c(i) = |\{U_j | j < i \wedge U_j = U_i\}|$  is the number of occurrences of an equivalent node in an earlier position in the trail and where  $0 < \delta < 1$  is a second discounting factor, typically with  $\delta < \gamma$ . This second discounting function reduces the importance of nodes with equal content. Although  $i = j$  implies  $U_i = U_j$ ,  $U_i = U_j$  does not imply  $i = j$ . Two distinct nodes may be considered equal if they have equal content, determined in advance using checksums and by comparing likely candidates. This definition of node equality can easily be extended to refer to near-duplicate documents (Broder 2000; Shivakumar and Garcia-Molina 1999). Using the weighted sum function with parameters  $\gamma = 0.75, \delta = 0.05$ , the score of the trail to tip 15 in figure 3.3 can be calculated as 16.6.

**Sum Distinct** The *sum* of the scores of the *distinct* URLs in the trail divided by the number of pages in the trail plus some constant (e.g. 1). This function penalises the trail when a URL is visited more than once. It also penalises trivial singleton trails and encourages trails where every node makes a significant contribution to the score. Removing the constant factor leads the objective function to return a maximal score in the case of a singleton node where that node is the highest scoring page in the corpus. Scoring functions such as the average score or maximum score of a node on a trail also suffer from this problem. Using the sum distinct function with a constant factor of 1, the score of the trail to tip 15 in figure 3.3 is 6.27. Formally, this function is given as

$$\rho(T) = \text{sumDistinct}(T) = \frac{1}{n + \epsilon} \sum_{i=1}^n \text{first}(i)$$

where  $\epsilon$  is a constant and  $first(i)$  returns 1 if and only if  $c(i) = 0$  and returns 0 otherwise.

Figure 3.4 shows how the trails in the navigation tree would be scored after two expansion (of tips 1 and 3) using the parameters  $\epsilon = 1$ ,  $\gamma = 0.75$  and  $\delta = 0.05$ . The examples shown in this paper are constructed by computing two trails from each starting point – one scored using the *sum distinct* metric and one using the *weighted sum*. High scores are associated with relatively short trails when using sum distinct. This forces the most relevant pages to the forefront of the display. The weighted sum encourages longer trails. Merging trails with common roots gives a good ordering to the display, as can be seen in figure 3.5, which shows the interface discussed in section 6.2.

Tip	Weighted Sum	Sum Distinct ( $\div n$ )
1	1.8076	0.9038
2	3.2593	1.2477
3	6.5056	2.6905
4	1.8076	0.6025
5	3.6534	1.4230
6	1.8076	0.6025
7	1.8076	0.6025
8	1.8076	0.6025
9	7.5940	2.5018
10	6.5056	2.0179
11	6.5056	2.0179
12	6.9194	2.2018

Figure 3.4: Table showing trail scores using Weighted Sum and Sum Distinct.

The screenshot shows a web browser window with the following elements:

- NavigationZone+ Header:** Includes search options: `NavSearch | TrailSearch | VisualSearch` and a search input field containing `dotty`.
- Search Results:**
  - Found 34 pages in 11 hits. Search took 0.4 seconds.
  - Trail > **dottyguide.pdf**
  - Graphviz FAQ 20020919
  - Graphviz
  - GI94.pdf
  - Graphviz
  - Graphviz FAQ 20020919
  - Graphviz
  - GI94.pdf
  - Graphviz
  - GN99.pdf
  - Graphviz
  - Graphviz FAQ 20020919
  - Graphviz
  - GI94.pdf
  - Graphviz
  - Graphviz download
  - SG2Dot
  - Graphviz
  - Graphviz FAQ 20020919
  - Graphviz
  - GI94.pdf
  - leftyguide.pdf
  - dynamdag.pdf
- Navigation Zone Sidebar:** Contains sections for `Bookmarks`, `Thumbnail`, and `Signature`.
- Main Content Area:**
  - Abstract:**

*dotty* is a graph editor for the X Window System. It may be run as a standalone editor, or as a front end for applications that use graphs. It can control multiple windows viewing different graphs.

*dotty* is written on top of *dot* and *lefty*. *lefty* is a general-purpose programmable editor for technical pictures. It has an interpretive programming language similar to AWK and C. The user interface and graph editing operations of *dotty* are written as *lefty* functions. Programmer-defined graph operations may be loaded as well. Graph layouts are made by *dot*, which runs as a separate process that communicates with *lefty* through pipes.

The screen dump below shows a snapshot of a typical *dotty* session.
  - DOTTY Screenshot:** A window titled "DOTTY" showing a complex graph layout with various nodes and edges. The nodes include labels like "do layout", "read graph", "load graph", "save graph", "copy view", "close view", "zoom in", "zoom out", "find node", "text view", "quit", "Process", "Print", "Repeat", "DAP", "WatchMail", "NDFS", "NewMail", "NetSer", "Lab-92", "AWK", "TXX", "T77", "P2C", "Lab-1", "Lab-68", "Lab-10", "Lab-11", "Lab-12", "Lab-13", "Lab-14", "Lab-15", "Lab-16", "Lab-17", "Lab-18", "Lab-19", "Lab-20", "Lab-21", "Lab-22", "Lab-23", "Lab-24", "Lab-25", "Lab-26", "Lab-27", "Lab-28", "Lab-29", "Lab-30", "Lab-31", "Lab-32", "Lab-33", "Lab-34", "Lab-35", "Lab-36", "Lab-37", "Lab-38", "Lab-39", "Lab-40", "Lab-41", "Lab-42", "Lab-43", "Lab-44", "Lab-45", "Lab-46", "Lab-47", "Lab-48", "Lab-49", "Lab-50", "Lab-51", "Lab-52", "Lab-53", "Lab-54", "Lab-55", "Lab-56", "Lab-57", "Lab-58", "Lab-59", "Lab-60", "Lab-61", "Lab-62", "Lab-63", "Lab-64", "Lab-65", "Lab-66", "Lab-67", "Lab-68", "Lab-69", "Lab-70", "Lab-71", "Lab-72", "Lab-73", "Lab-74", "Lab-75", "Lab-76", "Lab-77", "Lab-78", "Lab-79", "Lab-80", "Lab-81", "Lab-82", "Lab-83", "Lab-84", "Lab-85", "Lab-86", "Lab-87", "Lab-88", "Lab-89", "Lab-90", "Lab-91", "Lab-92", "Lab-93", "Lab-94", "Lab-95", "Lab-96", "Lab-97", "Lab-98", "Lab-99", "Lab-100".

Figure 3.5: Results for the query “dotty” on the topology shown in figure 3.1.

### 3.6 Sorting and Filtering

The set of trails returned by algorithm 2 (figure 3.2) is unsorted and may contain redundant information. To sort the trails would appear to be trivial – the same rules of sorting by the number of keywords matched and then by the trail score are applied. However, there is more than one mechanism for scoring trails, and trail scores can be computed in different navigation trees using different functions. The resulting trails can be sorted using a set of scoring functions,  $F$ , by specifying that a trail,  $T_1$  should be ranked higher than a trail  $T_2$  if :

$$\sum_{f \in F} \frac{f(T_1)}{f(T_1) + f(T_2)} > \sum_{f \in F} \frac{f(T_2)}{f(T_1) + f(T_2)}$$

Results can be improved by *filtering* – removing redundant trails and redundant sections within trails. To achieve this, a precise definition of a redundant trail is needed. For example, a trail  $T$  might be said to be redundant if all the pages in  $T$  were contained in higher scoring trails, but this definition would ignore the importance of the link structure in showing context. Alternatively, only trails which are strict subtrails of previous trails might be removed, but this would leave too much redundant information. A compromise is reached by saying that a trail  $T_1$  subsumes a trail  $T_2$  if and only if all the pages in  $T_2$  are contained in  $T_1$ . A trail,  $t_1$  is removed from a result set,  $r$  if and only if there exists a trail  $t_2 \in r$  such that  $t_2$  subsumes  $t_1$  and  $\rho(t_2) > \rho(t_1)$ .

Within a trail  $T$ , a page,  $t_i$  is considered to be redundant if and only if the page can be removed whilst still leaving a valid trail through the Web site topology (i.e. if  $t_i$  is the last node of the trail or  $(t_{i-1}, t_{i+1}) \in E$  and the information contained on page  $t$  is either not relevant or contained in a previous page (i.e. if  $\rho(t) = 0$  or  $\exists j t_j = t_i \wedge j < i$ ). This typically removes trivial reorderings and irrelevant content which may appear at the end of trails.

Whilst the proposed definition seems intuitive, it is unclear precisely what the effect of these operations are likely to be on the computed trail relevance scores. Experiments on the UCLCS web site crawl have been performed which show that on average a 22% improvement (increase in average trail score) can be achieved through filtering when scores are computed using the weighted sum. When scores are computed using this sum distinct metric, the average improvement remains high, at 14.8%. Figure 3.6 shows the effectiveness of the filtration in improving the trail scores. The curves show improvement over 26 queries taken from the UCLCS web site logs. They show that in 85% of cases, filtering produced higher scoring trails, regardless of scoring function. Figure 3.7 shows that the improvement achieved with respect to the weighted sum is strongly correlated with the improvement achieved with respect to the sum distinct score. Other experiments have shown that this effect is not restricted to the UCLCS web site, but appears to be a general rule.

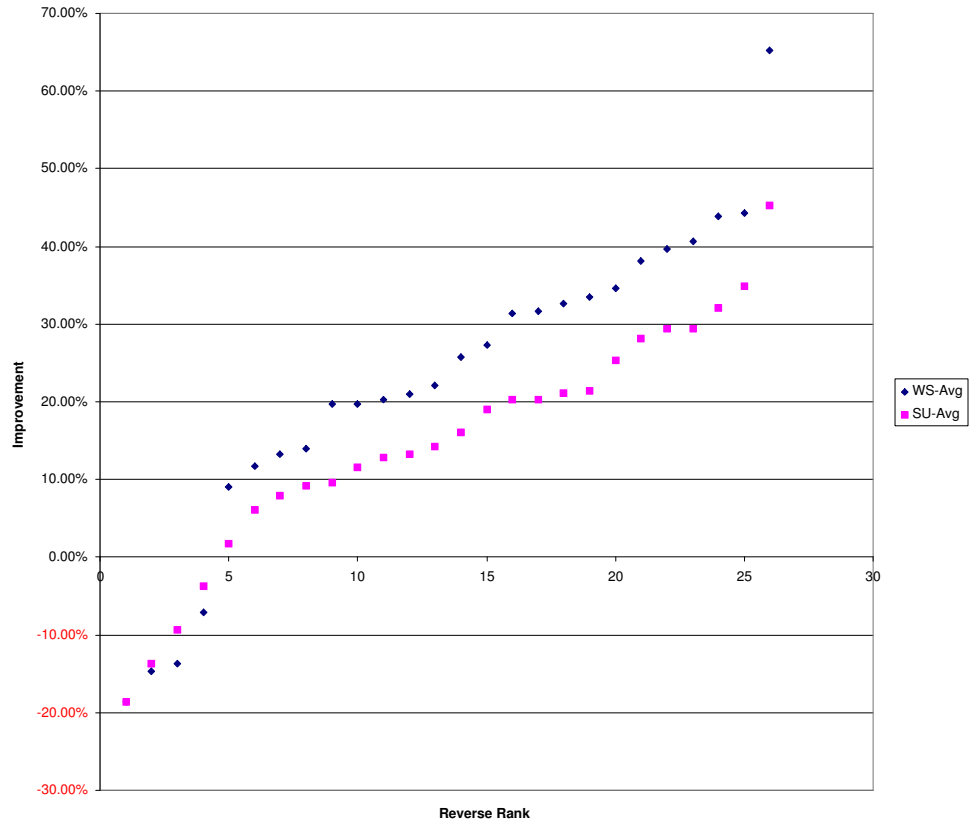


Figure 3.6: Improvements in average trail score induced by filtering.

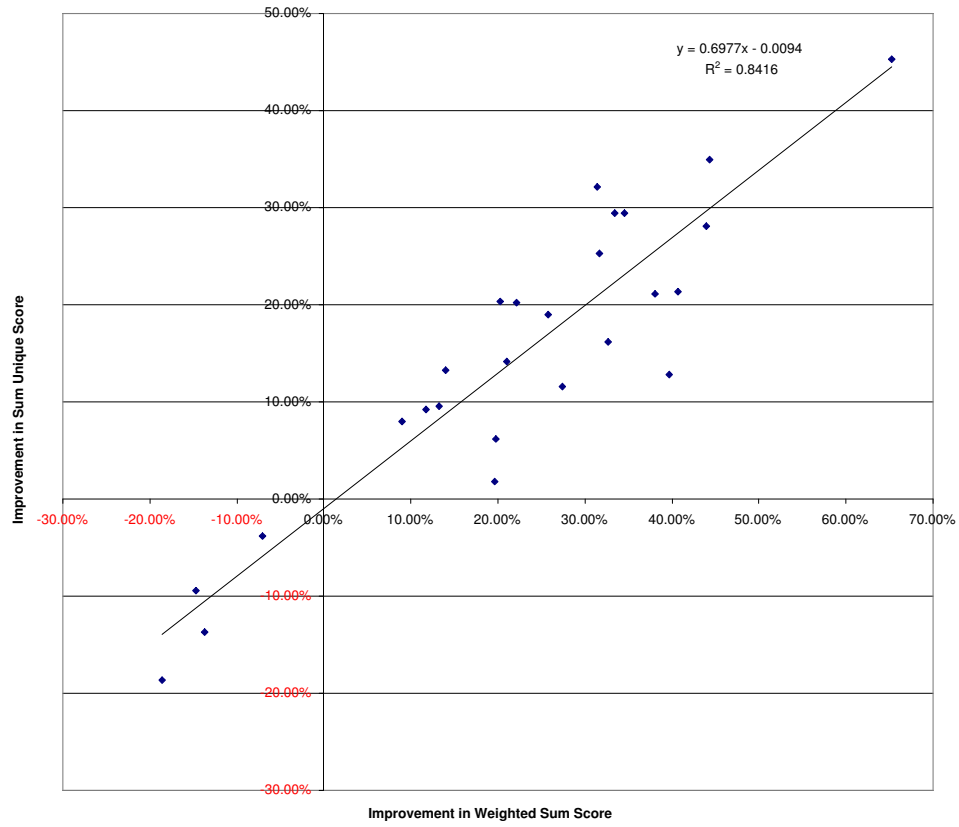


Figure 3.7: Correlation between the increase in trail scores given by the Weighted Sum and the increase shown using Sum Unique.

### 3.7 Implementation

Each node, page or URL is assigned a unique ID. IDs are 32-bit signed integers assigned in sequence (from 1) to each URL such that any two identical URLs will have an identical ID. The mapping between URLs and IDs is performed using Berkeley DB B-trees backed with large memory caches (Sleepycat Software 2001). Each page is associated with a relevance score, determined using *tf.idf* measures although they may be computed using any information retrieval metric (Salton and Buckley 1998; Baeza-Yates and Ribeiro-Neto 1999). Given a set of relevances and a graph in this form, the trails are computed by running the traversal stages in separate threads for each starting point.

There are many ways to access relevance data in constant time. For small webcases, the simplest solution of a sparsely populated array performs well. For larger webcases, hashtable lookups can be used which increase lookup time, but reduce memory overhead. The graph is stored using these identifiers in an approach which is almost identical to that used in the `Link1` database presented in Randall, Stata, Wickremesinghe, and Wiener 2002. The same optimizations and trade-offs between memory usage and speed can be applied here. Alternative strategies have been presented for returning sets of inlinks and outlinks from much larger graphs, with appropriate space-time trade-offs (Bharat, Broder, Henzinger, Kumar, and Venkatasubramanian 1998; Randall, Stata, Wickremesinghe, and Wiener 2002; Boldi and Vigna 2003).

At each step of the exploration and convergence processes, a tip must be selected based upon the probability distributions described in section 3.3. These distributions have been carefully chosen to allow the use of binary trees for storing trail score information. Figure 3.8 shows such a tree, based upon the first 2 expansions of the navigation tree shown in figure 3.3 using the weighted sum scores shown in figure 3.4.

This is efficiently implemented using arrays to form a table describing the tip selection tree at each stage. This reduces the object creation overhead which is a known issue in Java (Shirazi 2000). Associated with each tip is the sum of all relevances for all descendants, denoted as the *subscore*, and the total number of descendants which are referred to as the *subcount*. Figure 3.9 shows the table representing the tree in figure 3.8.

When selecting a tip to expand, a random number between 0 and  $x$  is selected where either  $x$  is the subscore or

$$x = \sum_{k=0}^{c-1} df^{\tau(t_k)jk}$$

where  $c$  is the subcount. This can be computed in constant time by applying the known result for the sum of a geometric series<sup>2</sup>. At each step in the subsequent traversal, this process is repeated for the nodes to the left and right of the current node, adjusting  $x$  and  $y$  appropriately. Thus, the interval in which the selected value lies can be chosen and a direction selected. Once completed a single tip will remain, which is then expanded. For example, in an exploration iteration, the process would start with the selection of a random number between 0 and 49.9809. If a number greater than 42.7505 was chosen ( $49.9809 - 7.2304 = 42.7505$ ), the process would proceed to the right. If a number less than 40.9809 was chosen, the process

---

<sup>2</sup>  $\sum_{k=x}^y a^k = \frac{a^x(1-a^{y-x+1})}{(1-a)}$

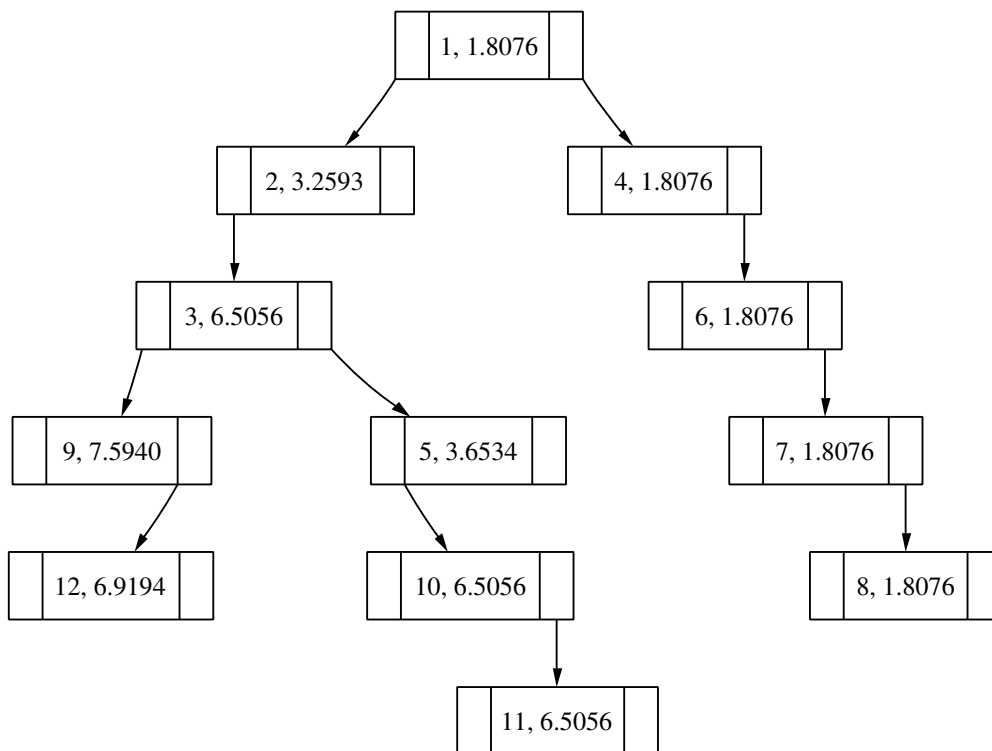


Figure 3.8: Tree of Tips for the navigation tree shown in figure 3.3. The table representing this tree is shown in figure 3.9.



Tip	Weighted Sum	Left	Right	SubScore	SubCount
1	1.8076	2	4	49.9809	12
2	3.2593	3		40.9429	7
3	6.5056	9	5	37.6836	6
4	1.8076		6	7.2304	4
5	3.6534	10		16.6646	3
6	1.8076		7	5.4228	3
7	1.8076		8	3.6152	2
8	1.8076			1.8076	1
9	7.5940		12	14.5134	2
10	6.5056		11	13.0112	2
11	6.5056			6.5056	1
12	6.9194			6.9194	1

Figure 3.9: Table showing candidate tips for expansion. SubScore is the sum of the scores for the current node and all descendants and SubCount is the number of active nodes reachable from that node. It should be noted that the nodes in this tree represent tips and should not be confused with either the nodes of the graph or the navigation tree produced by the Best Trail.

would proceed to the left.

### 3.8 Complexity

The previous section showed how the step  $select(D_i, df, j)$  could be implemented to run in time  $O(\log(n))$  where  $n$  is the number of candidate tips. The function  $best()$  has the same time complexity, but is slightly simpler in that each iteration is to the left of the current node.

Hence, the worst case complexity of algorithm 2 (figure 3.2) using this implementation can be given as  $O(KMI^2\beta^2)$  where  $I = I_{explore} + I_{converge}$  and  $\beta$  is the maximal outdegree of any link in  $E$ . This can be broken down as follows:

$I\beta$  as the worst-case insertion time for a tip. This factor emanates from the fact that the tree of tips may become a linked list if all new tips are added to the same part of the tree. This might occur in the simple case of nodes having identical scores, so these scores are biased using tiny random numbers to adjust the rank. The magnitude of these adjustments means that they affect only the speed of the operation, not the end results.

$\beta$  representing the number of potential tips which may be added to the candidate set at each iteration. This number would always be added on a fully connected graph, but graphs based upon Web data are very sparse and this will never occur in practice.

$KMI$  as the maximum number of iterations the Best Trail may take to find the given trails.

In practice the tree of tips is unlikely to be skewed to such a degree. Nor is the graph likely to be fully-connected. However, if the average-case complexity is estimated by substituting the average outdegree, the results are still inaccurate. Using the *weighted average outdegree* better models the expansion of the navigation tree during the exploration and convergence phases.

The weighted outdegree,  $W$ , of a node,  $n$ , is defined as the product of the number of outlinks  $(n, x)$  from that node and the proportion of links in the graph which point to that node  $\frac{|n, y \in E|}{|E|}$ . The weighted average outdegree for a graph  $G = \langle N, E \rangle$  can thus be calculated as:

$$W(N, E) = \frac{|\{x, y, z \mid (x, y) \in E \wedge (y, z) \in E\}|}{|E|}$$

The difference between these values and the average outdegree can be seen in figure 3.10, which shows the average outdegree and weighted average outdegree, for the webgraphs of eight corpora. It is assumed that all links are as likely to be followed as any others, given a sufficient number of queries. It should be noted that, when expanding a navigation tree, the number of potential trails to a depth of  $d$  is roughly equal to  $\sum_{i=1}^d w^i$ . where  $w$  denotes the weighted average outdegree of a graph.

The average case complexity can now be given as  $O(KMI\beta \log(I\beta))$  where  $\beta$  is the weighted average outdegree. Using binary trees the average-case complexity of the *expand* operation is  $O(\beta \log \beta I)$  since there are, on average,  $\beta$  elements to be added to the list of candidate tips and the complexity of operation to insert these new candidates is equal to that of the *select* function –  $O(\log \beta I)$ .

Webcase	Nodes	Pages	Links	Average Outdegree	Average Weighted Outdegree
Sleepycat	2 939	1 106	11 820	10.7	29.3
SCSIS	6 868	2 448	15 055	6.1	5.1
UCL	754 322	201 900	2 593 912	12.9	24.2
UCL-CS	186 239	37 957	748 734	19.8	66.1
Intel	311 841	128 890	10 776 299	25.0	297.6
Birkbeck	301 575	64 593	781 514	12.2	13.1
TREC	7 618 783	1 433 848	20 981 549	14.8	9.6
DTI	19 919	6 093	79 263	13.0	18.7
JDK 1.4	48 923	8 114	360 458	44.4	119.5

Figure 3.10: Properties of the test corpora. There are more nodes than pages in the graphs, as many of the pages refer to other pages not in the corpus.

### 3.9 Performance Evaluation

Numerous experiments have been conducted to test the behaviour of the algorithm and explore the effect of the various parameters which control it. Eight corpora have been used during these experiments. They are a crawl of Sleepycat software’s site, at [sleepycat.com](http://sleepycat.com); a crawl of the School of Computer Science and Information Systems (SCSIS) website, at [www.dcs.bbk.ac.uk](http://www.dcs.bbk.ac.uk); a crawl of the main site of Birkbeck University of London, covering all domains under [bbk.ac.uk](http://bbk.ac.uk); a crawl of the main Web site of University College London (UCL), at [www.ucl.ac.uk](http://www.ucl.ac.uk); a crawl of the UCL Computer Science (UCL-CS) department’s site at [www.cs.ucl.ac.uk](http://www.cs.ucl.ac.uk); a crawl of Intel’s site, [intel.com](http://intel.com); a crawl of the Web site of the Department of Trade and Industry (DTI), at [www.dti.gov.uk](http://www.dti.gov.uk); a corpus generated from the files of the TREC WT10g WebTrack data set (Bailey, Craswell, and Hawking 2001); and the AutoDoc webcase generated from the Javadocs of the Java Development Kit (JDK) version 1.4, which is discussed in greater detail in chapter 8. Figure 3.10 gives details of the corpora, which were chosen to provide a mix of academic, technical, commercial and government sites. Queries were selected from various query logs. The examples shown use queries chosen to highlight the effects which occur in general for those queries with non-trivial links between relevant pages.

Behaviour of the algorithm is controlled by the parameters  $df$ ,  $I_{explore}$ ,  $I_{converge}$ ,  $M$  and the set of starting points  $\{U_0, U_1, \dots, U_K\}$ . As would be expected, increasing the value of either of the parameters  $I_{explore}$  or  $I_{converge}$  produces higher scoring trails on average. Figure 3.11 shows the influence of increasing values of  $I_{explore}$  on the score of the trail using the query “Swing” on the AutoDoc (JDK1.4) webcase, whilst figure 3.12 shows the effect of increasing  $I_{converge}$  under the same conditions. Unsurprisingly, increasing  $I_{converge}$  finds the local limit of the trail score faster than increasing  $I_{explore}$ , as shown by the sharp rise at the very start of the curve. Perhaps more surprising is the behaviour when altering the ratio between  $I_{explore}$  and  $I_{converge}$ . Increasing  $I_{explore}$  whilst decreasing  $I_{converge}$  increases the scores of the resulting trails if the relevance is measured using *sum distinct* but decreases the trail score when calculated using the *weighted sum* (figures 3.13 and 3.14). The balance between the values  $I_{explore}$  and  $I_{converge}$  can be tuned to reflect the importance of the two metrics, but a more

consistent behaviour would have been expected.

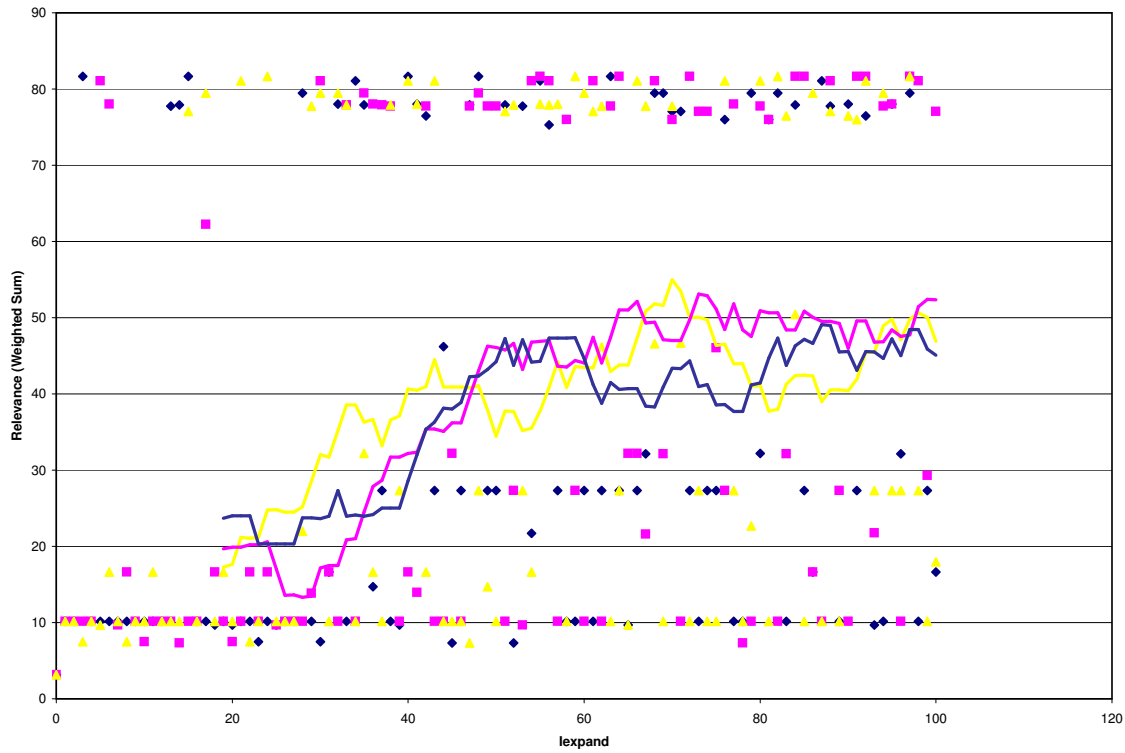


Figure 3.11: Increasing the number of exploration iterations increases the scores of the returned trails. The algorithm slowly tends to a limit, whilst exploring the solution space. Blue, purple and yellow trend lines show the pattern for 1, 5 and 20 repetitions respectively.

Increasing the value of  $M$  (the number of repetitions) is less effective, as repeated exploration from the same node causes many of the expansions to be duplicated in other trees. The multi-threaded environment can be used more effectively by expanding from a greater number of starting points, as shown in figure 3.15.

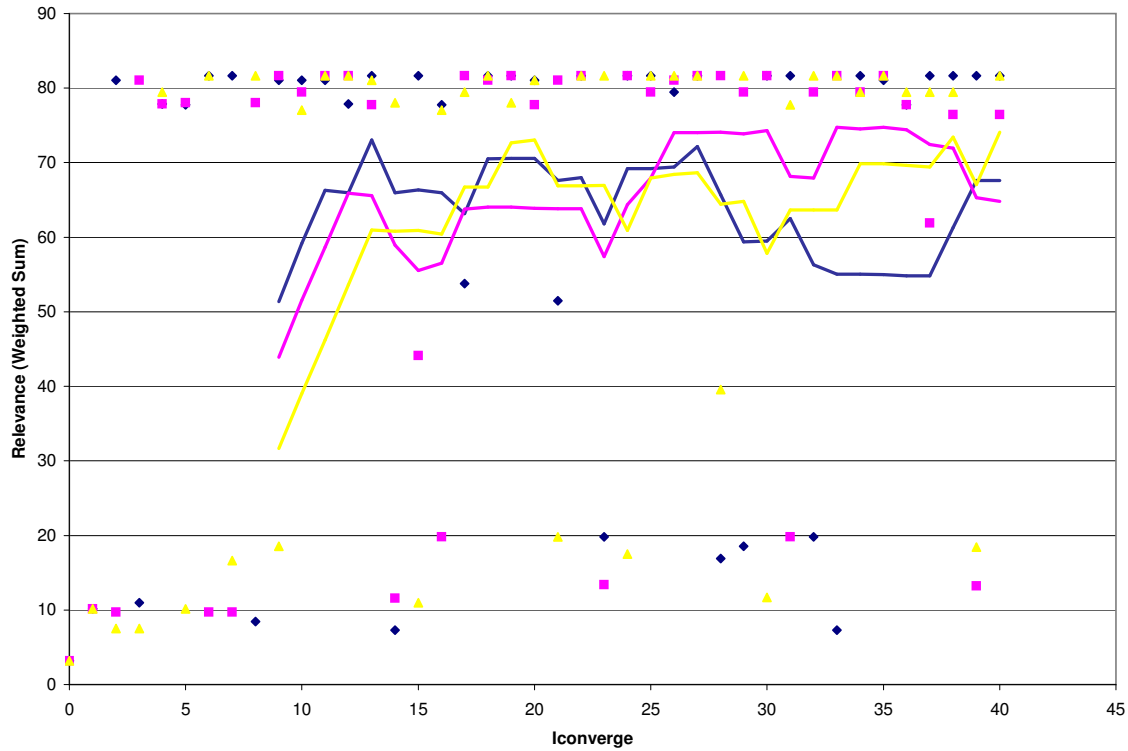


Figure 3.12: Increasing the number of convergence iterations increases the scores of the returned trails. The algorithm quickly tends to a limit. Blue, purple and yellow trend lines show the pattern for 1, 5 and 20 repetitions respectively.

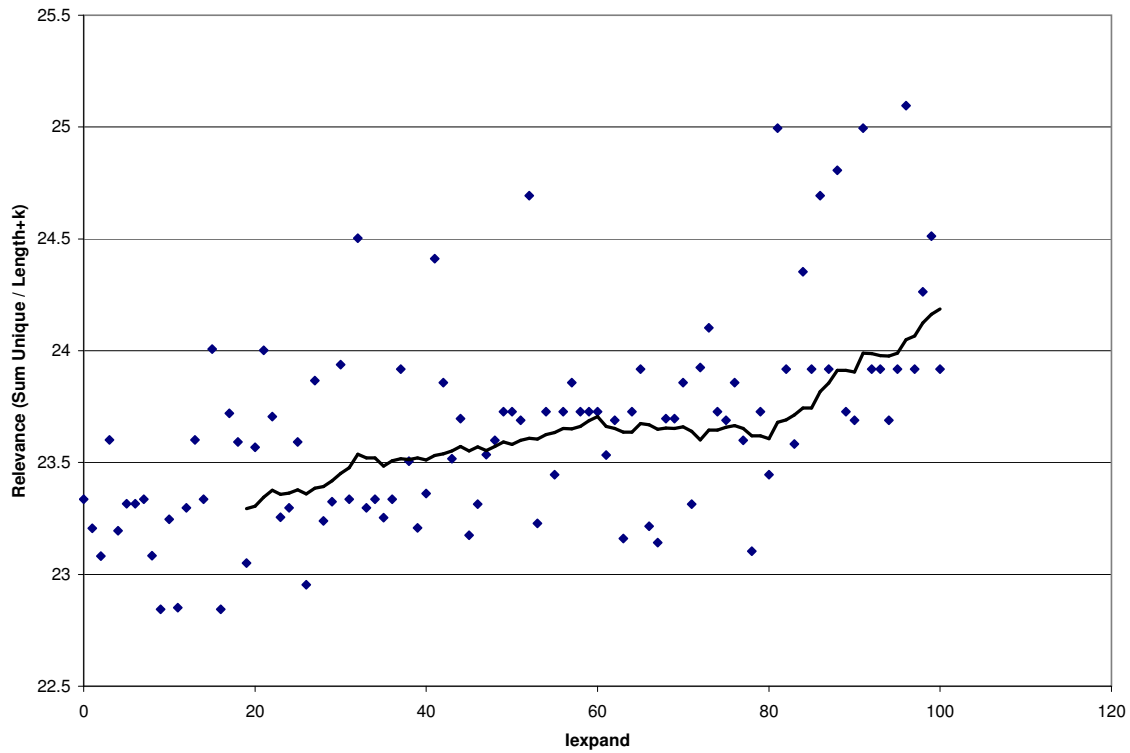


Figure 3.13: Increasing  $I_{expand}$ , whilst decreasing  $I_{converge}$  increases the resultant trail scores when calculated using *sum distinct*. The graph shows values for  $0 \leq I_{explore} \leq 100$  and  $I_{converge} = 100 - I_{explore}$ . A moving average trend line is shown to highlight the effect.

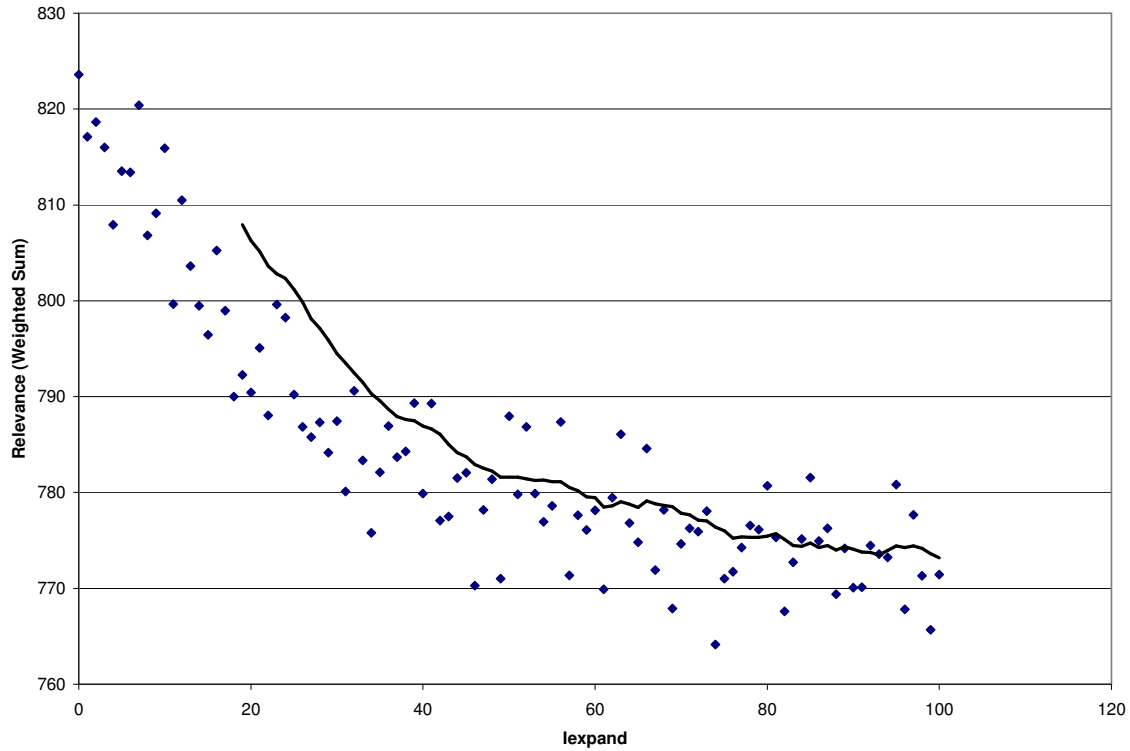


Figure 3.14: Increasing  $I_{expand}$ , whilst decreasing  $I_{converge}$  decreases the resultant trail scores when calculated using the *weighted sum*. The graph shows values for  $0 \leq I_{explore} \leq 100$  and  $I_{converge} = 100 - I_{explore}$ . A moving average trend line is shown to highlight the effect.

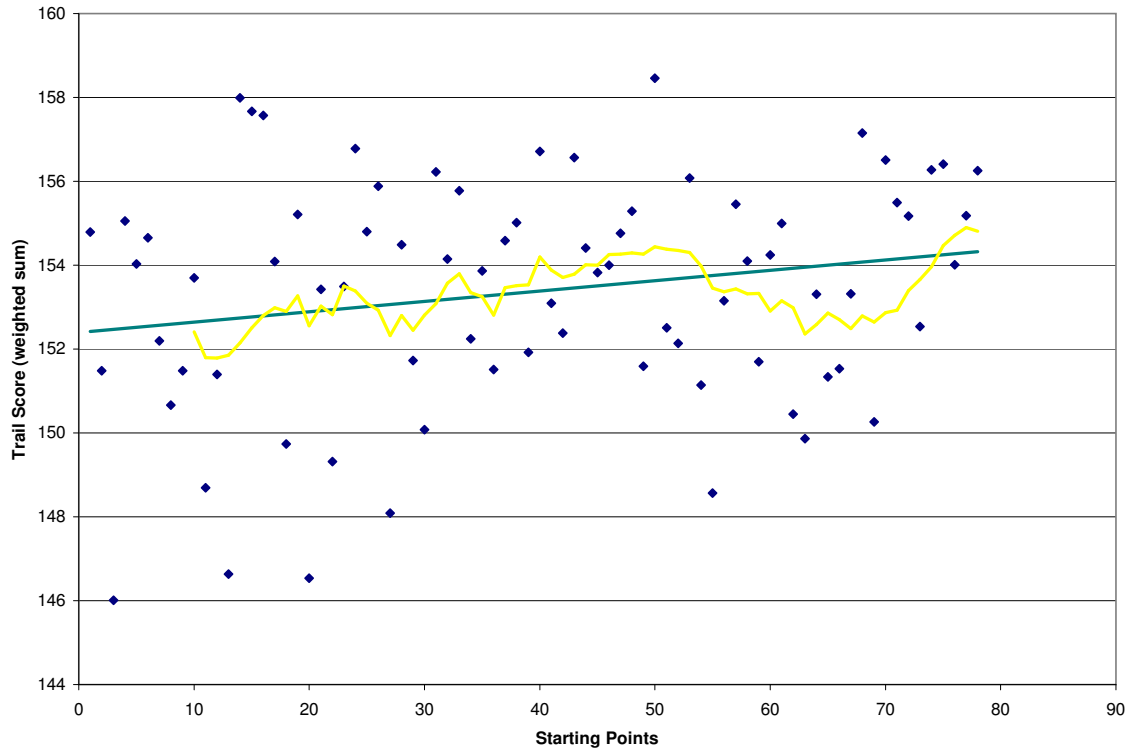


Figure 3.15: Increasing the number of starting points increases the score for trails, by allowing a greater number of opportunities for discovery. Trail sets are truncated to the same size. Moving average and linear trend lines are shown to highlight the effect.



### 3.10 Concluding Remarks and Future Work

This chapter has presented an algorithm for finding trails across the graph of linked pages in a Web site. Inspired by Bush's memex, these trails provide a structure to the returned results and provide users with contextual information not provided by traditional search facilities. The following chapter will show how link-analysis can be used to select starting points which lead to higher-scoring trails.

It is also possible to use the Best Trail algorithm to solve optimization problems such as TSP, or closely related problems such as the Hamiltonian Path Problem (HPP). Experiments have been performed to evaluate the effectiveness of using the Best Trail algorithm for solving TSP. Trails have been scored primarily using the formula:  $n^{\alpha}l^{\beta}$  where  $n$  is the number of nodes on the trail,  $l$  is the length of the trail and  $\alpha \geq 0$  and  $\beta \leq 0$  are constants. Anecdotal evidence has suggested that regardless of the choice of values, the Best Trail is consistently outperformed by ant colony optimization methods using values of  $\alpha = 0$  and  $\beta = 1$ .

The "no free lunch" theory states that some algorithms will always perform worse on some inputs than others, and that for any pair of optimization algorithms,  $A$  and  $B$ , there will always be as many inputs on which  $A$  outperforms  $B$  as inputs on which  $B$  outperforms  $A$  (Wolpert and Macready 1995; Wolpert and Macready 1997). Future work will investigate why the Ant Colony approach seems to out-performs the Best Trail in finding solutions to TSP, and why the Best Trail algorithm appears to out-perform the ant colony optimization approach for Web site trail finding.

Irrespective of the algorithm used to compute the trails, better scoring functions are required which reflect the needs and desires of users. These functions should maintain and enhance the existing virtues of relevance and conciseness. They should also prevent topic drift, discriminate between local and global links and emphasise the differences between embedded, structural and associative links (Nielsen 2000).

## Chapter 4

# Navigability and Starting Point Selection

For what shall it profit a man, if he shall gain the whole world, and lose his own soul?

Mark 8:36

To gain that which is worth having, it may be necessary to lose everything else.

Bernadette Devlin

If I were to wish for anything, I should not wish for wealth and power, but for the passionate sense of potential.

Søren Kierkegaard

## 4.1 Introduction

In order to find information in the Web, surfers often adopt the following two stage strategy. First, they submit their query to a search engine, such as Google, AltaVista or Fast, which directs them to a page within a Web site containing information relevant to whatever they are seeking, and secondly they navigate within this Web site by following hyperlinks until they either find the desired information, or they repeat the process by reformulating their original query (Nielsen 1997).

The previous chapter described an algorithm for automating the navigation process, given a set of suitable starting points. Whilst investigating a Web site by exploration from highly relevant pages is effective, better results can be achieved by considering future navigation opportunities in the starting point selection. This chapters describes a new metric, measuring the usefulness of a page in terms of the number of trails in a graph starting from a given node.

Although search engines most likely weight home pages higher than other pages and use metrics such as HITS and PageRank to determine authority, they do not have a general mechanism to take into consideration the *navigation potential* of Web pages. Once such a measure is available it can be weighted into the user query in order to find “good” starting points for navigation given the actual user query. From now on, this navigability measure will be referred to as the *Potential Gain* of a Web page. The application that initially led to the development of the Potential Gain metric is the navigation engine described in this thesis, but this notion should have wider applicability within the general context of Web search tools.

It is assumed that the only information available is the score, or relevance, of a URL to the user’s query or information seeking goal, and the topology of the Web. A Web page should be chosen from which to start navigation that maximises the probability that the user’s information need may be satisfied. If the density of the neighbourhood of some starting URL,  $u$ , is greater, many more pages can be reached in a short distance, then the Potential Gain, or utility, of  $u$  is high. This leads to a formalization of Potential Gain in terms of the number of trails in the induced by an out-tree.

The rest of the chapter is organised as follows:

**Section 4.2** describes the Potential Gain metric and the related Gain Rank metric, which can be used as an authority measure, similar to the HITS authority rank or the PageRank.

**Section 4.3** describes two algorithms used to compute Potential Gain and Gain Rank – one iterative, and one using matrices.

**Section 4.4** shows that convergence of the set of Potential Gain values occurs within a few iterations and that the resultant values are distributed according to a power law.

**Section 4.5** describes investigations into the correlation between various ranking metrics, proving that Potential Gain measures something distinct from that which is measured by other ranking measures.

**Section 4.6** describes experiments into the usefulness of Potential Gain as a mechanism for selecting starting points. The key result is that incorporating potential gain into the starting point selection measure increases the relevance of the resulting trails.

**Section 4.7** concludes with ideas for future enhancements and further application of the Potential Gain metric.

## 4.2 Potential Gain and Related Metrics

Informally, a “good” starting node is one which is relevant, central and is connected to a large number of other nodes (Botafogo, Rivlin, and Shneiderman 1992; Mukherjea and Foley 1995). Several metrics have been proposed for selecting nodes in search results which relate to the issue of starting point selection. The most famous, the PageRank citation ranking (Page, Brin, Motwani, and Winograd 1998) only considers the effect of incoming links, whilst Kleinberg’s HITS metrics (Kleinberg 1998) and extensions of it (Lempel and Moran 2000) only consider the effect of single links in each direction. The Potential Gain metric considers the effect of more distant pages.

Starting URLs should satisfy the following criteria:

1. They are *relevant*, i.e. their score with respect to the user’s goal is high.
2. They are *central* in that their distance to other pages is minimal.
3. They are *connected* in that they are able to reach a maximal number of other pages.

If the starting pages are not relevant, it will be difficult for users to navigate from them and difficult for the Best Trail algorithm to find quality trails. If they are not central, then excessively long trails may be required to find the relevant content. If they are not connected, then vital pages are likely to be missed.

Keeping with the definitions introduced in chapter 3, the Web is viewed as a hypertext system,  $H$ , having two components: a directed graph,  $G = (N, E)$ , having a finite set of nodes (or URLs),  $N = \{U_1, U_2, \dots, U_n\}$ , and edges (or links),  $E$ , and a scoring function,  $\mu$ , which is a function from  $N$  to the set of positive real numbers.

### 4.2.1 Potential Gain and Gain Rank

The *Potential Gain*,  $Pg(p)$ , of a page,  $p$ , is defined as the sum for all lengths (or depths, if trails are viewed as paths in an out-tree) of the product of the fraction of all possible trails which are of length  $d$  and a discounting function  $f(d)$ . The Potential Gain (or utility) of a trail of length  $d$  is measured in terms of this function.

Formally, if the fraction of trails,  $R_d$ , to a depth,  $d > 0$ , from a node,  $n \in N$  is given by

$$R_d(n) = \frac{\sum_{y \in \text{Out}(n)} R_{d-1}(y)}{\sum_{j \in N} R_d(j)}$$

where  $R_0 = 1$  and  $\text{Out}(i) = \{j \mid (i, j) \in E\}$ , then the Potential Gain of  $n$  is given by

$$Pg(n) = \sum_{d=1}^{d_{max}} R_k(n) f(d)$$

The constant,  $R_0 = 1$  is used to ensure that  $\log Pg(n) > 0$  holds for all  $n$ . The importance of this inequality will become clear in section 4.6.

An authority function similar to the PageRank can be defined in terms of trails. The *Gain Rank* of  $p$  is defined as the product of the discounting function and the fraction of trails of length  $d$  which terminate at  $p$ . This is formally defined using the same formulae as for Potential Gain, but with  $Out(n)$  replaced by  $In(n)$  where  $In(i) = \{j | (j, i) \in E\}$ .

### 4.2.2 Discounting Functions

Two reasonable functions for  $f(d)$ , when computing either Potential Gain or Gain Rank, are the reciprocal function:

$$f(d) = \textit{reciprocal}(d) = d^{-1}$$

and the geometric decay function:

$$f(d) = \textit{decay}(d) = \gamma^d$$

where  $0 < \gamma < 1$  is a constant. The justification for these measures is based on the assumption that the utility of browsing a page diminishes with the distance of the page from the starting URL. This assumption is consistent with experiments carried out on real Web data (Huberman, Pirolli, Pitkow, and Lukose 1998; Levene, Borges, and Loizou 2001), and with studies showing that the probability of a user following a path of length  $n$  decreases as  $n$  increases (Silverstein, Henzinger, Marais, and Moricz 1999).

### 4.3 Computing Potential Gain

The Potential Gain is computed by algorithm 3 (figure 4.1). The algorithm takes three parameters – a directed graph,  $G = (N, E)$ , describing the topology of the Web site; a number,  $d_{max} > 1$ , defining how many iterations should be performed; and the discounting function,  $f$ . For the following experiments, the reciprocal function  $f(d) = d^{-1}$  is used. This algorithm clearly takes time in the order of  $O(d_{max}|E|)$  and requires space in the order of  $O(|N|)$ , in addition to the space required to store the graph. The algorithm computes values for all nodes in advance and is suitable for an in-memory graph as used with Web sites. For larger graphs similar techniques can be utilized to those proposed for the PageRank citation metric (Page, Brin, Motwani, and Winograd 1998; Haveliwala 1999; Kamvar, Haveliwala, Manning, and Golub 2003b; Kamvar, Haveliwala, and Golub 2003; Kamvar, Haveliwala, Manning, and Golub 2003a).

**Algorithm 3** ( $\text{Gain}(G, d_{max}, f)$ )

```

1. begin
2.   for  $d = 1$  to  $d_{max}$ 
3.     foreach  $n \in N$ 
4.        $c[1, n] \leftarrow \sum_{(n,o) \in E} c[0, o]$ 
5.        $nf \leftarrow nf + c[1, n]$ 
6.     end foreach
7.     foreach  $n \in N$ 
8.        $c[1, n] \leftarrow c[1, n]/nf$ 
9.        $Pg[n] \leftarrow Pg[n] + c[1, n] * f(d)$ 
10.    end foreach
11.     $c[0] \leftarrow c[1]$ 
12.  end for
13.  return  $Pg$ 
14. end.

```

Figure 4.1: The algorithm for computing Potential Gain values where  $Pg$  represents the array of gain values,  $c$  represents the array of normalized reference counts,  $nf$  represents a normalization factor. The array  $c$  is initialized such that  $c[0, n] = 1$  for all  $n \in N$ .

The Potential Gain can also be defined in terms of the adjacency matrix,  $M$  for the graph  $G$ . Given such a matrix and a vector,  $R_{init}$  of initial values where each element of  $R_{init}$  is set to the reciprocal of the number of nodes in  $G$  ( $|N|^{-1}$ ), the Potential Gain can be calculated using algorithm 4.

A weighted matrix,  $A$  can be given by  $A = \text{diag}(\text{ones}(1, S) ./ (1 + \text{Outdegree})) * M$  where  $\text{Outdegree} = \text{sum}(M')$ ,  $S$  is the size of  $M$  and  $./$  denotes matrix right division. It is the matrix  $A$  which serves as the basis of the matrix definition for PageRank (Page, Brin, Motwani, and Winograd 1998).

**Algorithm 4** ( $\text{Gain}(M, R_{init}, d_{max})$ )

1. **begin**
2.   **for**  $d=1$  **to**  $d_{max}$
3.      $R = R_{init} * (M^d)$
4.      $df = \text{sum}(R)$
5.      $R = R/df$
6.      $Pg = Pg + d^{-1}R$
7.   **end for**
8.   **return**  $Pg$
9. **end.**

Figure 4.2: The algorithm for computing Potential Gain using matrix transformations.



## 4.4 Experiments

Several experiments have been conducted to test various properties of the Potential Gain and Gain Rank metrics. These have been performed using the same corpora detailed in section 3.9, and summarized in figure 3.10.

### 4.4.1 Convergence

The algorithm to compute Potential Gain was described as taking time proportional to  $O(d_{max} \cdot |E|)$ . In practice, after a brief settling period, convergence to a set of Potential Gain values occurs in a short space of time. Figure 4.3 shows this convergence process on the TREC WT10g corpus. The same process occurs with all corpora, albeit at slightly different rates. The probable error refers to the level of discrepancy between the values in the current iteration and those in the previous one.

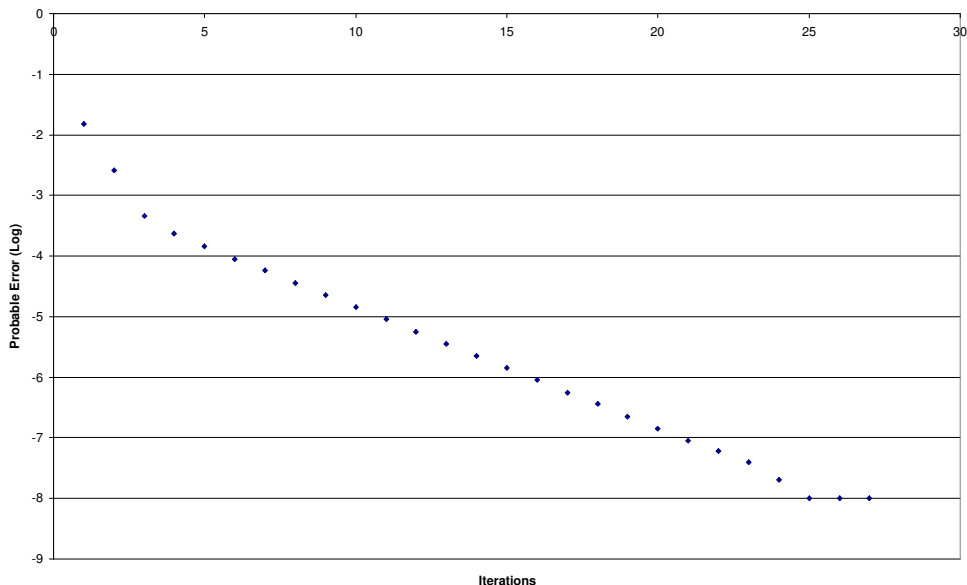


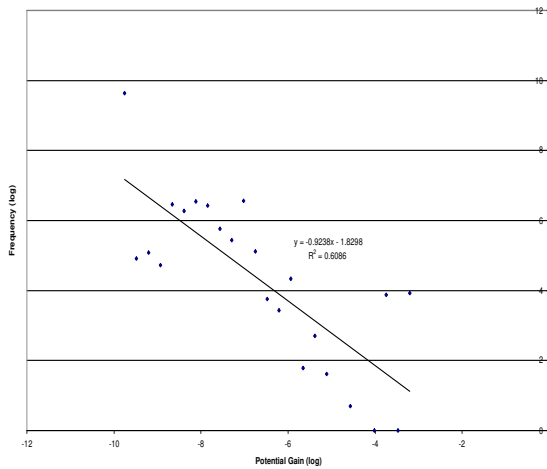
Figure 4.3: Potential Gain values converge rapidly in a few iterations.

### 4.4.2 Power Law Distributions

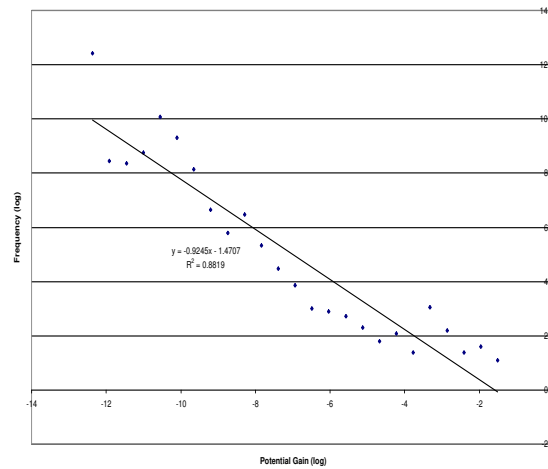
Power law relationships have been found for PageRank and many other Web-related phenomena (Adamic 2002; Broder, Kumar, Maghoul, Raghavan, Rajagopalan, Stata, Tomkins, and Wiener 2000; Pandurangan, Raghavan, and Upfal 2002). Given the power-law distribution in Web page outdegrees, it was predicted that the Potential Gain metric, which is a function of the outdegree should be similarly distributed.

Figure 4.4 shows that bucketed values for Potential Gain appear to follow a power-law distribution, with exponents of between 0.9238 and 1.1488 for the Web sites of the DTI, Birkbeck

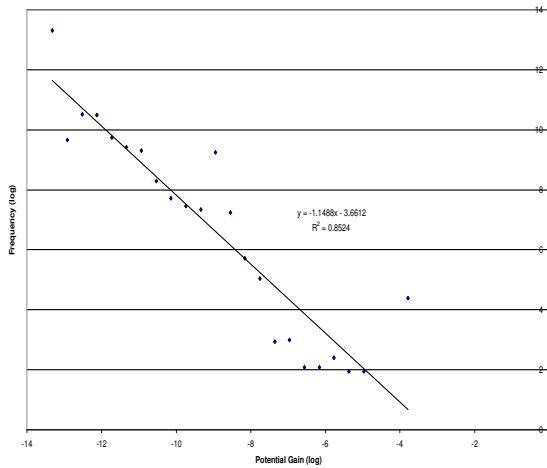
and UCL and for the TREC WT10g corpus. However, figure 4.5 shows that the potential gain values for the the page on the Sleepycat web site are not distributed according to a power law. One possible explanation for this is that the extensive program documentation found on the site has links which are not distributed according to any power law. The hypothesis that this is common to all such documentation can be rejected for two good reasons. Firstly, the JDK 1.4 Javadocs have Potential Gain values which are distributed in such a manner (figure 4.5). Secondly, the structure of object-oriented programs leads directly to power law distributions, as will be shown in chapter 8. An alternate hypothesis is that the result is due to systematic link creation by a single user. This hypothesis is also rejected, as the site and the documentation was built up over many years and is maintained by a large group of people.



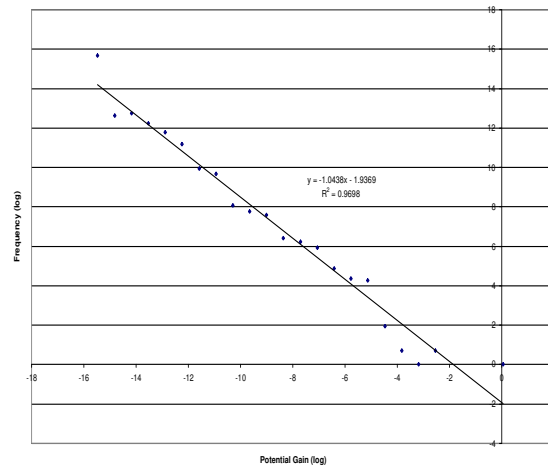
(a) DTI



(b) Birkbeck



(c) UCL



(d) TREC

Figure 4.4: Log-Log plots showing power law distributions in the values of Potential Gain for the web sites of (a) the DTI, (b) Birkbeck and (c) UCL and also for the pages of the TREC WT10g corpus.

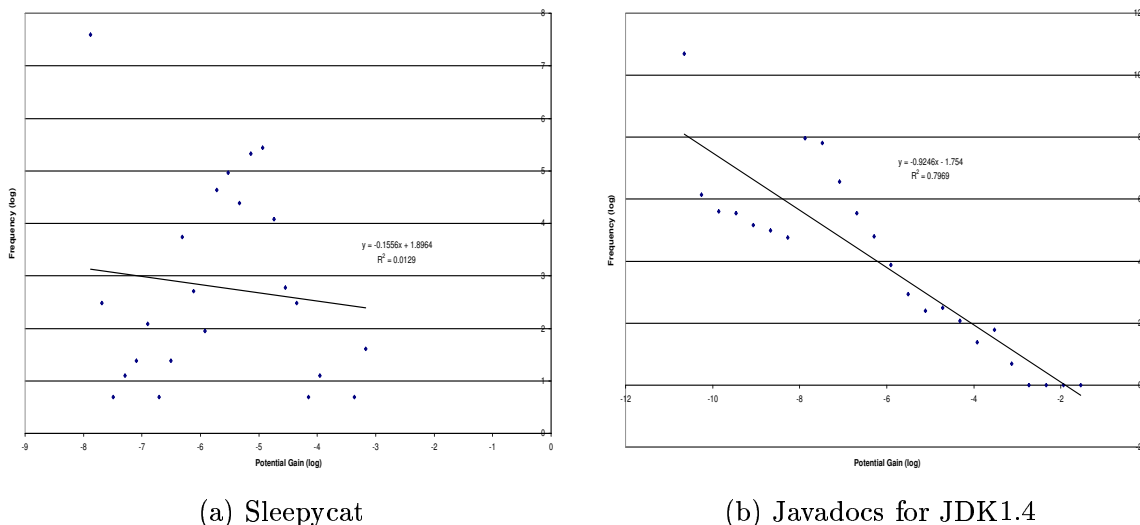


Figure 4.5: Log-Log plots showing (a) that there is no power law in the distribution of Potential Gain values on the Sleepycat web site (b) that there is such a distribution in the values of potential gain for pages within the JDK 1.4 Javadocs.

## 4.5 Correlations between Ranking Metrics

The next two sections will attempt to answer two key questions. This section will attempt to answer the question of whether or not calculating the Potential Gain for the pages in a corpus tells us something new about the pages within it, or whether Potential Gain is simply a reformulation of existing metrics. The following section, will attempt to answer the question of whether Potential Gain is a useful metric in the selection of starting points for expansion via the Best Trail algorithm. A third, and equally important question is whether the Potential Gain approximates any human assessment of the pages in questions. Unfortunately, answering this question requires a great deal of time and resource and is beyond the scope of this thesis.

### 4.5.1 Experimental Methods

It can be established that the Potential Gain provides something novel, by direct comparison of the values and the rankings provided by various Web metrics. Analysis has been performed on the correlation matrices for values of Potential Gain, Gain Rank, PageRank, indegree and outdegree as well as scores computed using Kleinberg’s HITS algorithm (Kleinberg 1998), Foley’s metric for identifying landmark nodes (Mukherjea and Foley 1995) and the logarithms of all these values.

Correlations were computed using Pearson’s product moment correlation coefficient ( $r^2$ ), Kendall’s  $\tau$  statistic and Spearman’s  $\rho$  - the latter two being considered most appropriate for measuring non-linear correlation (Everitt 1998):

**Pearson’s  $r^2$**  product moment correlation coefficient.  $r$  is “an index that quantifies the

linear relationship between a pair of variables” (Everitt 1998) and is given by:

$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2 (y_i - \bar{y})^2}}$$

**Kendall’s  $\tau$**  shows a correlation between two rankings, and is given by:

$$\tau = \frac{2S}{n(n-1)}$$

where  $S = P - Q$  where P is the number of pairs where the observed rankings are equal and Q is the number where the observed rankings are in the opposite direction.

**Spearman’s  $\rho$**  is equivalent to a Pearson correlation on the rankings of the values. If the ranked values of two variables are  $a_i$  and  $b_i$  and  $d_i = a_i - b_i$  then  $\rho$  is given by:

$$\rho = 1 - \frac{6 \sum_{i=1}^n d_i^2}{n^3 - n}$$

Pearson’s product moment correlation coefficient was used to construct a set of correlation matrices showing linear correlation between each of the Web metrics. The webcases from Sleepycat, SCSIS, UCL, UCL-CS, Intel, Birkbeck, DTI and JDK 1.4 were all tested. The resulting matrices are given in appendix C. The non-linear correlation metrics (Kendall’s  $\tau$  and Spearman’s  $\rho$ ) were used to construct a second set of matrices, showing non-linear correlation. These are given in appendix D.

#### 4.5.2 Discussion

Some evidence was found of correlation between PageRank and indegree and between Gain Rank and indegree but this is neither consistent nor statistically significant. The analysis confirms the result that PageRank is not strongly correlated with indegree – despite being proposed as a predictor of indegree values (Pandurangan, Raghavan, and Upfal 2002; Page, Brin, Motwani, and Winograd 1998).

The most pertinent question is whether there is evidence of correlation in values or rankings between Potential Gain and the other hub metrics such as outdegree or Kleinberg’s hub rank. Strong correlations have been found between between the rankings induced by all three metrics. However strong *linear* correlations exist for some of the Webcases specified. If each combination is considered, then on at least one real-world graph, the correlation is low. This phenomena is interesting, but its explanation is beyond the scope of this thesis. Since the metrics are clearly related but evidently distinct, investigation is required into which metric is most effective in increasing the scores (and relative quality) of the returned trails.

## 4.6 Improving Starting Point Selection

In order to evaluate the effectiveness of the Potential Gain metric in improving trail finding, the following experiment was performed. Trails were found by traversing the graph from starting points selected by taking the top 15 pages. These pages were selected by sorting the list, firstly by using the number of keywords matched in the trails and in the documents, then by each the measures shown in figure 4.6, in which  $\mu(p)$  denotes the relevance to the query of  $p$ ,  $Out(p)$  denotes the outdegree of  $p$ ,  $Hub(p)$  denotes the hub score of  $p$  as computed by HITS,  $Pg(p)$  denotes the Potential Gain of  $p$ ,  $Found(p)$  denotes the order in which the URLs were indexed and  $a = \max(\frac{2}{3}, 1 - \frac{1}{20} \log(C))$  where  $C$  is the number of documents matching the query. The values for  $a$  and  $C$  come from Pinkerton's metrics, as described in section 2.6 (Pinkerton 2002).

The first measure ( $\mu(p)$ ,  $Found(p)$ ) was used as a baseline against which the others were compared. This baseline measure is implied naturally by the order of pages in the posting lists, yet should still lead to good trails. The Web robot uses a BFS crawling strategy, which has been shown to lead to high quality pages (Najork and Wiener 2001). Hence, preferring pages which were found earlier in the crawl should be beneficial. However, link-analysis techniques can lead to better results.

$$\begin{array}{ll}
 \mu(p), Found(p) & \mu(p), Pg(p) \\
 \mu(p), Out(p) & \mu(p), Hub(p) \\
 \mu(p)Pg(p) & \mu(p)Out(p) \\
 \mu(p)Hub(p) & \mu(p) \log Pg(p) \\
 \mu(p) \log Out(p) & \mu(p) \log Hub(p) \\
 \log \mu(p) \log Pg(p) & \log \mu(p) \log Out(p) \\
 \log \mu(p) \log Hub(p) & a\mu(p) + (1 - a)Pg(p)
 \end{array}$$

Figure 4.6: Metrics used in tests of Potential Gain as a starting point selection metric.

The tests were conducted using eight data sets – based upon six corpora, summarized in figure 4.7, and eight query sets, summarized in figure 4.8.

Data Set	Corpus / Webcase
Birkbeck	Crawl BirkBeck College Web site
SCSIS	Crawl of the School of CS&IS Web site
UCL-CS	Crawl of UCL's Computer Science Web site
TREC	TREC WT10g data set
DBLP-Authors	Index of DBLP data (see chapter 7)
DBLP-Docs	Index of DBLP data (see chapter 7)
JDK	JDK 1.4 Javadocs
JDK-Classes	JDK 1.4 Javadocs

Figure 4.7: Corpora used in tests of Potential Gain as a starting point selection metric.

Figure 4.9 shows the results of experiments into the effectiveness of the first four measures in figure 4.6 in which the measures  $Found(p)$ ,  $Pg(p)$ ,  $Out(p)$  and  $Hub(p)$  are used as an

Data Set	Query Set
Birkbeck	Queries taken from HT://Dig logs
SCSIS	Queries taken from HT://Dig logs
UCL-CS	Queries taken from Glimpse logs
TREC	Queries 451-500
DBLP-Authors	List of Citeseer's most popular Authors
DBLP-Docs	List of Citeseer's most popular documents
JDK	Queries taken from AutoDoc Logs
JDK-Classes	List of Java Classes

Figure 4.8: Queries used in tests of Potential Gain as a starting point selection metric.

additional sorting filter after the results have been ordered by  $\mu(p)$ . Scores are calculated using the *weighted sum* and *sum distinct* and give a measure of performance relative to the baseline.

The table shows the average increase across all test sets. However, since not all the test sets contain equal numbers of queries, the results are skewed in favour of those metrics which perform well on the queries in the smaller test sets. A weighted average is also given, in which the averages are given across the set of possible queries. Unfortunately, the results are then skewed in favour of the metrics which perform well on the queries in the *larger* test sets.

The results show that, on average, any of the three metrics improve on the baseline result on over 75% of the test corpora. By considering potential navigation, trails can be discovered with more content relevant to the user's information need, hence raising the assigned scores. Kleinberg's hub measure appears the strongest of the metrics overall, with the Potential Gain and Outdegree both performing better on certain test sets. Overall, it is possible to conclude only that the metrics all represent useful functions but that the magnitude of the improvement is small, and that tests must be conducted relative to the expected queries and data for optimal results.

In certain circumstances, more effective results can be achieved by a weighting of the IR score,  $\mu(p)$ , and the link-analysis metrics. Figure 4.10 shows the average relative increase in trail scores achieved by combining the IR score in the remaining 10 ways listed in figure 4.6.

The results show that application of these techniques on the TREC data set leads to consistently negative results, despite positive results on the previous tests. In contrast, any of the metrics perform well on the JDK Javadocs. It is possible that this is because the Javadocs are well connected, whilst the TREC data is loosely connected, with a low average outdegree.

The combination of Potential Gain based upon Pinkerton's method,  $a\mu(p) + (1 - a)Pg(p)$  is a weighted average of  $\mu(p)$  and  $Pg(p)$  and leads to consistently negative results as the units of  $\mu(p)$  and  $Pg(p)$  differ substantially. Other experiments suggest these results hold for combinations with  $Out(p)$  and  $Hub(p)$ .

The overall results show that in order to obtain the best pages to start navigation, given a user goal, the set of pages  $p_1, p_2 \dots p_{max}$  should be selected that maximise the values of either  $\mu(p)Pg(p)$ ,  $\mu(p) \log Pg(p)$  or  $\mu(p) \log Out(p)$  unless specific knowledge of the corpus is given. For example, testing using TREC data should be performed using  $\mu(p), Pg(p)$ .

Test Set	Queries	Weighted Sum			Sum Distinct		
		Pg	Out	Hub	Pg	Out	Hub
UCLCS	26	3.55%	-2.05%	3.60%	2.43%	-3.49%	3.60%
TREC	50	3.10%	1.37%	2.77%	3.57%	1.25%	3.11%
SCSIS	151	1.13%	2.22%	1.57%	0.62%	1.52%	1.07%
JDK/Classes	2664	0.78%	1.12%	0.59%	1.08%	1.28%	0.90%
JDK/Misc	23	-0.66%	-2.41%	-1.13%	0.10%	-0.76%	0.15%
BBK	12698	0.70%	0.65%	0.99%	0.69%	0.64%	1.00%
DBLP/Docs	100	0.09%	0.75%	-0.29%	0.44%	1.36%	-0.32%
DBLP/Authors	100	-0.26%	0.55%	0.17%	-0.02%	0.17%	-0.10%
Corpus Avg.		1.05%	0.27%	1.03%	1.11%	0.25%	1.18%
Query Avg.		0.71%	0.73%	0.92%	0.76%	0.75%	0.98%

Figure 4.9: Percentage increases in trail score achieved by sorting by the given metrics in preference to  $Found(p)$  after previous sorted by the number of keywords then by the relevance  $\mu(p)$ . The two averages show the mean values taken over all *corpora* or webcases and over all *queries*.

A limited analysis was performed of starting-point selection considering only the single best trail returned. The findings supported the conclusions drawn here without significant new developments.

	Test Set	Queries	$\mu(p)Pg(p)$	$\mu(p)Out(p)$	$\mu(p)Hub(p)$	$\mu(p)\log Pg(p)$	$\mu(p)\log Out(p)$	$\mu(p)\log Hub(p)$	$\log \mu(p)Pg(p)$	$\log \mu(p)Out(p)$	$\log \mu(p)Hub(p)$	$out(p) + (1 - out(p))Pg(p)$	
Weighted Sum	UCLCS	26	7.26%	6.42%	-15.48%	8.15%	2.05%	-22.14%	-2.04%	-0.01%	-24.12%	-42.74%	
	TREC	50	-12.25%	-12.84%	-39.32%	-10.65%	0.28%	-41.35%	-22.32%	-8.24%	-37.32%	-51.86%	
	SCSIS	151	-0.85%	1.53%	-13.98%	-0.42%	1.90%	-14.27%	-3.55%	0.67%	-13.58%	-21.79%	
	JDK/Classes	2 664	9.73%	9.72%	8.28%	9.91%	10.75%	8.45%	6.74%	10.40%	7.54%	0.11%	
	JDK/Misc	23	3.40%	6.07%	-1.31%	3.92%	11.40%	0.39%	-0.73%	13.03%	-2.64%	-11.20%	
	BBK	12 698	6.88%	3.54%	-27.79%	6.83%	6.82%	-28.27%					-29.58%
	DBLP/Docs	100	-0.03%	-2.19%	-2.96%	-0.30%	-0.14%	-3.50%	-1.86%	-3.34%	-3.50%	-6.86%	
	DBLP/Authors	100	3.09%	-5.02%	-3.36%	3.09%	-4.71%	-3.69%	2.37%	-6.02%	-3.00%	-11.80%	
	Corpus Avg.		2.15%	0.90%	-11.99%	2.56%	3.54%	-13.05%	-3.06%	0.93%	-10.95%	-21.96%	
	Query Avg.		7.16%	4.43%	-21.25%	7.15%	7.30%	-21.63%	1.03%	1.69%	0.94%	-24.31%	
Sum Unique	UCLCS	26	5.99%	3.63%	-13.87%	7.22%	-0.25%	-20.98%	-2.20%	-2.94%	-21.41%	-45.15%	
	TREC	50	-13.39%	-13.32%	-37.87%	-12.11%	-4.23%	-38.88%	-21.02%	-9.49%	-34.50%	-49.59%	
	SCSIS	151	-5.67%	-3.90%	-18.58%	-5.57%	-3.64%	-18.53%	-8.39%	-4.79%	-18.46%	-26.27%	
	JDK/Classes	2 664	10.04%	9.55%	7.99%	10.18%	10.72%	8.16%	5.48%	10.39%	6.75%	-5.77%	
	JDK/Misc	23	5.36%	3.26%	0.15%	5.25%	10.35%	0.35%	-0.03%	9.59%	-0.93%	-13.54%	
	BBK	12 698	4.11%	-1.14%	-29.32%	3.95%	1.54%	-29.69%				-29.71%	
	DBLP/Docs	100	-1.48%	-0.83%	-3.28%	-1.76%	-0.48%	-4.07%	-3.26%	-0.93%	-4.05%	-2.61%	
	DBLP/Authors	100	-3.34%	-3.67%	-5.63%	-3.47%	-3.06%	-5.55%	-4.11%	-4.38%	-6.53%	-6.41%	
	Corpus Avg.		0.20%	-0.80%	-12.55%	0.46%	1.37%	-13.65%	-4.79%	-0.36%	-11.30%	-22.38%	
	Query Avg.		4.88%	0.60%	-22.57%	4.78%	2.98%	-22.86%	0.73%	1.65%	0.75%	-25.39%	

Figure 4.10: Shows the increase in trail scores achieved by using various measures for starting point selection.



## 4.7 Concluding Remarks and Future Work

This chapter has described a metric for expressing the utility of a page in terms of the potential navigation paths available from it. By combining the IR score with the potential gain better starting points can be selected from which to run the Best Trail algorithm. The experiments conducted have also shown that Kleinberg's hub metric and the outdegree of the pages in question can also be valuable for this task in certain situations.

However, to fully test the effectiveness, an evaluation is required using user judgements of whether the individual pages selected are more useful. Such an experiment could be performed through a user-study or via TREC-style, corpus-based measurement. If the findings of such experiments are positive, the Potential Gain will have far greater impact.

### 4.7.1 Query Specific Potential Gain

Some extensions of the Potential Gain metric are possible. For example, it is possible to compute a query specific version of Potential Gain, analogous to the query specific and topic specific PageRanks (Richardson and Domingos 2002; Haveliwala 2002). The fraction of trails,  $R_d$  to a depth,  $d$  from a node  $n$  is given by

$$R_d(n) = \sum_{y \in \text{Out}(n)} \frac{R_{d-1}(y)}{\sum_{j \in N} R_{d-1}(j)}$$

where  $N$  is the set of nodes in the Web graph. If, this is initialized with the page relevances,  $R_0(n) = \mu(n)$ , then a query specific Potential Gain measure can be gained in terms of the relevant pages reachability with paths of given lengths. If the relevance values are normalised between 0 and 1, then the formula for Potential Gain gives an upper bound for the query sensitive measure.

### 4.7.2 Multi-Metric Combinations

It is also possible to combine some or all of the metrics shown in many ways. Unfortunately, it is impossible to exhaustively test all combinations. However, it may be possible to use a GA to compute optimum combinations using various metrics. For example, the starting points could be selected using a combination such as:

$$\mu(p)^\alpha Pg(p)^\beta Gr(p)^\gamma Hub(p)^\delta Auth(p)^\epsilon Foley(p)^\zeta PR(p)^\eta$$

where  $\alpha \dots \eta$  are learned by the GA and  $Foley(p)$  refers to Mukherjea and Foley's metric for landmark nodes (Mukherjea and Foley 1995). Of course, as more metrics are combined, the perceived improvement in quality lessens and the cost of computation increases, both when generating the webcase data and during query time.

### 4.7.3 Site-based Potential Gain

It has been shown how potential gain can be computed for Web *pages*. It is also possible to compute the potential gain of a Web *site*. In this instance, sites are taken as nodes and a

link is added to the graph wherever there exists any two pages on different sites with a link between them. Such an approach would be useful for finding hubs and directories (such as Yahoo! or the ODP) for use with global search systems. If the Potential Gain was combined with a measure of relevance, topic specific hubs could also be identified.

## Chapter 5

# Architecture of a Navigation Engine

Architecture is the art of how to waste space.

Philip Johnson

Remember that happiness is a way of travel – not a destination.

Roy M. Goodman

## 5.1 Introduction

This chapter presents a description of a Web site *navigation engine*. The navigation engine is a complete system for trail-based IR, combining work in the fields of search engines, IR, graph representation and design patterns. It is termed a navigation engine as it is designed to solve the *navigation problem*. As discussed in chapter 2, the navigation problem is distinct but closely related to the problem of resource discovery. Similarly, the navigation engine that has been developed is closely related to a conventional search engine, but with an added emphasis on showing results in context and providing aids for future navigation.

Problems such as file-type recognition, index creation and document summarization are relevant to conventional search engines. The need to maintain efficient access to the graph structure is a problem faced in the development of other engines. For example, those which utilise the concepts of information units (Li, Candan, Vu, and Agrawal 2001), or variants of the HITS (Kleinberg 1998; Kleinberg, Gibson, and Raghavan 1998) and SALSA (Lempel and Moran 2000) metrics. A similar problem, also faced by such engines is the need to provide relevance scores for documents in close proximity to the best nodes, but which themselves have a low (or zero) relevance scores.

The rest of this chapter is organized as follows:

**Section 5.2** provides a brief overview of the main components of the navigation engine. These are the robot, parsers, index builder, query engine, trail engine and NavSearch UI.

**Section 5.3** discusses the creation of the files required to support the IR and trail generation algorithms.

**Section 5.4** provides details of a flexible architecture for supporting multiple trail discovery algorithms and optional index creation options. Several algorithms have been testing using this architecture including the Best Trail algorithm.

**Section 5.5** discusses the advanced query syntax and explains how the various features are supported. The syntax is based upon analysis of the features supported by AltaVista, Fast and Google.

**Section 5.6** discusses options for supporting PDF, Postscript and other non-HTML file types. All the files supported are converted into HTML, XML or plain text.

**Section 5.7** discusses a simple algorithm for computing context-sensitive web page summaries. The summaries offer noticeable advantages over those produced by most search engines and can be computed quickly and efficiently.

**Section 5.8** concludes with directions for future work.

User interface options are discussed in the following chapter which describes how this technology is applied to finding and displaying trails on Web sites. Issues related to scalability such as distributed indexes, distributed webcases, and merging of results from multiple sources are also discussed in the next chapter as these relate strongly to Web-scale data sets. The current single-machine architecture will happily scale to several million pages.

## 5.2 Top level Overview

Figure 5.1 shows the navigation engine architecture. The main components are :

- The *robot* or *crawler* which downloads Web pages.
- The *parsers* or *processors* which extract keywords, links and metadata from HTML, XML and plain text files.
- The *index builder*, which performs a set of *post-processing* operations to transform the processors output into usable *webcases*.
- The *query engine*, which assesses the relevance of pages to the users' queries.
- The *trail engine* which implements the Best Trail algorithm.
- The *NavSearch* user interface for displaying the results.

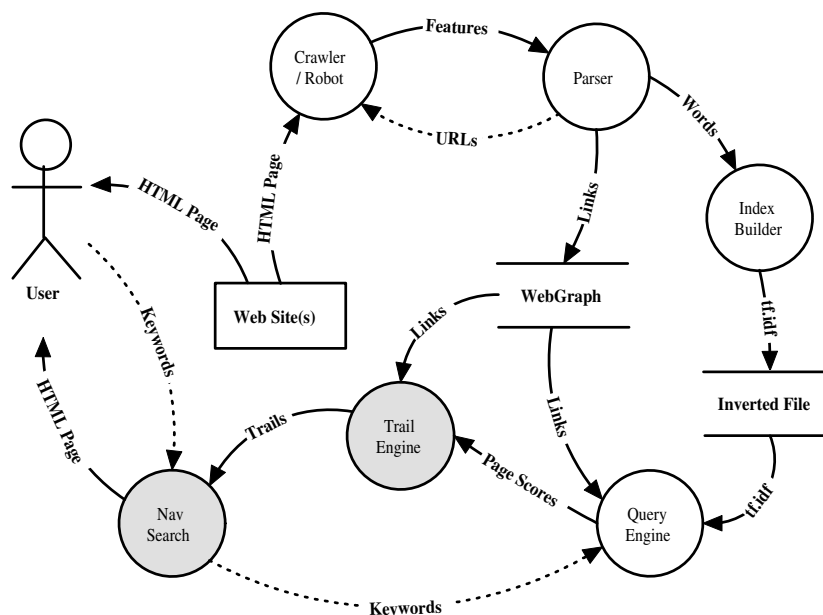


Figure 5.1: The navigation engine architecture. Unshaded circles indicate components common to search engines; shaded circles indicate components specific to the navigation engine; boxes denote external entities; and open-ended boxes denote internal data stores. Solid arrows represent data flow and dotted arrows represent flows of important requests for information (URLs and Queries).

The navigation engine uses an extensible, Java-based robot ideal for controlled crawls of Web sites (Shaw 1998; Wheeldon 1999; Skene 2001). A larger, faster, distributed robot is the Mercator robot (Najork and Heydon 2001; Heydon and Najork 1999a). Many of the problems faced during the development of the robot have already been discussed with respect

to Mercator (Heydon and Najork 1999b). The strategies used by Mercator to deal with the various problems could be applied in this case. These strategies include methods for dealing with queue distribution (as noted in section 2.4, and the re-implementation of the URL, HTTP and text handling sections of the Java class libraries.

The processors are components controlled by the robot which provide parsing facilities for HTML, XML and plain text. Each component has a Common Object Request Broker Architecture (CORBA) interface which allows distributed communication. The downloaded files are converted to abstract document representations known as **FeatureDocs**. These contain abstractions of identifiable *features* within the document, such as XML element tags, HTML heading or emphasis tags, URLs and hyperlinks.

The IndexBuilder performs a number of *post-process* operations on the data provided by the robot as **FeatureDocs** writing data to a number of files to form a *webcase*. The navigation engine takes these webcase files, and the query from the user and returns a set of trails in XML format. This XML is then converted using Cocoon<sup>1</sup> to form the NavSearch user interface with which the user interacts. Not shown in figure 5.1 is the summarization engine which takes a document representation and query and returns a short relevant extract. The summarization engine is called by the NavSearch UI generator, and is discussed in section 5.7.

---

<sup>1</sup> <http://xml.apache.org/cocoon/>

### 5.3 Webcases

A webcase is a set of files which can be used for answering queries. These files typically relate to a single Web site, domain or other logical structure. In order to be of use to the navigation engine, the files must be arranged so that they can satisfy the following functional dependencies. In order to achieve maximum speed, the files are indexed according to these definitions using either a Berkeley DB (Sleepycat Software 2001) file or an in-memory lookup table with offsets to a position accessed using a `RandomAccessFile`.

In memory references to webcase files are encapsulated by `Webcase` objects. These are extended *singletons* (Gamma, Helm, Johnson, and Vlissides 1995) which allow only one object per Java Virtual Machine (JVM) for each distinct webcase on disk. `Webcase` references are loaded on demand when first queried, as per the *Lazy Load* pattern (Fowler, Rice, Foemmel, Hieatt, Mee, and Stafford 2002).

$$\begin{aligned} URL &\rightarrow ID \\ ID &\rightarrow URL \end{aligned}$$

IDs are 32-bit integers assigned in sequence to each URL such that any two identical URLs will have an identical ID. It is necessary to convert URLs to IDs whenever a URL is specified as part of a query argument. This same operation is required frequently during index creation, during which the map will expand as more URLs are recognized. It is also necessary to convert internal IDs into URLs for returning to the user. The mappings between URLs and IDs are both stored in a Berkeley DB file, and accessed through a wrapper class which adds additional in-memory caching. This may also be used when checking equality between URLs from webcases with non-comparable IDs.

$$KEYWORD \rightarrow POSTINGS, IDF, [ID, Score]*$$

This represents the operation of an *inverted file* which returns posting lists for documents. Many variants return keyword position information with each score. In the navigation engine, this information is aggregated and stored in a Berkeley DB file. The normalized *tf.idf* scores are pre-computed in such a way that a document score can be computed via a simple sum (Salton and Buckley 1998). Scores are calculated for all documents. Ranks are calculated only for the top *n* documents to be used as starting points.

$$\begin{aligned} ID &\rightarrow AUTHOR, TITLE, FILETYPE, FILESIZE \\ ID &\rightarrow [KEYWORD, SCORE, SENT]* \end{aligned}$$

It is essential for the presentation of results that information relating to a given web page can be extracted quickly. Meta-data concerning each page is presented directly to the user. The document keywords and scores are passed to the summarizer which generates short extracts as described in section 5.7. The keyword scores are the same *tf.idf* scores as used in the inverted index. All data is stored in a random access file at keyed locations. The greatest bottleneck in the whole system is currently the misuse and required translation of Universal Character Set (UCS) Translation Format (UTF) data in this context.

$$ID \rightarrow ID^*$$

A *Web Graph* is a directed graph (digraph)  $G = (N, E)$  where each page in the corpus is represented by a node,  $n \in N$ , and each link between two pages by an edge,  $e \in E$ . A Web Graph implementation is required to return lists of URL Ids corresponding to inlinks and outlinks as indicated. A canonical representation of a graph  $G = (N, E)$  is a representation such that both  $N$  and  $E$  are ordered sets and contain no duplicate edges. Unless stated, it is assumed that all graphs dealt with are stored in a canonical format. These are stored in-memory using an adjacency list similar to the Link1 implementation (Randall, Stata, Wickremesinghe, and Wiener 2002).

$$\begin{aligned} &KEYWORD \rightarrow IDF \\ &ID \rightarrow \sqrt{(\sum tf.idf)} \\ &ID \rightarrow [CandidateTitle]^* \end{aligned}$$

These dependencies must be satisfied by temporary files at various stages during post-processing. The files in question are typically stored alongside those required for the webcase and are used only during construction of the webcase. In order to construct the index, the system must be able to obtain *idf* values for any given keyword, as well as normalization factors for the *tf.idf* scores. These normalization factors correspond to those suggested by Salton, as discussed in section sec:lit-ir.

Candidates titles are taken from the text on inlinks. A single link-text title is selected to describe a document when no other title can be found – i.e. when the document has not been indexed or contains no title or heading tags. This choice of title is motivated by a desire to avoid reliance on starting text or filenames, which may often be misleading. By selecting link-text titles, document are shown in the context of titles given in the pages which references them. Selecting a single title is a naive step to prevent query-time processing of the pages.



## 5.4 An Extensible Component Architecture

The `TrailAlgorithm` class acts as a *facade* (Gamma, Helm, Johnson, and Vlissides 1995) to any external client, hiding the complexity of the caching mechanisms, summarization, webcase handling and information retrieval options from the Graphical User Interface (GUI). It allows for a *strategy* (Gamma, Helm, Johnson, and Vlissides 1995) in which many different trail finding algorithms can operate, most notably the Best Trail algorithm, and acts as a *template* (Gamma, Helm, Johnson, and Vlissides 1995) for their implementation. The `ActiveWebcase` class also acts as facade to `TrailAlgorithm` subclasses by hiding the complexity of the `IEngine`, `TrailNode` construction and webcase handling. This separates the webcase management from the trail finding algorithms. Figures 5.2 and 5.3 show the structure of the classes concerned.

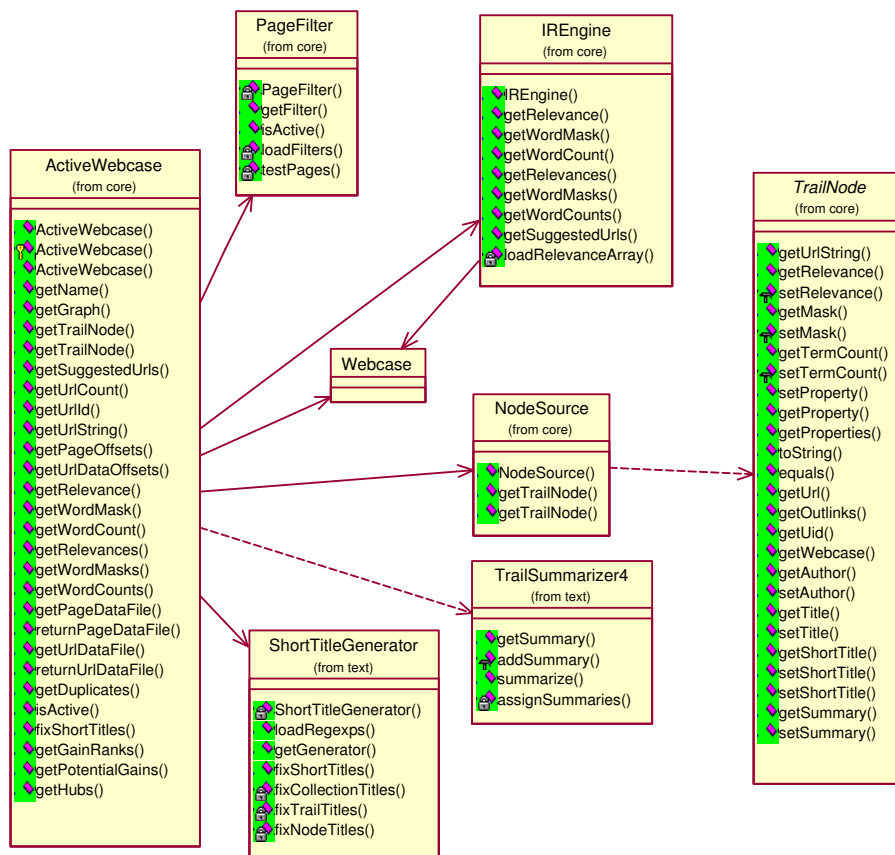


Figure 5.2: Behind the facade. Structure of classes in the `core` package

From this, the following assumptions can be made for each JVM, of which there should be only one during normal server operation:

1. Only one Web Graph will be present in memory, per webcase.

2. Only one copy of any Berkeley DB file will be open, per webcase.
3. Only one copy of any offsets file can be found in memory, per webcase.
4. Changes to any of these are going to impact on any other sessions, and any `TrailAlgorithm` subclass.

### 5.4.1 TrailAlgorithm subclasses

Figure 5.3 shows the position of two implemented algorithms and the infrastructure needed to write them. The two classes, `TrailAlgorithm` and `ActiveWebcase` are all that is needed alongside abstract data types to represent nodes, trails and queries.

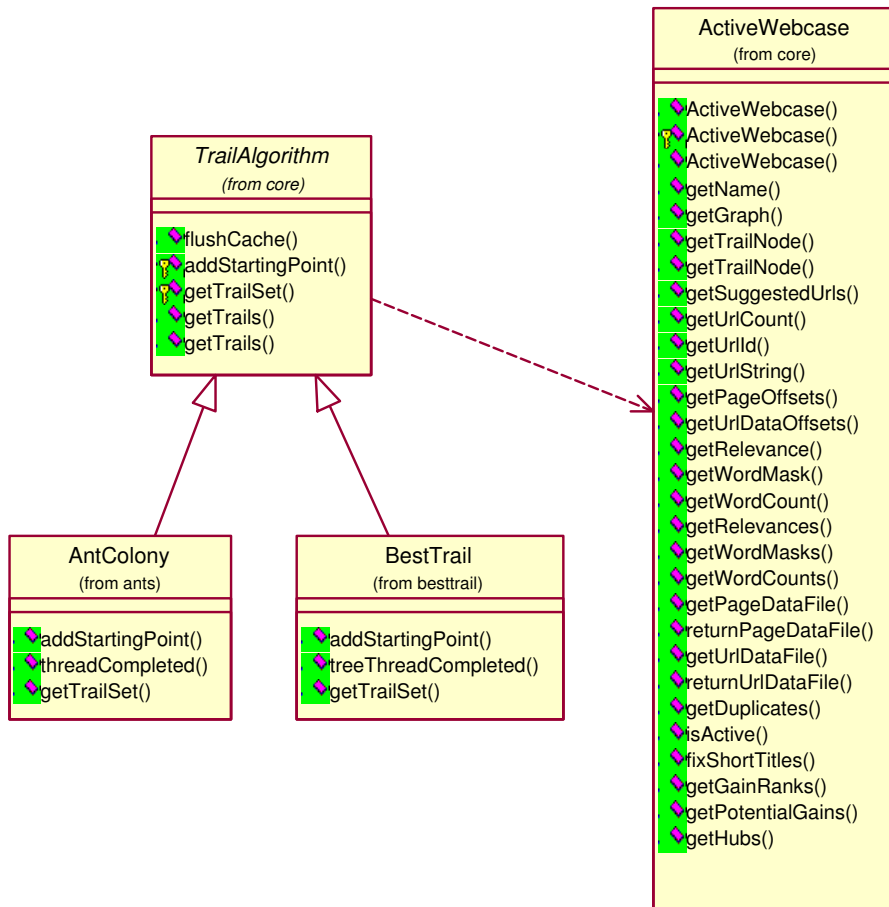


Figure 5.3: Two subclasses inherit from `TrailAlgorithm`, which relies heavily on `ActiveWebcase`. `TrailAlgorithm` and `ActiveWebcase` are the only two classes with which either `BestTrail` or `AntColony` need interact.

To demonstrate how this fits together, a simple example of a trail finding algorithm will be described which returns the starting point and most relevant outlook as a trail for each starting point.

First a class is created extending the template class.

```
public class BestLinkAlgorithm extends TrailAlgorithm
```

This means there is no longer any concern regarding:

1. Starting point selection.
2. Computing relevances (IR).
3. Loading node properties (title, etc.).
4. Document summarization.
5. Caching.

All these features are provided by the `TrailAlgorithm` and `ActiveWebcase` classes. Figure 5.4 shows the interaction of these classes, using `BestTrail2` as an example of a subclass, from the request of a `TrailSet` to the given answer.

A container is required for the completed Trails.

```
private TrailSet myTrailSet = new TrailSet();
```

Something must be done with the starting points, in this case obtaining the `WebGraph` and relevances and finding the best outlook.

```
public void addStartingPoint(ActiveWebcase webcase, int urlid) {
    // Get the WebGraph
    WebGraph graph = webcase.getGraph();
    // Get the outlooks of the starting point
    int[] outlooks = graph.getNode(urlid).getOutlinks();
    // Get the array of relevances for all nodes in the corpus.
    double[] relevances = webcase.getRelevances();
    // Find the best one.
    int bestUrl = -1; double bestRel = 0.0;
    for (int c1=0;c1<outlinks.length;c1++)
        if (relevances[outlinks[c1]]>bestRel)
        {
            bestUrl = c1;
            bestRel = relevances[outlinks[c1]];
        }
    // Construct a new Trail to hold the nodes
    Trail myTrail = new Trail();
```

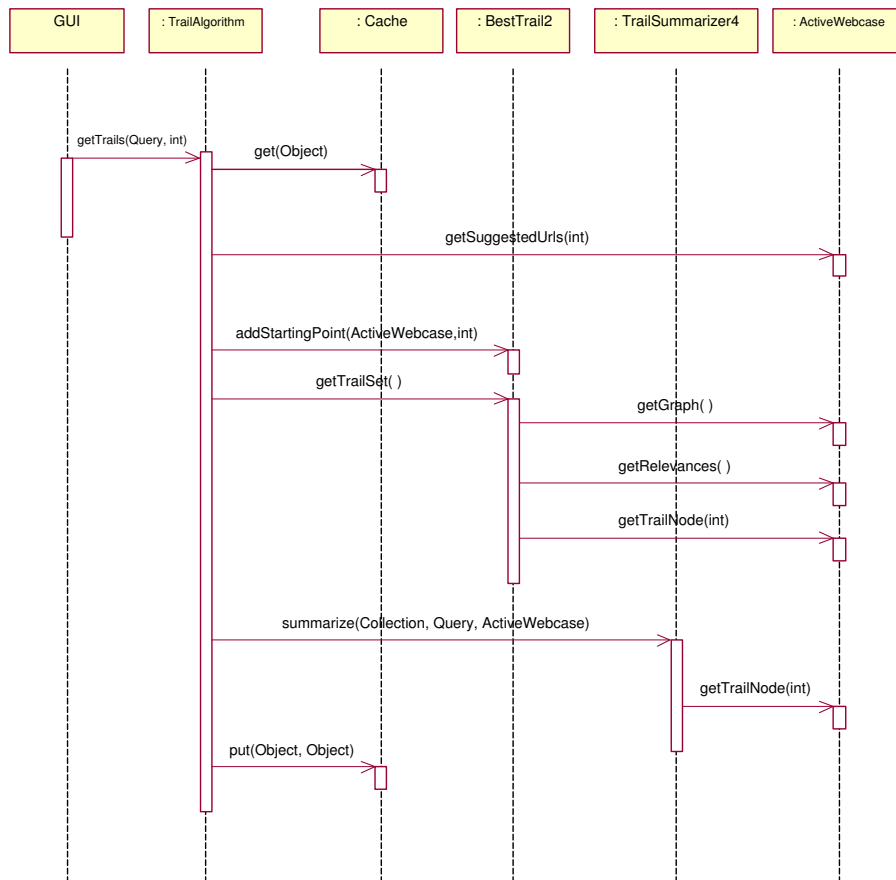


Figure 5.4: Behind the facade. Interaction of core classes.

```

    // Build the Trail
    myTrail.add(webcase.getTrailNode(urlid));
    myTrail.add(webcase.getTrailNode(bestUrl));
    // Add it to the TrailSet
    myTrailSet.add(myTrail);
}

```

Successive calls to this function will populate the `TrailSet` such that all that remains is to provide access to it with the following method:

```

public TrailSet getTrailSet()
{
    return myTrailSet;
}

```

This provides a complete implementation. It should be noted that there is scope here for many improvements. Possible errors have not been accounted for. For example, what if there are no outlinks? It should be noted that complex algorithms are better implemented by starting (or restarting) `Threads` for each given starting point so that they scale better on multi-processor machines. This can be achieved using thread-pooling techniques.

Five subclasses of this algorithm have been developed :

**BestTrail** implements the Best Trail algorithm as description in chapter 3. It uses a large pool of `TreeThreads`, each of which expand a given navigation tree from a specified starting point. The fixed limit on the size of this pool provides a fixed limit to the performance of the server and prevents overloading.

**AntColony** implements the Ant colony optimization system described in Dorigo, Maniezzo, and Colorni 1996. This has been used for comparative testing against the Best Trail algorithm.

**BestTrailTSP** was an attempt to extend the Best Trail algorithm to apply different functions and parameters, more suitable for finding solutions to TSP. It was consistently outperformed by the ant colony scheme in limited testing.

**StandardSearch** implements a conventional search engine list by directly returning the starting points.

**MultiGraphBestTrail** extends the Best Trail to provide support for trail finding across multiple graphs. This is discussed in further detail in chapter 8.

### 5.4.2 Post-Processing and Index Creation

The term “post-processing” covers all the activity between the initial data indexing (usually done by the robot) and the creation of a finished webcase. The construction of the index works using three operations - `readfeatures` (see figure 5.5), `mergefiles` (see figure 5.6),

and `buildtree` (see figure 5.7). Many subtle variations are possible within this framework, to permit additional weighting schemes. It should be noted, for example, that when writing the `keyword, urlid, score` tuples, the URL IDs written may refer not to the document in which the keyword was present but to some other document linked to by that document. This is why the merge operations are required subsequently.

**Algorithm 5 (Read-Features(*collection*))**

```

1. begin
2.   foreach document  $\in$  collection
3.     write document.metadata to urldata
4.     foreach keyword  $\in$  document
5.       write keyword, urlid, score tuples to queuej
6.       if queue is full
7.         sort queuej
8.         write distinct keyword, urlid, score to tempfilej
9.         inc j
10.      end if
11.    end foreach
12.  end foreach
13. end.

```

Figure 5.5: Algorithm for reading features and writing temporary files.

**Algorithm 6 (mergefiles(*collection*))**

```

beginforeach distinct keyword
  foreach distinct urlid
     $t_{score} = \sum score$ 
    write keyword, urlid, tscore tuples to tempfile0
     $n_{f_{urlid}} = n_{f_{urlid}} + t_{score}$ 
  end foreach
end foreach
end.

```

Figure 5.6: Algorithm to merge and aggregate files of `keyword, urlid, score` tuples.

Separate from the creation of the inverted index is the task of maintaining the Web graph. This is essential for trail detection, and the computation of ranking measures such as PageRank (Page, Brin, Motwani, and Winograd 1998) or Potential Gain.

It can be seen that there are many distinct processes in this algorithm – some of which need not wait until the crawl has finished before starting. An architecture has been defined in which many small operations or processes can be set in motion. Each process has a set of associated dependencies which must be fulfilled before it can begin. Once a set of these processes has been selected by the administrator, the `PostProcessHandler` will attempt to run each process in turn by resolving all dependencies until a successful conclusion is reached before running the process itself. Should a process fail, other processes will be tried, so as to

```

Algorithm 7 (Build-Tree(collection))
  begin
    foreach keywordintempfile
      foreach urlid
        normalize score w.r.t. nfurlid
      end foreach
      write keyword, [urlid, score]* to B-Tree
    end foreach
  end.

```

Figure 5.7: Algorithm to build inverted file in B-tree.

minimize the time lost due to failure. This has distinct similarities to the role performed by the Unix command `make -k` (Oram and Talbott 1993). The algorithm for performing this operation is given as algorithm 8. Future work might include wrapping these in `Ant` tasks which perform a similar function (Bailliez et al. 2002). This would allow parallelism with minimal code changes.

```

Algorithm 8 (RunProcess(P))
1. begin
2.   foreach  $D \in P.dependencies$ 
3.     if  $D.status = UNTOUCHED$ 
4.       RunProcess(D)
5.     end if
6.     if  $D.status = FAILED$ 
7.        $P.status = FAILED_DEPS$ 
7.     return
8.     end if
9.   end foreach
10.  P.process()
11. end.

```

Figure 5.8: Algorithm to run post-processing operations to generate webcase data.

All operations are defined in classes which extend the abstract template `PostProcessor`. There is a second abstract class `TransientPostProcessor` which defines a post-processing operation that must be performed (when required) every time the program is loaded. For example, this includes loaders for in-memory processing operations, such as `LoadWebGraph` and `LoadRootSum`. An operation performed by a class inheriting from `TransientPostProcessor` will not be associated with a flag file.

The `PostProcessHandler` is the most important class involved in the post-processing operation. It provides the run-time environment in which all operations are performed. When requested to do so, the `PostProcessHandler` will examine the list of available processes, and instantiate all the specified post-process operations. The `PostProcessHandler` is a *singleton*

in which every `PostProcessor` is associated with a webcase. The synchronization of the handler guarantees that no `PostProcessor` can be called more than once simultaneously for the same webcase, but allows many simultaneous threads to monitor the status information. If a second attempt is made to process the same webcase, the second `PostProcessor` will block waiting. The interaction between these classes is shown in figure 5.9.

The `PostProcessStatus` is used to keep track of the status of each post-process operations and records whether the operation has taken place, its success or failure, and (if it has failed) the cause of failure. A `MutablePostProcessStatus` subclass of `PostProcessStatus` provides an easy mechanism to set these properties, although on most occasions, using the provided constants of `FAILED`, `FAILED_DEP` (failed due to dependencies), `SUCCESS`, `DONE` (on a previous occasion), `UNTOUCHED`, (succeeded with) `WARNINGS` or `RUNNING` is sufficient.

Since all classes defining operations in the post-process system extend from either `PostProcessor` or `TransientPostProcessor`, only two examples of each type have been shown. The true interaction between `PostProcessor` subclasses is defined in terms of the dependencies between them, which are shown in 5.10. The `PostProcessHandler` will automatically determine a route through the dependency tree in order to create the needed webcase files. It should be noted that the dependancy tree is implied by the results of the `getRequirements()` method in `PostProcessor`, and is never defined explicitly. The following list gives a brief description of the function of each processor.

**WebcaseDirectory** Creates a directory structure to house the webcase files.

**CreateIdFactory** Creates a `UrlIdentifier` for the webcase, which maps URLs to IDs and vice-versa.

**ReadFeatures** Reads `FeatureDocs` from the repository, parses their content, and writes files out for subsequent processing.

**WebGraphFile** Constructs a webgraph file from outlink data created by `ReadFeatures`.

**CalcMaxUrl** Calculates how many pages are in the corpus.

**SortedTempFile** Sorts and aggregates the main keywords file, which contains tuples of the form  $(keyword, urlid, score)$ . These are merged such that only one tuple exists per  $(keyword, urlid)$  combination and all tuples for the same keyword are adjacent in the file. The scores are taken from the original  $tf$  values are combined into  $tf.idf$  values. This is done using a modified mergesort, which sacrifices a theoretical  $O(n^2)$  worst-case complexity for very low I/O costs.

**LoadRootSum** Loads the normalization factors (a byproduct of `SortedTempFile`) into memory.

**KeywordBTree** Combines the keywords file into a Berkeley DB B-Tree suitable for the navigation engine's use. This process includes normalizing each score with respect to the document containing it. The result is the inverted file.

**LinkTextTitlesDatabase** Creates a database file mapping URL IDs to potential titles, obtained from link text.



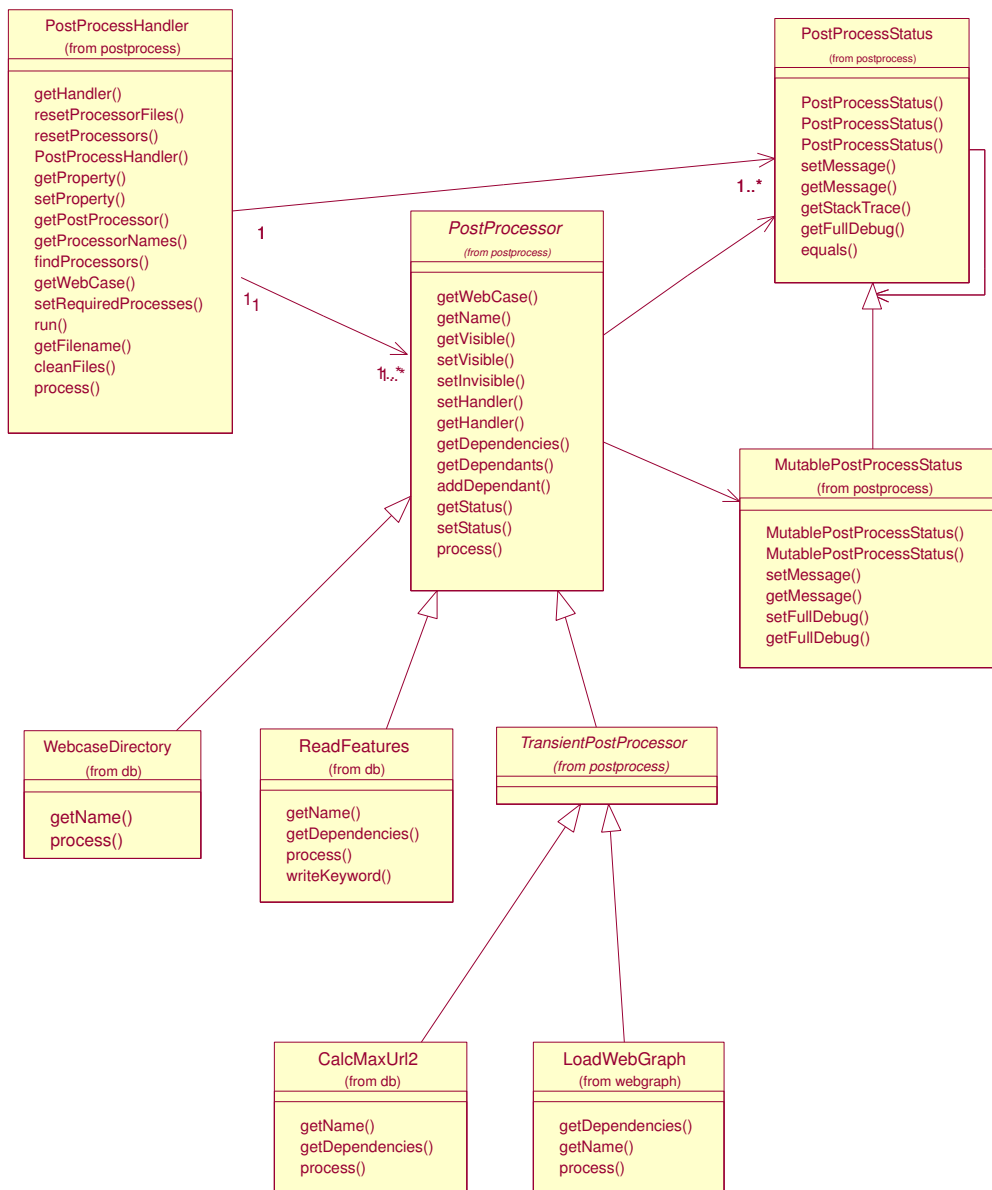


Figure 5.9: Structure of the post-process classes.

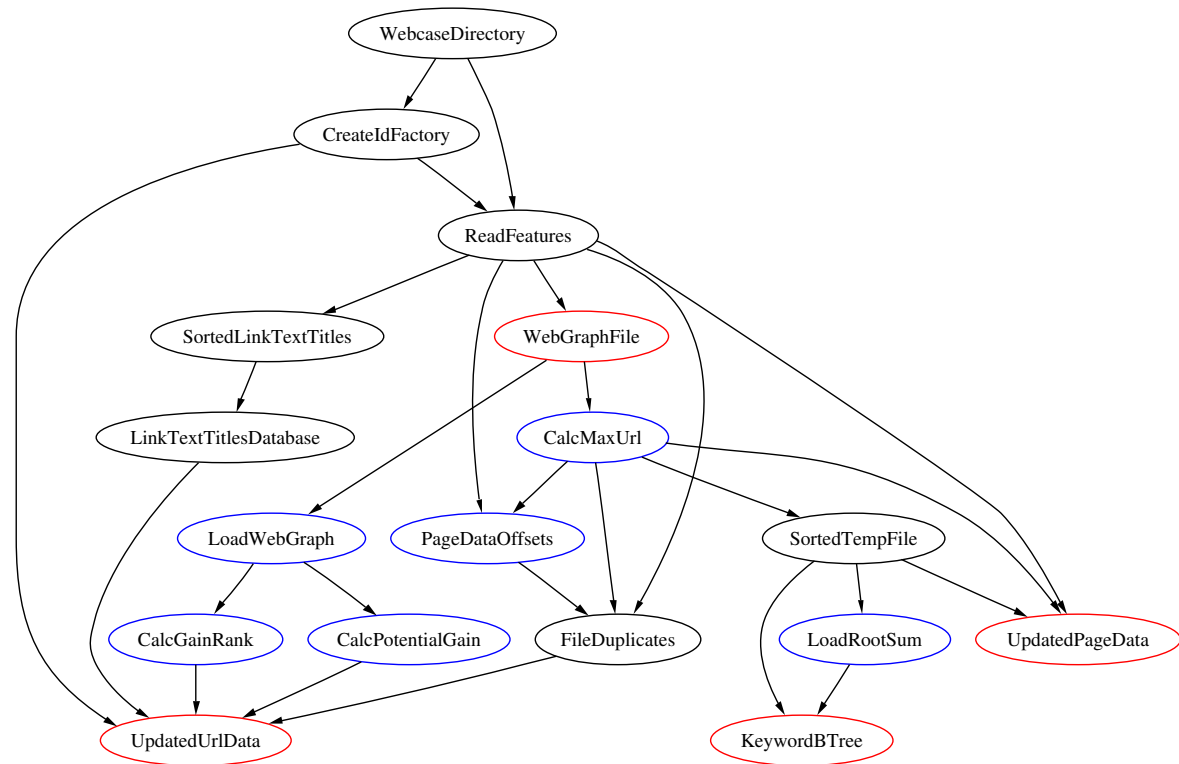


Figure 5.10: Dependencies between post-process operations. Blue ellipses denote transient operations. Red ellipses denote operations producing output request for the navigation engine to work. Typically it is these operations which must be selected by the administrator.

**SortedLinkTextTitles** Sorts the table of link-text titles generated as a by-product of the **ReadFeatures** process.

**UpdatedPageData** Updates the pagedata file (generated by **ReadFeatures**, and storing the document representations) to contains accurate *tf.idf* scores for terms.

**CalcGainRank** Calculates Gain Rank values for each node in the graph.

**CalcPotentialGain** Calculates Potential Gain values for each node in the graph.

**LoadWebGraph** Loads the Web Graph into memory, either from outlink data or a serialized WebGraph file.

**FileDuplicates** Identifies (exact) duplicate documents. This can and should be extended to near-duplicate detection as prescribed in Broder, Glassman, and Manasse 1997; Broder 1997; Broder 2000; Shivakumar and Garcia-Molina 1999.

**UpdatedUrlData** Updates the meta-data to contain titles, Potential Gain values, etc. which were not available when the file was created by **ReadFeatures**.

**PageDataOffsets** Loads the offsets to the pagedata file into memory.

## 5.5 Advanced Features

Most users of a search engine type short queries and do not use advanced features. However, there is a sizable minority who either use such features regularly or who require them occasionally. Advanced features are typically accessed either through a separate web form or via special commands. These commands are almost invariably of the form **option:parameter**. It is worthwhile reviewing the options supported by the major search engines before applying them to the navigation engine. Figure 5.11 shows the options supported by Google, AllTheWeb and Altavista's Web Search. The systems used for intranet search may differ slightly.

The common elements of search engine query syntax have been adapted and extended for the navigation engine. In addition to + and - symbols for word inclusion and exclusion, the following options may be supported :

**site** All pages are assigned a zero score if they are not within the specified site(s). This may be a domain or server name or a predefined name specific to each webcase or organization. The commands **host** and **domain** will perform the equivalent tasks.

**url** The given URL is used as the sole starting point. If many URLs are given, the thread-count is divided by the number of URLs. This is also the default action for a URL given without a command.

**link** All pages have zero score if they do not link to the given page.

**language** All pages are assigned a zero score if they do not contain some text in the specified language. RFC1766 defines a **Content-Language** header tag which can be used by authors to specify the page's language(s) (Alvestrand 1995). When this is not available, it is possible to make a best guess at a page's language by looking at the frequency of common words (Wood 1998). For example "the", "is", "a" and "when" would suggest an English text whereas "die", "ein", "oder" and "für" would suggest a German text.

**related** All pages are assigned scores determined (in part) by their relationship to a given page. This can be achieved by taking the  $n$  most significant words for the given page and substituting them into the query. Alternatively, a clustering algorithm could describe the proximity of pages in advance.

Of these, **site**, **url** and **link** are currently supported.

Command	Search Engine	Operation
host	AltaVista	Finds pages on a given server.
domain	AltaVista	Finds pages on servers within a given domain.
url	AltaVista	Finds pages with the given text in the URL of the page.
link	Google	Finds pages which link to a given URL.
link	AltaVista	Finds pages which link to a given URL or site. e.g. link:www.dcs.bbk.ac.uk matches links to any page on the SCSIS Web site
anchor	AltaVista	Search for pages with outlinks containing the given in anchor text.
image	AltaVista	Finds pages with images having a specific filename.
text	AltaVista	Finds pages that contain the specified text in any part of the page other than an image tag, link, or URL. Pages must contain word in the body of the text (i.e. ignore linktext/meta tags).
title	AltaVista	Finds pages that contain the specified word or phrase in the page title
applet	AltaVista	Finds specified Java applets.
like	AltaVista	Finds pages similar to or related to the specified URL.
site	Google	Search within a site. Similar to Altavista's <b>domain</b> option.
related	Google	Equivalent to Altavista's <b>like</b> option.
allinurl	Google	Equivalent to Altavista's <b>url</b> option.
allintitle	Google	Equivalent to Altavista's <b>title</b> option
url.tld	AllTheWeb	Finds pages within a specified top level domain (TLD). e.g. url.tld:fr will find pages from France.
url.host	AllTheWeb	Equivalent to Altavista's <b>domain</b> option.
link.all	AllTheWeb	Equivalent to Altavista's <b>link</b> option.
normal.title	AllTheWeb	Equivalent to Altavista's <b>title</b> option.
url.all	AllTheWeb	Equivalent to Altavista's <b>url</b> option.
normal.titlehead	AllTheWeb	Finds pages with the specified word or phrase in the title or in its head content.
url.domain	AllTheWeb	Finds pages with the specified word or phrase anywhere in the domain name.
link.extension	AllTheWeb	Finds pages linking to files with a given extension. e.g link.extension:jpeg will find pages that contain .jpeg images.

Figure 5.11: Advanced query syntax for search engines.

## 5.6 Improving File-Type Recognition

It has recently become common practice for search engines to index content from pages other than those written in HTML or plain text. This section attempts to describe techniques for dealing with some of these types of files.

This is achieved using filters – components which transform documents of one MIME type to those of another. These are written with a CORBA interface and can be accessed from anywhere on the local network. Thus they can make use of operating system specific applications on multiple systems. They all follow the same basic algorithm, which takes a resource description  $R$ , which encapsulates the data and metadata elements and returns a modified resource description with the new text or HTML representation of the data and an updated MIME-type.

This technique of using filters is not new. Verity claim to index over 250 document formats using a similar method (Ragghavan 2001). However, since there has been no published work describing either the algorithm or the required filter programs, it is worthy of discussion.

### Algorithm 9 (Filter( $R$ ))

1. **begin**
2.   **if**  $R.mimetype \in accepted\_mime\_types$
3.      $tempfile1 \leftarrow R.data$
4.     **run** external program
5.      $R.data \leftarrow tempfile2$
6.     **return**  $R$
7.   **end if**
8. **end.**

Figure 5.12: Algorithm for filtering documents from one mime-type to another using an external program.

The following file types can be supported using this technique <sup>2</sup>:

**PostScript and PDF** To handle these formats, two Unix programs, `pstotext` and `pdftotext` are used. It is almost certain that these programs also provide the basis for Google's PDF and Postscript support as the output is identical to that shown in the Google cache (including the bugs!). The PDF or Postscript is converted to plain text which is then parsed as normal. In order to obtain link information, the plain text parser recognizes URLs and will treat them as links where appropriate. However, attempts to recognize local URLs or e-mail addresses in plain text files often identify false positives.

**Microsoft Office** Microsoft Word documents can be converted to HTML using the program `wvHtml` by Dom Lachowicz. The programs `xlHtml` and `pptHtml`, both written by Steve Grubb provide the same facilities for Excel spreadsheets and Powerpoint files.

---

<sup>2</sup> Please note that if using the Sun's Hotspot JVM, it is advisable to compile all programs without debug due to a known issue with the process handling.

**ShockWave Flash** The utility `swfstrings` by Rainer Boehme is distributed as part of the `swftools` package and can be used in the same manner as the previous utilities.

**Archives** Archive files can contain compressed versions of many different files. There are three ways in which these can be handled. Firstly, a list of the files in the archive can be extracted in text format. This may seem a weak attempt, but it conserves space and enables important resource discovery questions to be answered, such as finding patches for missing files. The second way is to concatenate the text or HTML representations of all the files in the archive. The third is to generate a fake page with the archive listing and a servlet which will download the archive and return a specific file from within it. The text and HTML representations of the files within the archive can then be indexed making reference to this servlet.

The first two options are supported for Tar, Gzipped Tar (`.tar.gz`) and Zip archives and Redhat Package Manager (RPM) and Debian (Deb) packages. File lists can be obtained from these archives using the Unix commands shown in figure 5.13. Full data extraction is possible using similar techniques. Packages can also be installed using fake root filesystems on loopback devices if required.

Archive Type	Unix Command
Tar	<code>tar tvf filename.tar</code>
Gzipped Tar	<code>zcat filename.tar.gz   tar tv</code>
Gzipped Tar (Gnu)	<code>tar xvfz filename.tar</code>
Zip / PkZip / WinZip	<code>unzip -l filename.zip</code>
RPM (RedHat)	<code>rpm -qpl filename.rpm</code>
Deb (Debian)	<code>dpkg -c filename.deb</code>

Figure 5.13: Unix commands for extracting lists of files from archives.

## 5.7 Web Page Summaries

Search Engines typically display small summaries, descriptions or extracts of Web pages as part of their search results. The navigation engine described here is no different in this regard. Section 6.2 will show how such summaries are incorporated into each of the available user interfaces.

Descriptions are often given by authors in the meta tags of a site's pages, but these usually attempt to describe the full content of the page and, in doing so, fail to show the user's query terms in context. Simple extracts are often created by taking the first few sentences or terms of a document. These also fail to show the query terms in context. Research has shown that displaying terms in context improves the user's ability to discriminate between documents (Drori 2000; Tombros and Sanderson 1998). This is why more advanced search engines use dynamically created extracts to describe the document and its relevance to the user's query. An algorithm for computing such summaries is now described.

The summarizer uses a greedy algorithm which constructs summaries by combining sentences. Several assumptions were made during its development, which should first be justified:

1. Good summaries are created from important sentences. That summaries must contain key terms in order to describe the document is held as self-evident. The decision to include only complete sentences was made both to reduce the search space and to aid comprehension and is consistent with previous attempts at summarization (Luhn 1958; Edmunson 1969; Tombros 1997).
2. The importance of a sentence is directly proportional to the importance of its constituent terms. One could argue that the ordering of these terms has equal significance, in which case terms may be replaced by  $n$ -grams.
3. The importance of a term is directly proportional to the frequency with which that term occurs in the document. This is the same principle which motivated the  $tf$  measure, and has been used as a core principle in classic summarization algorithms (Luhn 1958).
4. The importance of a summary term in representing a document and allowing the user to discriminate between relevant and non-relevant documents is inversely proportional to the number of documents in the corpus which contain that term. This is the same principle which motivated the  $idf$  measure.
5. Sentences which repeat concepts found in other sentences are of little value. Other algorithms have been developed which attempt to reduce redundancy (Edmunson 1969). However, none of the algorithms used by search engines appear to remove redundant content.
6. Query terms are valuable and should be included in the summary (Drori 2000; Tombros and Sanderson 1998).



### 5.7.1 A Summarization Algorithm

The algorithm builds a summary by selecting sentences before arranging them in the order in which they appear in the original text. It could equally be based upon arbitrary groups of words, but doing so drastically increases the search space, and removes contextual information. Sentences are repeatedly selected until certain stopping conditions are met. For example, until the maximum length of the summary is reached. Sentences are selected based upon the average score for words within the sentences.

A common criticism of sentence-based summarization techniques is that they produce variable length summaries. This problem is solved by computing the summary length using different measures. The general framework of the algorithm allows the length to be specified in any convenient measure, such as the number of sentences, the number of words, the number of characters, the number of bytes or the number of pixels. In practice, the length is calculated using both the number of sentences and the number of characters.

The weight given to each term is proportional to the number of times the word appears in the document,  $tf$ , so that the sentence picked is representative of that document (Baeza-Yates and Ribeiro-Neto 1999). Increasing weights using link text terms and biasing those in specific HTML markup tags as described in chapter 3 helps this process (Cutler, Deng, Maniccam, and Meng 1999). A term's weight is also proportional to the number of times the term appears in the query. This implies that sentences selected are those most likely to be relevant to the user's information need. The term's weight is also inversely proportional to the number of times the term appears in the corpus  $idf$ , so that the sentence (and by extension, the summary) contains good discriminators and inversely proportional to the number of times that term has already appeared in the summary, so the minimum amount of information is repeated.

#### Algorithm 10 (Summarize( $D, M$ ))

```

1. begin
2.    $S \leftarrow split(D)$ 
3.   repeat
4.     foreach  $s \in S$  do
5.        $calcScore(s)$ 
6.     end foreach
7.      $s \leftarrow pickBestUnflagged()$ 
8.      $flag(s)$ 
9.      $summary.add(s)$ 
10.  until  $|summary| \geq M$ 
11.  return  $summary$ 
12.  sort  $summary$  by position
13. end.

```

Figure 5.14: Algorithm to Summarize Web Documents

The summarizer (algorithm 10) takes a document,  $D$ , and the maximum length of the summary,  $M$ . It takes time in the order  $O(|S|.M)$  where  $|S|$  is the number sentences in  $D$ . The

following auxillary functions are required by the algorithm:

*Split(D)* Takes a document,  $D$ , and returns it's component sentences.

*calcScore(s)* Takes a sentence,  $s$ , calculates its score based upon the terms within it, and stores this value for use with *pickBest()*. The weight attached to a sentence is given by

$$score(s) = \frac{1}{|s|} \sum_{x \in s} k_1^{qf_x} k_2^{tf_x} k_3^{df_x} k_4^{sf_x}$$

where  $tf_x$  is the frequency of term  $x$  in the sentence,  $qf_x$  is the frequency of term  $x$  in the query,  $df_x$  is the frequency of term  $x$  in the document collection,  $sf_x$  is the frequency of term  $x$  in the portion of the summary constructed so far,  $|s|$  is the length of  $s$  and  $k_1 > 1$ ,  $k_2 > 1$ ,  $k_3 < 1$  and  $k_4 < 1$  are constants. The values for  $sf_x$  and  $|s|$  start at zero and increase with each iteration of the algorithm, whilst the values for  $k_1$ ,  $k_2$ ,  $k_3$ ,  $k_4$ ,  $tf_x$ ,  $qf_x$  and  $df_x$  remain constant.

*flag(s)* flags the sentence  $s$  as being included in the sentence.

*pickBestUnflagged()* Returns the sentence with the highest score as previously calculated, from those which have not been previously included.

### 5.7.2 Examples

The following results show a comparison between the document summaries produced by this algorithm and those provided by leading search engines Google, Teoma and Ask Jeeves, Lycos and AllTheWeb. Altavista failed to find any of the documents in question with respect to our queries.

The first two documents are taken from a list of those found on the World Wide Web Conference site<sup>3</sup> with the query “precision recall”. The first document is a paper from the refereed papers track, entitled “Template Detection via Data Mining and its Applications” (Bar-Yossef and Rajagopalan 2002)<sup>4</sup>. The query terms have been highlighted in each case. Teoma and Google both highlight query terms in their search result pages – the others do not.

**Lycos** suggested “We formulate and propose the template detection problem, and suggest a practical solution for it based on counting frequent item sets. We show that the use of templates is . . .” which is simply the start of the body text and doesn't show any of the query terms or how the page relates to the IR evaluation metrics concerned.

**AllTheWeb** suggested “. . .increases in **precision** at all levels of **recall**. Categories and . . . improvements in **precision** across a wide range of **recall** values. We feel . . . consequently, reduce **precision**. In almost all . . . improvements in **precision** at all **recall** levels across . . .” which shows clearly shows the query terms in context, but the incomplete sentences leave the user to fill in the missing words.

<sup>3</sup> <http://www2002.org/>

<sup>4</sup> <http://www2002.org/CDROM/refereed/579/>

**Teoma** highlights the key phrase “A large overlap indicates both high **precision** and high **recall**”. Ask Jeeves uses the same engine and returned the same results, but strangely without the keyword highlighting!

**Google** identifies the same phrase – and repeats it. “. . . A large overlap indicates both high **precision** and high **recall**. The . . . A large overlap indicates both high **precision** and high **recall**. The . . .”

In contrast the proposed summarization algorithm returns “We show that applying our method results in significant improvements in **precision** across a wide range of **recall** values. . . The Yahoo! . . . All of them belong to a template of the site. . . A large overlap indicates both high **precision** and high **recall**.” which shows two distinct uses of the same terms.

The second document is a poster entitled “Development and Evaluation of the WithAir Mobile Search Engine” (Kawai, Akamine, Kida, Matsuda, and Fukushima 2002)<sup>5</sup>. The summarization algorithm returned the “The **precision** and **recall** of regional information classification were calculated in the same way as those of the focused crawler. Table 1: Performance of i-mode focused crawler Gathered i-mode pages **Precision Recall** 1,300,000 99% (1246/1251)” compared to Google’s result of “. . . Table 1: Performance of i-mode focused crawler Gathered i-mode pages **Precision Recall** 1,300,000 99% (1246/1251) 88% (1169/1326) Table 2: Performance of . . .” which highlights one use of the terms, but not the second.

The extract generated from the DTI page of a speech made by Lord Sainsbury of Turville to the Asian Technology Markets Conference<sup>6</sup> with respect to the query “american patents” was “By 1997 the number of doctoral science degrees had risen in Asian universities to 18.5 thousand and 5.5 thousand in **American** universities . . . In the USA it accounted for 20% of **patents** granted behind the USA but ahead of both Germany and the UK at” which compares with:

**Google** whose description “. . . In 1989 the number of **American** visas (temporary work permits) issued to qualified . . . In the USA it accounted for 20% of **patents** granted, behind the USA but ahead . . .” is very similar but highlights a different instance of the term “american”.

**Lycos** which again returns the start of the document with no regard for context or query terms: “Mr Chairman, ladies and gentlemen. Thank you for your kind invitation to deliver the keynote address here, at Chatham House, this morning. The Royal Institute of International . . .”

### 5.7.3 Implementation

The algorithm is implemented within a class `SummarizerThread`. Multiple documents can thus be summarized in a multi-threaded environment, controlled by a `TrailSummarizer` which iterates through a trailset and assigns summaries as appropriate. The method `generateSummary` implements the algorithm described above, whilst the `updateScores` method refreshes the scores as given in equation 5.7.1.

---

<sup>5</sup> <http://www2002.org/CDROM/poster/102.pdf>

<sup>6</sup> <http://www.dti.gov.uk/ministers/archived/sainsbury210301.html>

As currently implemented,  $tf_x$ ,  $qf_x$  and  $df_x$  have been replaced with the formulae given by Salton and Buckley (Salton and Buckley 1998).  $\log tf$  appears to perform better than  $tf$  and the document normalization is constant for all keywords in the document so does not affect the results.

An interface `MarkedText` describes text with highlighted terms, which can be converted to `Strings`, HTML or XML. Summaries are returned as `SummaryText` objects, which implement this interface. The `TrailSummarizer` class hides this by providing a single `summarize` method which controls the thread pooling and allows multiple summaries to be computed in parallel. Document terms are loaded from a file at offsets which are specified in a separate file and loaded a-priori.

#### 5.7.4 Performance Analysis

Experiments were performed to analyse the effects of increasing resources on the time taken to summarize the documents for trails returned in response to a set of queries taken from the query logs for the Birkbeck Web site.

Increasing the number of threads to a number greater than the number of CPUs improves the overall speed of operation. Further increases yield little improvement, as can be seen in figures 5.15 and 5.16. Increasing the number of open file handles on the `pagedata` file has a similar effect, but increasing the number of file handles to excessive numbers can noticeably worsen performance, as can be seen from figure 5.17.

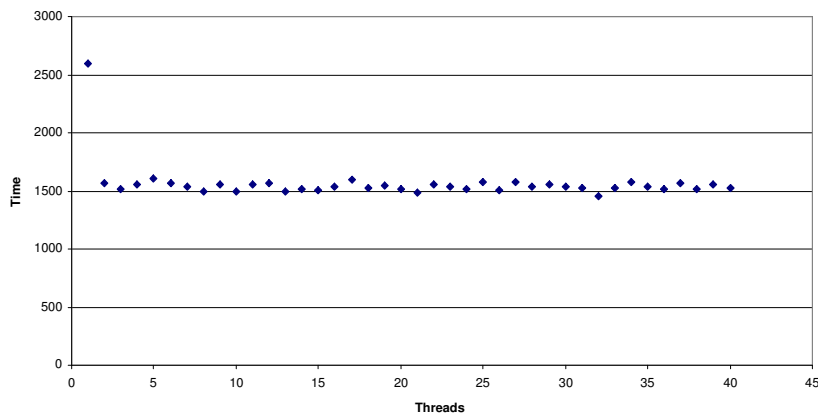


Figure 5.15: Time taken to load and generate an average summary, given a certain number of threads.

Increasing the number of documents cached does not seem to have the expected impact, as can be seen from figure 5.18 This result needs further investigation. However, if the result is confirmed by subsequent investigation, then it supports the conclusion that memory resources are better allocated to other tasks, such as increasing the graph sizes in a distributed environment, increasing search result caches, increasing inverted file caches or increasing `UrlIdentifier` caches.

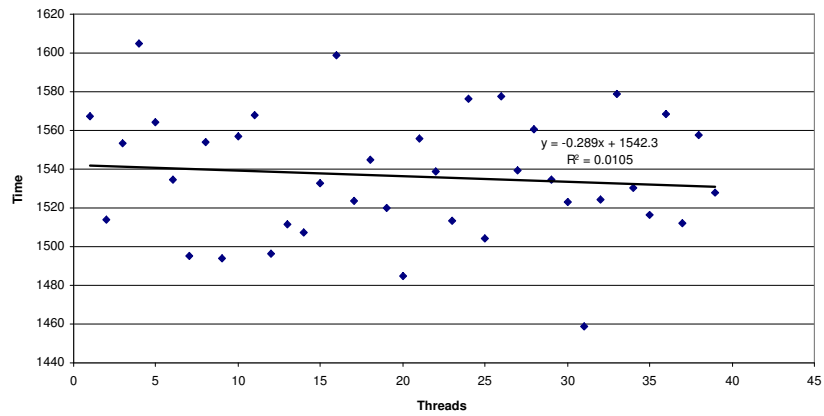


Figure 5.16: Time taken to load and generate an average summary, given a certain number of threads, where the number of threads is greater than one.

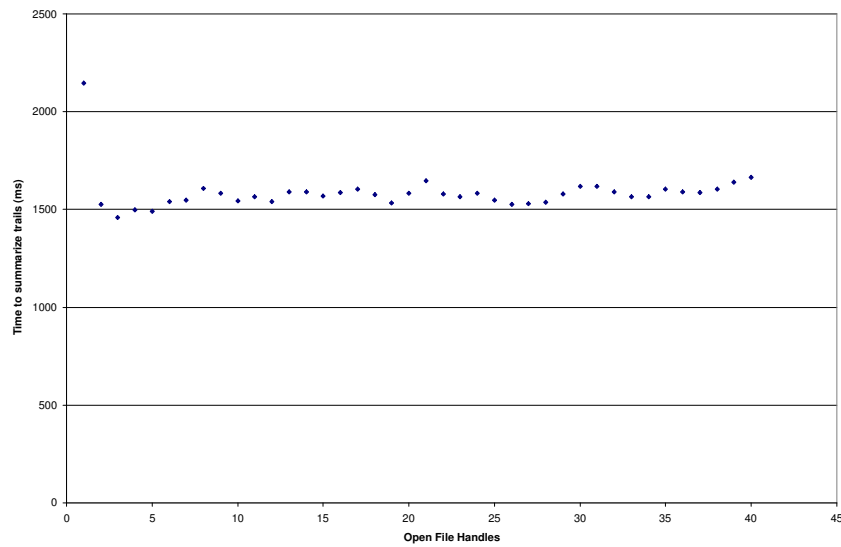


Figure 5.17: Time taken to load and generate an average summary, given a certain number of open file handles.

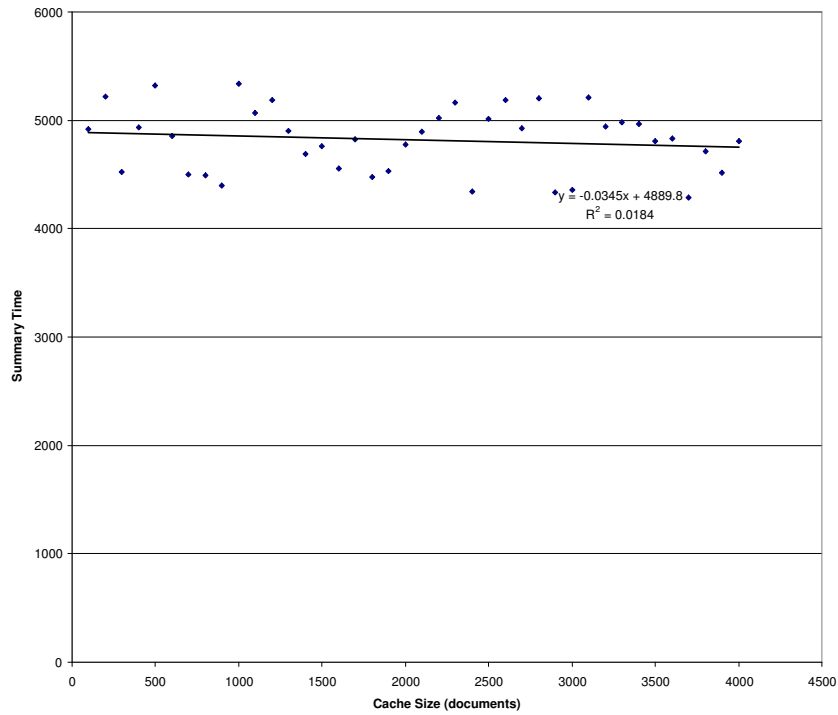


Figure 5.18: Time taken to load and generate an average summary, given a certain number of documents cached.

Experiments were also made to confirm the hypothesis that for a constant set of parameters, the algorithm works in linear time in practice. Figure 5.19 shows that there is a strong correlation between the size of the document (number of words) and the time taken to load the document. Surprisingly, this is not the same for the time taken to generate the summaries, as can be seen from figure 5.20. For a document of a given size, the time taken is distributed normally. The hypothesis can therefore be rejected. External factors have a greater role than the size of the document in real-life situations.

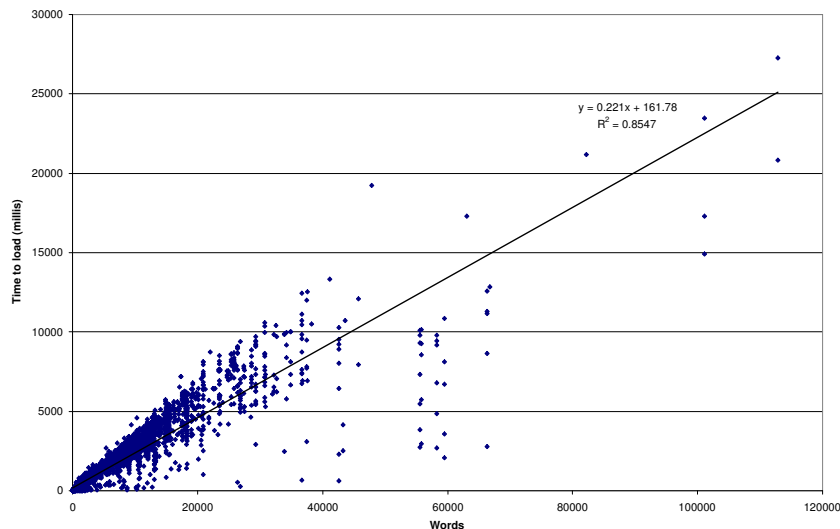


Figure 5.19: The time taken to load a document correlates strongly with its length.

Experiments were then performed to investigate the overall speed of the implementation and to identify any bottlenecks in the system. The time taken to summarize each document was recorded throughout a single run. The most commonly summarized page was the Birkbeck Library A-Z Index<sup>7</sup>. This is not particularly surprising as it is a major hub from which documents matching all categories of information need can be reached very quickly. The mode, median and mean times to summarize documents were 1, 2 and 18.4 milliseconds respectively, with the longest document taking 5.6 seconds to summarize. It is hypothesized that this is again due to garbage collection pauses. Overall the summarization was found to be at an acceptable speed for supporting the sub-second return of results.

In comparison, the times taken to load the documents were disturbingly high. Again there was a skewed distribution in the times taken, as would be expected when the distribution of document sizes is skewed. In the case of times to load documents, a power law distribution was expected to match the previously recorded power law distribution in document sizes. However, the data best fits an exponential distribution, as can be seen from figure 5.21. The longest time taken to load a single document was 27256 milliseconds. The mode, median and mean times to load the documents were 31, 243 and 939.6 milliseconds, respectively. This is completely unacceptable for sub-second response times. Further investigation showed over 80% of time to be spent in the JDK's `readUTF()` method. this method was re-implemented, removing

<sup>7</sup> <http://www.bbk.ac.uk/lib/atoz.html>

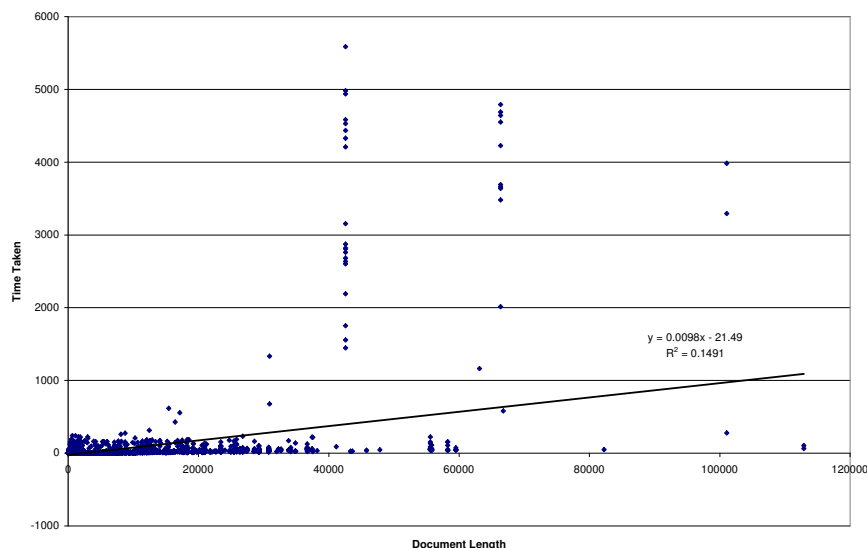


Figure 5.20: The time taken to generate a summary for a document does not correlate strongly with its length.

the `StringBuffer` usage. `StringBuffers` have already been identified as a major cause of performance problems in Java (Heydon and Najork 1999b). This improved performance, but not to an acceptable level. Overall, this highlights a poor choice of document representation, and a key area for future development.

### 5.7.5 Titles and Short Titles

Closely related to the problem of computing document summaries is the problem of generating short titles for page labels in nodes on displayed graphs or trees. The requirement is to produce a short title (say, 10 characters) which gives the user useful information about a web page. Labelling the nodes using the full title of the page often leads to a node occupying too much screen space. A rule based system has been developed to truncate titles, which is based upon regular expressions. Any string matching a given expression may be substituted by a new string. The following actions can be performed within this framework:

1. Removing any initial substring common to a node and its parent
2. A stop-list of words to remove (e.g. “the”, “a”, etc.)
3. Substituting in abbreviations and shortened forms (e.g. “approx.”)

The following actions might also be useful but are somewhat drastic, and cannot be performed with the current system:

1. Remove words according to their *tf.idf* values. Words with low *tf.idf* values are more likely to be good candidates for removal.



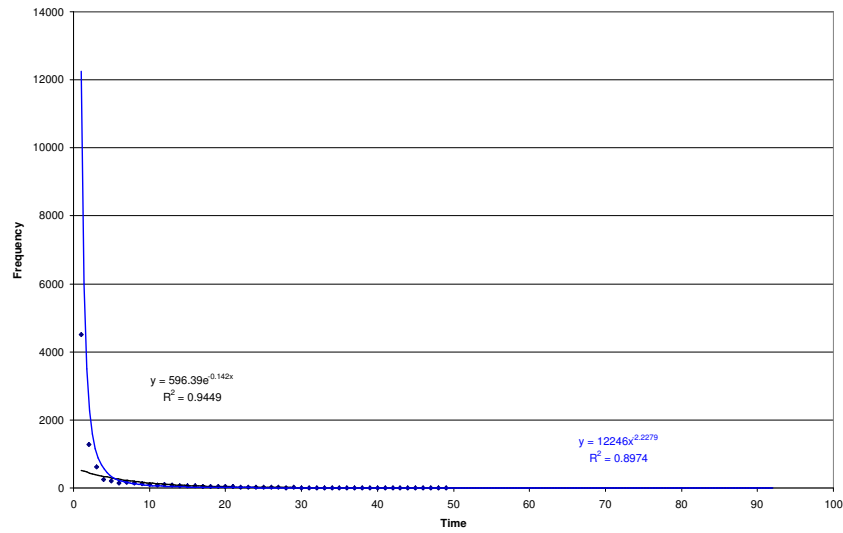


Figure 5.21: Distribution in the times taken to load individual documents.

2. Keep capitalized words and attempt to remove other words.

## 5.8 Concluding Remarks and Future Work

This chapter has described various aspects of a system for providing efficient, automated trail discovery. The following chapters will now explore how this technology can be applied to the fields of Web navigation, join discovery in relational databases and program comprehension. However, there are several important areas still to be addressed, which will have an impact in some or all of these areas.

### 5.8.1 Summaries

The summaries computed are effective for natural text. However, many of the pages on the web feature large numbers of short phrases, often in lists and tables. Further work is required to make the summaries more effective in these situations.

In order to evaluate the effectiveness of these summaries in allowing users to discriminate between relevant and non-relevant documents, the following experiment is proposed. Documents from the TREC corpus are presented at random as the results of fixed queries. Subjects should then be asked to mark the documents as non-relevant, relevant or highly relevant with respect to the query. The better the correlation between the subjects assessments and those of the NIST assessors, the better the quality of the summaries. The use of TREC data in the evaluation of computed summaries is discussed in detail in Tombros 1997. An alternative way to conduct the experiment is to ask users to assess pages based upon the summary then on the basis of the page's full text.

### 5.8.2 Multi-page Search

Search engines display pages of results with the option to display the next or previous  $n$  matching documents. A similar option should be supported to display the next or previous  $n$  trails. Displaying previous results should be fairly easy. If results are cached in an appropriate format, they may be read back when required. To compute the next  $n$  trails, either the trails from the next  $n$  starting points can be returned, or the next  $n$  trails can be computed with the page scores set to zero for all pages in the previous trails. This will increase the likelihood of new content in the following trails. There is an implicit assumption in this algorithm that the  $p$ th set of trails will only be accessed once the  $(p - 1)$ th set has been viewed. Otherwise, recomputation of all previous results is required. This would be complex and slow. However, with this restriction, computing the  $p$ th set becomes trivial as the  $k$ th set can be retrieved from the cache whenever  $k < p$ .

### 5.8.3 Partial Collection Ranking

Conventional search engines return a ranked list of documents. The list of potentially relevant documents is often extremely long and full rankings will take time in the order of  $O(\sum_{keyword \in collection} count(keyword))$ . However, users very rarely go beyond the first few pages of results (Silverstein, Henzinger, Marais, and Moricz 1999; Spink, Jansen, Wolfram, and Saracevic 2002). It is therefore unnecessary to compute the scores for the pages beyond

those which are displayed in the first instance. The higher computation cost of the few occurrences in which later pages are viewed is offset by the saving for the most common events. The trail finding algorithm works by computing the first set of trails, but still requires a fairly comprehensive scoring of the document set.

Partial document ranking is a strategy in which only these top documents are scored and ranked. Wong and Lee's proposal is to sort the posting list in descending order of  $tf$  values. These lists are then arranged into buckets. At query time, buckets are selected in descending order of combined  $tf.idf$  values. Scores are computed only for those pages in buckets where there is a possible alteration of the result (Wong and Lee 1993).

Partial document ranking is difficult to implement in the navigation engine due to the need to determine scores for documents which match few keywords. If the operation of the query engine is split into two – with the inverted index being used only for selecting starting points and a separate index being used for the Best Trail algorithm – then the risk is run of several thousand, or even several tens of thousands of disk accesses, which is orders of magnitudes less efficient than a full ranking. The algorithm outlined in Wong and Lee could be applied with lists sorted by the probability of being required, but there is probably a more effective way to achieve these savings. Until this is achieved, there will be a significant performance penalty in the IR stage before trail discovery can proceed. It is interesting to note that a previous system for constructing query-specific guided tours was reported to work best when the only documents returned were those with scores higher than 30% of the maximum (Guinan and Smeaton 1992). This may suggest a similar threshold for partial document ranking.

#### 5.8.4 Incremental Crawling and Merging Webcases

Search engine companies need to update their indexes frequently. In order to save bandwidth, increase coverage and improve results, incremental crawling techniques are used. A subset of the indexed pages are recrawled. These and newly found pages are indexed and merged with the master index. In order to apply these techniques to the navigation engine architecture, the following data structures need to be merged:

**Web Graphs** If the graphs are small enough to fit in main memory this is easy, the graphs can be loaded in order with the URL IDs converted using the forward and reverse lookup tables.

**MetaData** Metadata from multiple sources can be merged using the same techniques.

**Inverted Files** These are probably best merged at the previous temporary file stage, as the normalization factors and  $idf$  values will change globally with each page.

The Implementation of these strategies is considered a priority for future development.

## Part III

# Applications for Trail-Discovery

## Chapter 6

# Navigating the Web

“Would you tell me, please, which way I ought to go from here?”

“That depends a good deal on where you want to get to.” said the Cat.

“I don’t much care where—” said Alice.

“Then it doesn’t matter which way you go,” said the Cat.

“—so long as I get somewhere,” Alice added as an explanation.

“Oh, you’re sure to do that,” said the Cat, “if you only walk long enough.”

The Cheshire Cat, in *Alice in Wonderland* (Carroll 1865)

I must not fear. Fear is the mind-killer. Fear is the little-death that brings total obliteration. I will face my fear. I will permit it to pass over me and through me. And when it has gone past I will turn the inner eye to see its path. Where the fear has gone there will be nothing. Only I will remain.

Bene Gesserit litany against fear, in *Dune* (Herbert 1965)

Try not to think of it in terms of right and wrong. She is a guide, Neo. She can help you to find the path.

Morpheus, in *The Matrix* (Wachowski and Wachowski 1999)

## 6.1 Introduction

Chapter 3 presented an algorithm for computing memex-like trails on Web-like graphs. Chapter 5 described a navigation engine based around this algorithm, which computes these trails based upon computed node scores and augments the returned trails with metadata and document extracts. This chapter describes how this engine presents trails in Web sites and presents a comprehensive set of studies evaluating the effectiveness of the system and the trails produced. The last section explores possibilities for scaling to larger Web-sized corpora by distributing the indexes, splitting the associated Web graph and merging the results from multiple sources.

The rest of this chapter is organized as follows:

**Section 6.2** presents three user interfaces which can be used to display the trails. The NavSearch UI provides continuing support and display of results in a tree constructed by merging trails with common roots. The flat TrailSearch UI is very similar to that of a traditional search engine. The VisualSearch UI is a prototype developed using the GraphViz (Gansner, Koutsofios, North, and Vo 1993) program to presents trails in the form of a graph.

**Section 6.3** presents examples of trails provided by the navigation engine on indexed Web sites. The examples shown come from the Web sites for UCL<sup>1</sup>, Birkbeck<sup>2</sup> and Sleepycat Software<sup>3</sup>.

**Section 6.4** describes a case study into the use of the navigation engine. Taking the SCSIS Web site as an example, various problems are analysed along with an examination of the effectiveness and usefulness of the results to 9 queries taken from the department search logs.

**Section 6.5** discusses Mat-Hassan and Levene's user study showing how the trail-based search and navigation engine improves users' navigation efficiency. After using the system, 96% of the study's subjects chose NavSearch over Google and Compass as their preferred search engine.

**Section 6.6** presents an empirical evaluation comparing trails produced by the system to those authored by human editors. Although the authored trails are of consistently higher quality, the effort expended in producing them is prohibitive.

**Section 6.7** describes a technique which could potentially be used to allow the navigation engine to scale to multi-billion document corpora. The Web graph is split and distributed with trails computed on each section. New algorithms are presented for merging the resulting trail sets, allowing large scale deployment.

**Section 6.8** highlights the main conclusions which can be drawn from the evaluation sections and makes suggestions for future work.

---

<sup>1</sup> <http://www.ucl.ac.uk/>

<sup>2</sup> <http://www.bbk.ac.uk/>

<sup>3</sup> <http://www.sleepycat.com/>

## 6.2 Navigation Interfaces

The previous chapters have shown how trails can be found in Web-like graphs. This section presents three interfaces for presenting such trails.

### 6.2.1 NavSearch

Continuing support and display of results is important in reminding users of their options. Pollock and Hockley state that “Search engines should aim to communicate the concept that searching on the Internet is a process rather than an event” and that during their usability studies “Yahoo was the most popular search engine, largely because it leads users through the process of browsing” (Pollock and Hockley 1997). Principles such as these provided the motivation for the Mozilla sidebar, which allows search results and other information to be kept constantly present in a pane on the left hand side of the browser.

Continuous access to search facilities is equally important. Nielsen states that Web site authors “should make search available from every page on the site; you cannot predict where users will be when they decide they are lost” (Nielsen 2001). This philosophy has motivated many Web page layouts, and the development of tools such as the Google tool bar (Google 2002a).

The default *NavSearch* interface combines both of these ideas and extends them. The NavSearch interface includes a navigation toolbar at the top and a navigation tree window on the left hand side. The navigation tool bar displays both the best trail found with respect to a users query and a search box allowing the user to enter further queries.

The navigation tree displays a set of trails arranged in a tree-like form by merging trails with common roots. Roots are combined by extending the techniques described in Wexelbat and Maes 1999 to accomodate duplicates. Any page on a trail may be brought into the main window by clicking on its associated hyperlink.

To allow a quicker overview of a page’s key features, a pop-up window with several important pieces of information appears whenever the user positions the cursor over the link. This pop-up contains the page’s title, filetype (if not HTML), size, and the summary computed using the algorithm described in section 5.7. This strategy has been shown to improve users’ understanding. The pop-ups are based on ideas shown in Weinreich and Lamersdorf 2000 and contain many of the same elements. However, these pop-ups are based upon metadata stored by the system, and do not display in the main page. Hence, they do not require any proxy to filter the Web pages.

Figure 6.1 shows the first NavSearch interface. This incorporated the same pop-up mechanism used by Weinreich and Lamersdorf and an expandable tree. Some confusion arose with this interface. Similarities between this layout and the document view tree used in Microsoft Word and Adobe Acrobat led several people to believe either that the interface displayed trails through a *document*, instead of a Web *site*, or that it displayed a hierarchy. NavSearch’s ability to display links to sections within documents (if given by the link’s authors) supported these misconceptions. In order to improve the usability, the collapsible tree was replaced by a fixed, fully-expanded tree with new graphics highlighting the trail with arrows, as shown in figure 6.4.

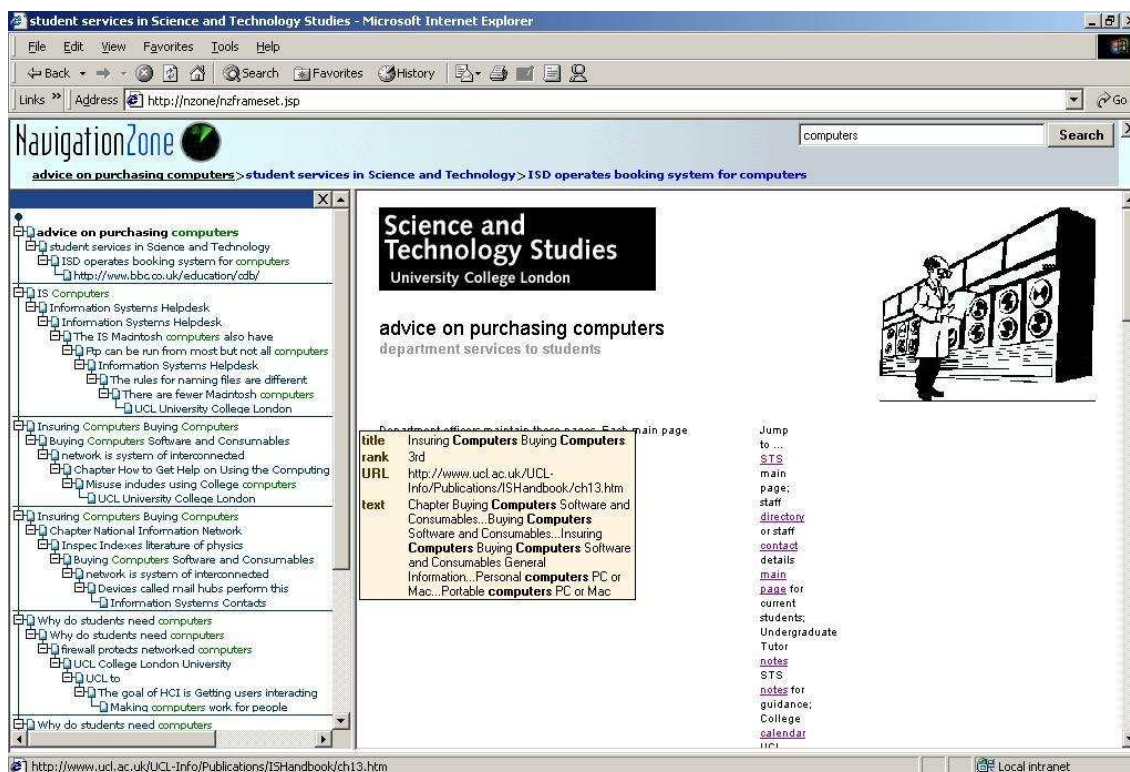


Figure 6.1: Results for the query “computers” on the UCL Web site. Whilst the trails are relevant to the query, some confusion arose with the interface layout and the trails contained too much redundant information.



Later versions of the interface replaced the pop-up code to allow the pop-ups to move accross frames without the need for proxying. The delay between moving the mouse over the URL and the pop-up appearing was also greatly reduced as this was found to be frustrating when trying to find information, despite being common practice in applications where such idioms are used to supplement help systems.

## 6.2.2 TrailSearch

The flat TrailSearch user interface appears very similar to that of a traditional search engine. Each trail appears sequentially and users can follow any link they choose. Such an interface is the best choice for introducing new users who are already familiar with search engines to the ideas of a returned trail. However, it is difficult to see the context of a node or the structure of a site in such a display and there is inadequate support during the navigation session.

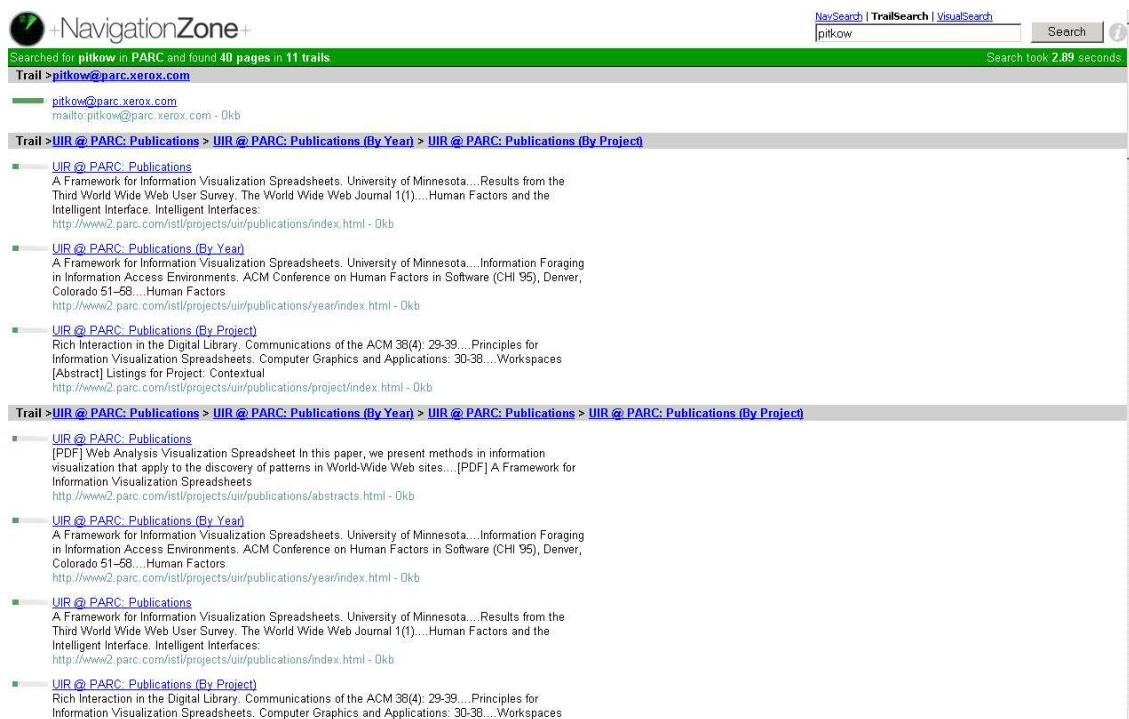


Figure 6.2: Results for the query “pitkow” on Xerox PARC’s Web site, presented using the TrailSearch interface.

### 6.2.3 VisualSearch

A prototype interface has been developed, using the GraphViz program (Gansner, Koutsofios, North, and Vo 1993) which displays the results in the form of a graph, where each trail is indicated by a different colour. The trail set output is simply piped to the clickable image map generator. Each trail is shown in a different colour on a graph where common nodes are shown only once. Figure 6.3 shows an example of how the trails would be presented using the VisualSearch interface for results to the query “oxygen” on the Creoscitex<sup>4</sup> Web site. The trails describe the Oxygen scanning application.

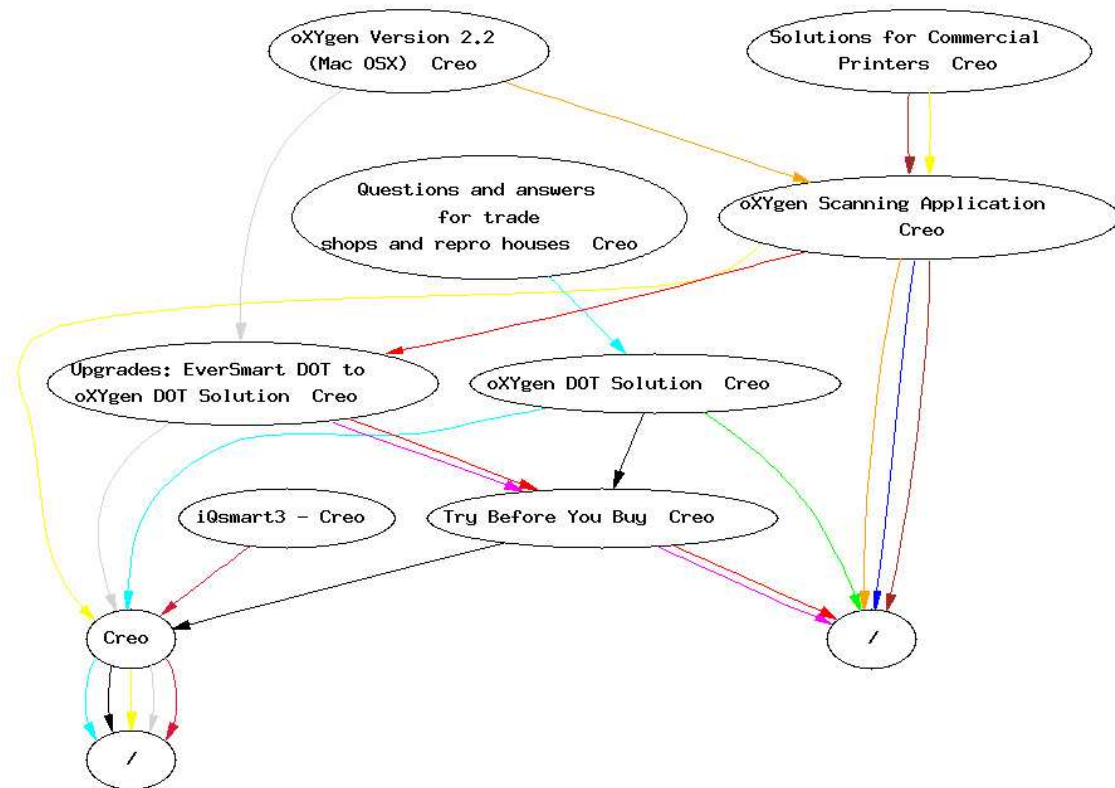


Figure 6.3: Results for the query “oxygen” on the Creoscitex Web site, as they would be presented using the VisualSearch interface.

<sup>4</sup> <http://www.creo.com/>

## 6.3 Web Site Examples

This section presents examples of the trails provided by the navigation engine in response to some queries on various Web sites. Many sites were indexed during the development of the navigation engine, only a small sample of which are represented in this thesis. The examples chosen are representative of the trails produced by the navigation engine and reflect some of the problems faced during development.

### 6.3.1 SleepyCat

Sleepycat Software produces Berkeley DB, a commonly used datastore, used in the development of the navigation engine, and in the Stanford WebBase project (Hirai, Raghavan, Paepcke, and Garcia-Molina 2000). The site consists of around 1 106 pages (see figure 3.10), many of which document the Berkeley DB libraries. Figure 6.4 shows the results of the query “Dbt”. A Data Base Thang (DBT) is a simple structure used to represent each element in a key/data pair.

The screenshot shows a web browser window with the NavigationZone interface. The search bar at the top right contains the query "dbt". The main content area is titled "DBT: Key/Data Pairs" and contains the following text:

Storage and retrieval for the Berkeley DB access methods are based on key/data pairs. Both key and data items are represented by the DBT data structure. (The name *DBT* is a mnemonic for *data base thang*, and was used because no one could think of a reasonable name that wasn't already in use somewhere else.) Key and data byte strings may refer to strings of zero length up to strings of essentially unlimited length. See [Database limits](#) for more information.

```
typedef struct {
    void *data;
    u_int32_t size;
    u_int32_t ulen;
    u_int32_t dlen;
    u_int32_t doff;
    u_int32_t flags;
} DBT;
```

In order to ensure compatibility with future releases of Berkeley DB, all fields of the DBT structure that are not explicitly set should be initialized to nul bytes before the first time the structure is used. Do this by declaring the structure external or static, or by calling the C library routine `bzero(3)` or `memset(3)`.

By default, the `flags` structure element is expected to be set to 0. In this default case, when the application is providing Berkeley DB a key or data item to store into the database, Berkeley DB expects the `data` structure element to point to a byte string of `size` bytes. When returning a key/data item to the application, Berkeley DB will store into the `data` structure element a pointer to a byte string of `size` bytes, and the memory to which the pointer refers will be allocated and managed by Berkeley DB.

The elements of the DBT structure are defined as follows:

```
void *data;
    A pointer to a byte string.
u_int32_t size;
```

The left sidebar shows a trail of navigation links starting from "Dbt" and going through various database-related pages, including "C Interface", "DBcursor", "DBT", "Key/data pairs", "Berkeley DB Tutorial and Reference", "Release 3.1: DB>put", "DbMemoryException", "Common errors", "Dbt", "Db", "Secondary indices", and "Dbt->associate".

Figure 6.4: Results for the query “Dbt” on the Sleepycat Web site.

Figure 6.5 shows the trails found for the query “Dbt.” The last trail shows the Java classes relating to the Dbt handling – in particular the documentation for the method `Db.get()` and the classes `Dbt` and `DbMemoryException`. In this sense the trails separate the different language Application Programming Interfaces (APIs) and the context allows developers to discriminate between C, C++ and Java APIs. Unfortunately, the trails all eventually lead to the C interface declarations as this is default target for links in the documentation. The suc-

cess of the Sleepycat crawl was inspirational to the development of AutoDoc and AutoCode, as discussed in chapter 8.

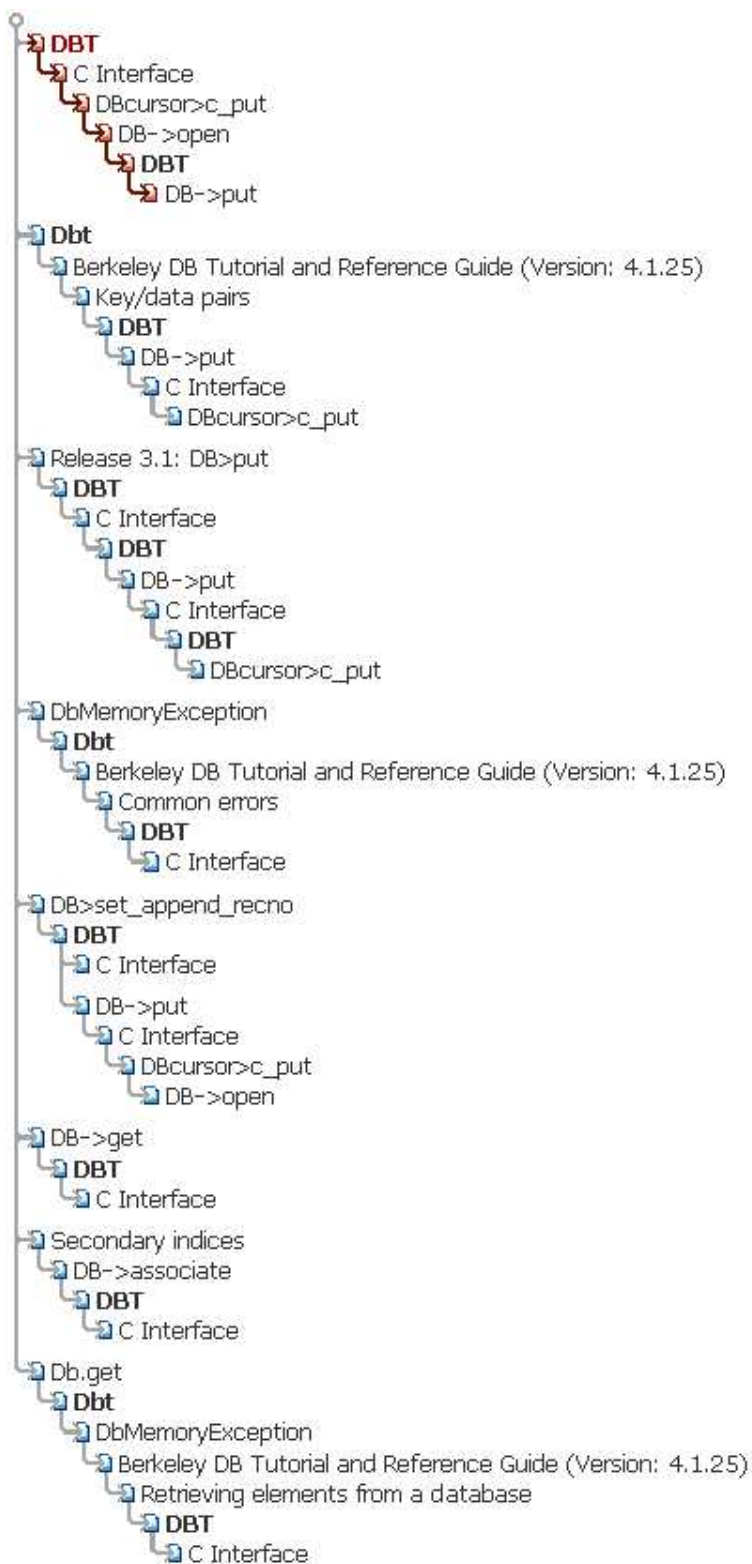


Figure 6.5: Trails found for the query “Dbt” on the Sleepycat Web site.

### 6.3.2 University College London

UCL was founded in 1826 and is one of the largest universities in England. The university is comprised of 70 departments spread over 10 Faculties and Schools, including the Bartlett Faculty of the Built Environment, the Royal Free and University College Medical School and the Mullard Space Science Laboratory.

Not surprisingly, the Web site is large, disorganized and covers a wide range of topics. A full crawl of the `ucl.ac.uk` domain covers over 1 million pages and is typical of large organizations where the responsibility for authoring pages is spread between many departments and staff members. By comparison, Google claim to index only 353 000 of these pages. Coverage such as this is a penalty that must be paid for using the results from a global search engine in preference to a locally administered site search.

Figure 6.6 shows the results for the query “Cryptography” on the UCL Web site. The first two trails show details of public cryptography techniques, as taught in the Computer Science department. Later trails detail related classes and implementations of cryptographic systems.

NavigationZone  
Trail: Public Key Cryptography > What size keys? > Symmetric Cryptography > What is Cryptography?

Found 31 pages in 11 hits  
Search took 2.01 seconds

Next: [Network level solutions Up: A brief Introduction to](#) Previous: [What size keys?](#)

## Public Key Cryptography

Public key encryption is a much slower alternative to symmetric cryptography. Its based upon mathematical functions upon two pairs of numbers. For the well-known RSA algorithm, the security comes from the difficulty of factoring large numbers in Galois Fields.[\[1,4\]](#)

Each key is a pair of keys  $K$  and  $K^{-1}$ . If a message is encrypted using  $K$  then it can only be decrypted using  $K^{-1}$ . If  $\wedge$  means the application of the encryption function and  $text$  is the cleartext, then the following all hold true.

$$(text) \wedge K \wedge K^{-1} = text$$

$$(text) \wedge K \wedge K \neq text$$

$$(text) \wedge K^{-1} \wedge K = text$$

$$(text) \wedge K^{-1} \wedge K^{-1} \neq text$$

Importantly, one cannot derive  $K$  from knowledge of  $K^{-1}$  or vice versa. This allows the primary use of public key technology, where one key is made public and one key remains secret. This provides a much larger degree of functionality, extending the use of cryptography to supply authentication and integrity as well as confidentiality.

Java 2 Platform SE v1.3.1: Class [SecureRandom](#)  
See Appendix A in the Java [Cryptography Architecture](#) API Specification Reference for information about standard PRNG algorithm names. provider - the name of the provider. Returns: the new SecureRandom object. Throws: [NoSuchAlgorithmException](#)

<http://www.cs.ucl.ac.uk/teaching/java/dk1.3/tccs/api/java/security/SecureRandom.html>

NavigationZone © 2000-2002

Figure 6.6: Results for the query “Cryptography” on the University College London Web site.

### 6.3.3 Birkbeck

Figure 6.7 shows the trails produced on the Birkbeck site for the query “hotel management”. Presumably the query was posed by a prospective student looking for courses in hotel management similar to those provided by Shannon College<sup>5</sup> or Thames Valley University<sup>6</sup>. Unfortunately, no such course is run by Birkbeck. However, the context provided by the trails in figure 6.7 allows the user to quickly locate the management school course lists and determine the courses which best match the student’s requirements.

---

<sup>5</sup> <http://www.shannoncollege.com/>

<sup>6</sup> <http://www.tvu.ac.uk/>



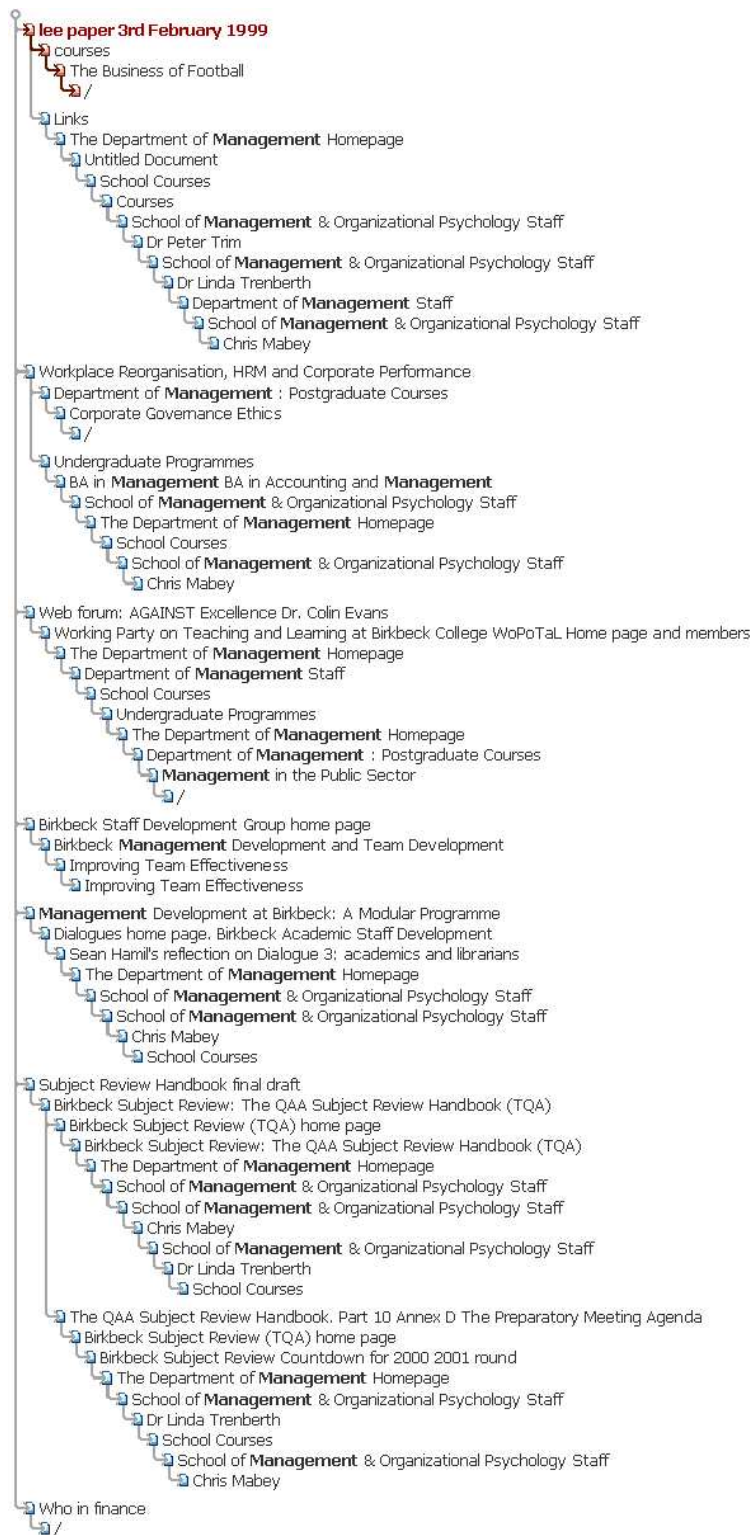


Figure 6.7: Trails found for the query "hotel management" on the BBK site.

## 6.4 Case Study – SCSIS

This section describes a case study into the use of the navigation engine. The SCSIS Web site has been chosen as an example of how the navigation engine can be used to provide site search facilities. The problems encountered are discussed, along with an examination of the effectiveness and usefulness of the results to 9 queries taken from the department search logs.

The crawl of the department's Web site yielded a graph with a total of 6 863 nodes and 15 055 links covering 2 448 fully indexed pages. The remaining nodes correspond to external URLs or inaccessible pages. The average outdegree of the indexed pages was 6.15 (see figure 3.10).

Nine queries were taken from a recent log file. The results of the queries are presented in alphabetical order:

**accomodation** The results shown in figure 6.8 highlight three major problems. Firstly, neither the Web site authors nor the end-user seem to be able to spell *accommodation* (two “m”s) correctly or consistently. Research has shown that this is a common problem (Silverstein, Henzinger, Marais, and Moricz 1999). Stemming (Porter 1980) and *n*-gram based indexing can help reduce these errors, but cannot eliminate them and introduce different complexities. A better solution is that used by Google, which is to prompt the user for alternative suggestions.

When given the correct query, the results are substantially more useful, as can be seen in figure 6.9. Trails cover the accommodation facilities provided in halls of residence and local hotels for the International Conference on Database Theory (ICDT) conference and associated Web dynamics workshop hosted by the school of computer science in January 2001.

Secondly, there is a problem with keeping Web site material and indexes current. The information in the trails, whilst the most relevant and pertinent on the site and clearly associated with the conference by the structure of the trails, is over 2 years old. However, the location of the Hotels and halls of residence is unlikely to have changed in that time. Identifying which resources will be effected by changes over time is a difficult topic worthy of future research.

Finally, there is the problem of generating short titles (see also section 5.7). Many different pages are shown in the trails shown in figure 6.8, all of which relate to the Web Dynamics workshop and all of them contain the keyword, “accomodation”. Unfortunately there is no means to discriminate between them. The authors of the pages made no changes in the `h1` or `title` tags by which to identify the differences. The most appropriate title is contained in a later `h3` tag.

**andrew** The results in figure 6.10 are useful in that they highlight the home pages of Andrew Bielinski, Andrew Watkins and Andrew Mair. They are also informative in that the structure of related pages shows Andrew Bielinski to be a research student under the supervision of Mark Levene and that Andrew Mair is associated with the BSc Information Systems and Management course.

Two problems remain. Firstly, that a description of Andrew Watkin's page is missing, along with an appropriate title. This is because the pages are excluded by the

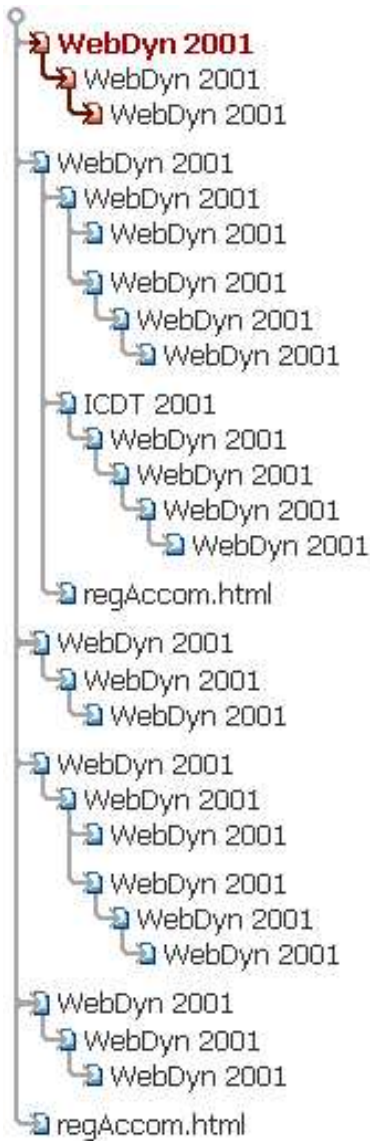


Figure 6.8: Trails found for the query “accomodation” on the SCSIS site.

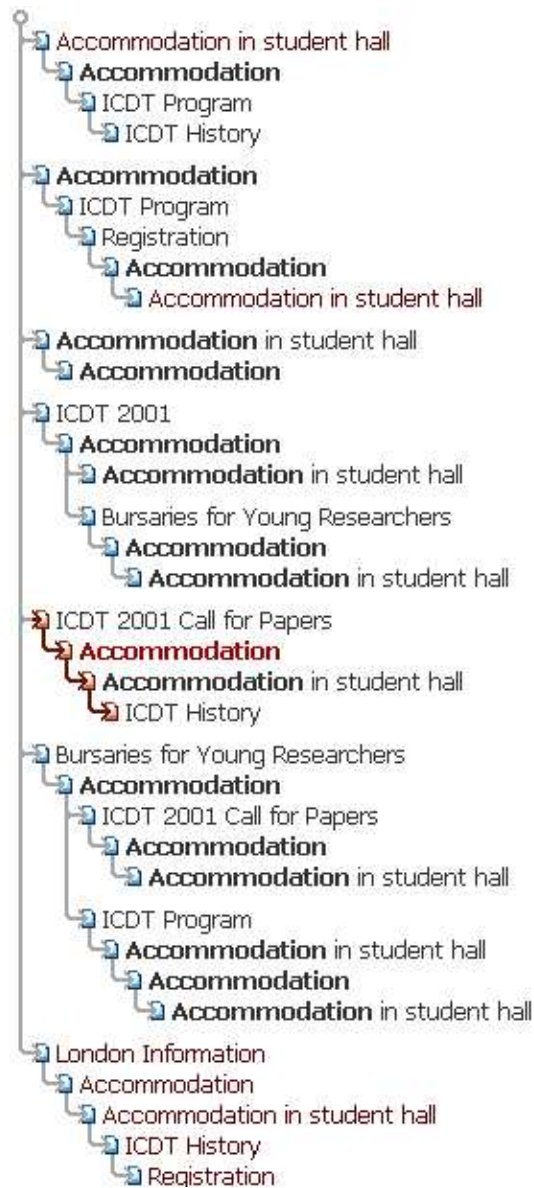


Figure 6.9: Trails found for the query “accommodation” on the SCSIS site.

`robots.txt` file (Koster 1994). Without close co-operation with the Web site administrators, no search facility will ever be fully effective. Secondly, the link between `rstudentperson.asp?name=bielinski` and `bielinski` is missing due to the robot following current best-practice and ignoring CGI-based pages.

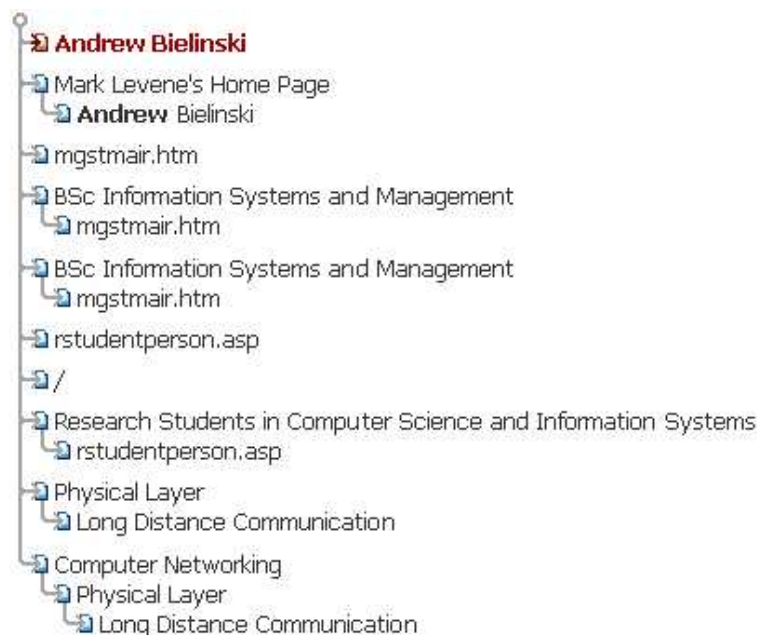


Figure 6.10: Trails found for the query “andrew” on the SCSIS site.

**application form** Figure 6.11 shows that some useful trails are found for this query. The first trail identifies the enquiry/application form for the MSc E-Commerce course. The second trail identifies the application form required for the undergraduate program. The third trail represents the start of the problems. The pages entitled “IT APPLICATIONS” are distinct but differ only by the inclusion of an irrelevant “assessment” section. This small difference causes the creation of 2 separate trails. This can be fixed with the application of near duplicate detection as described in section 3.5 (Broder 2000; Shivakumar and Garcia-Molina 1999).

More serious is that the engine finds neither the form for the foundation degrees nor the form for the postgraduate programme although both are within 2 clicks of the pages given. Further analysis shows that some serious problems still remain with the information retrieval techniques. The IR metrics used are significantly simpler, but slightly flawed when compared with the current state of the art.

**birkbol programmes** Figure 6.12 shows details of undergraduate courses and research degrees (first and second trails respectively) and an FAQ and guide for new students on the MSC course (sixth and eighth trails respectively). The results are promising, but the short trails fails to show much structure and the choice of query again highlights the failure of users to construct meaningful queries.

**c++ notes** Figure 6.13 shows that the query “c++ notes” produces by far the worst results

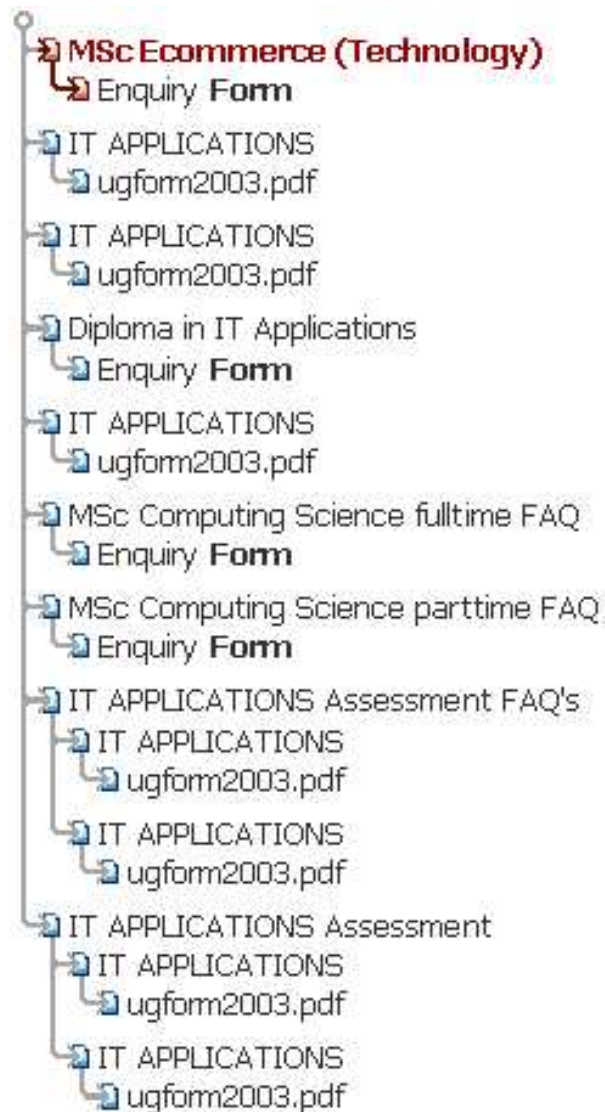


Figure 6.11: Trails found for the query “application form” on the SCSIS site.

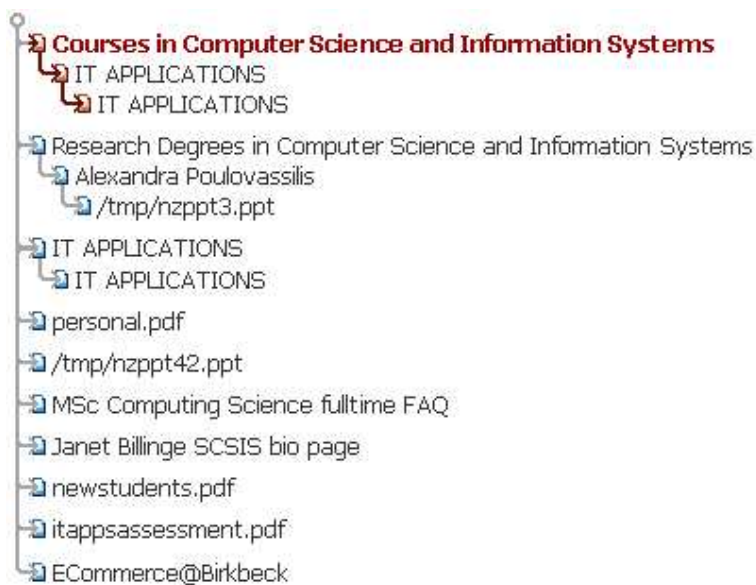


Figure 6.12: Trails found for the query “birkbol programmes” on the SCSIS site.

of any of those taken from the logs. The ability of the navigation engine to construct trails through lecture notes on various subjects is rendered useless by the inability of the parser to recognise the letters `c++` as a single term.

**exam papers** The user posing this query was almost certainly a student looking for past papers for revision. Figure 6.14 shows that the first two trails provide exactly that. The second trail shows that the papers relate to the module “Developing Internet Applications”. There are surprisingly few past papers available on the SCSIS site and the remaining trails for this query shows details relating to arrangements for sitting exams for that summer. The context provided by the trails makes it easier to distinguish between the two types of result.

**Mark** Mark Levene’s page appears continually throughout the top trails in the context of the Web dynamics workshop (which he organized), the Database and Web Technologies Group (of which he is a key member), his research students (Gaurav and Azy, whose page is entitled “Welcome to my homepage”) and his co-authors on several papers (Alexandra Poulouvassilis and George Roussos). However, the page never appears at the start of the first trail, as can be seen in figure 6.15. The probabilistic nature of the Best Trail algorithm causes this page to change occasionally but never to Mark’s home page. This strange behaviour is despite the IR algorithm scoring that page higher and is a disappointing result which raises questions over the suitability of the scoring functions used.

As with the results for the query “andrew”, neither the graph nor the resulting trails contains the link from `staffperson.asp?name=mark` to `~mark`, due to non-indexing of CGI-style pages. This behaviour needs to be customized on a per-webcase basis.

Links are also present on each page in the SCSIS site to the home page, news, courses, research, seminars, etc. All these are constructed using javascript, which the robot

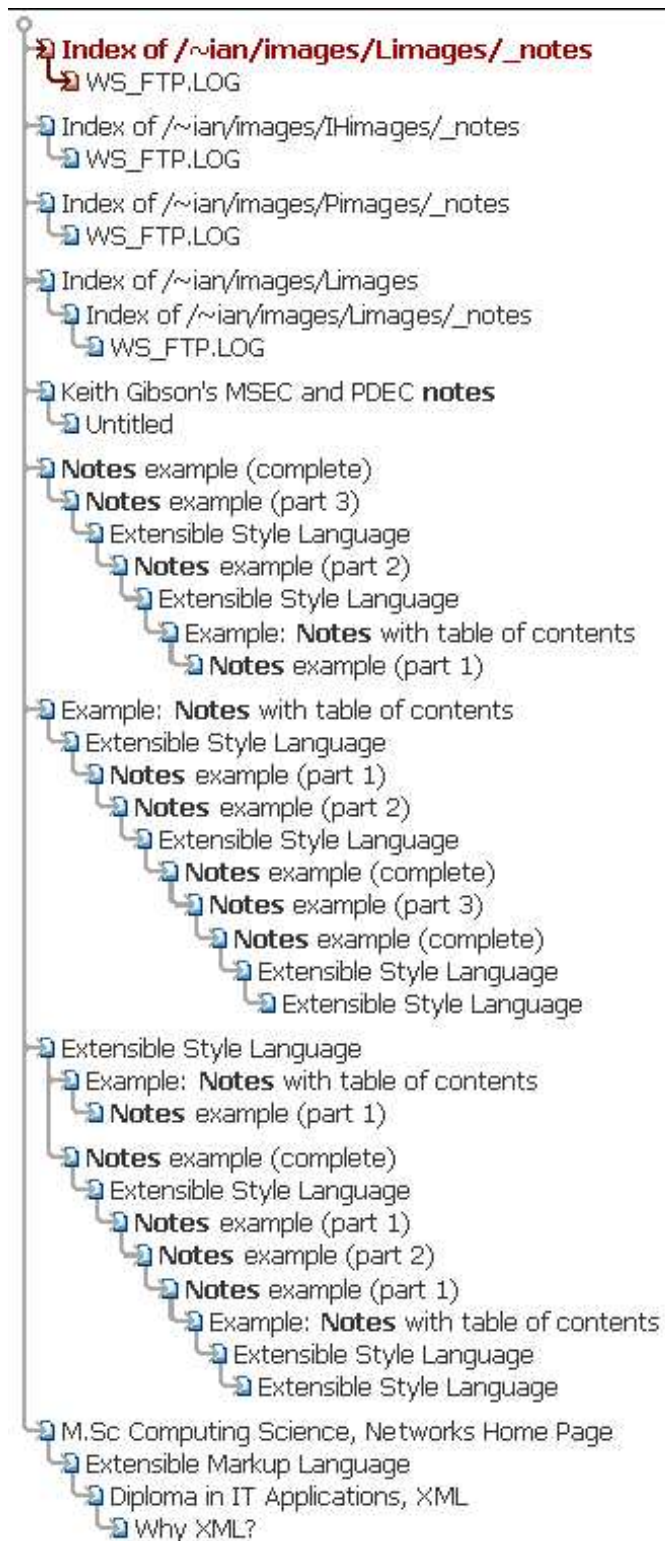


Figure 6.13: Trails found for the query “c++ notes” on the SCSIS site.





Figure 6.14: Trails found for the query “exam papers” on the SCSIS site.

will not recognize. Similar behaviour found with the output of Content Management Systems (CMSs) such as Vignette or Documentum. The long-term solution to this problem is to tie the trail engine into a better IR system and offer interface to the main CMSs. For the current research prototype this is not feasible, but would be essential if the navigation engine was to be developed fully.

**neural network** The first trail shown in figure 6.16 shows the course “Artificial Intelligence & Neural Networks”, as taught by Chris Christodoulou. The course is an introduction to a neural networks, genetic algorithms and clustering methods, but is given the title “BSc ISM Option Expert Systems”, despite being unrelated to expert systems.

Chris Christodoulou is the SCSIS expert on neural networks. The second trail leads from his home page to the only one of his papers, “A Spiking Neuron Model: Applications and Learning” linked to from his home page. Subsequent trails show the activities of the SCSIS research group relating to neural networks and various relevant papers.

**xml** The first two trails in figure 6.17 give brief tours of an XML tutorial, always linking to external resources containing a great deal of relevant information. The third trail provides an explanation of XML namespaces connected to hub with lots of XML references. The use of Potential Gain in the starting point selection encourages such hubs to be chosen. The fourth trail details the use and history of XML as a markup language and it’s relationship to SGML. Subsequent trails describe the Information Technology (IT) applications module on XML. This information is highly relevant for new students but is once again masked by problems generating short titles.

The conclusion that can be drawn from this analysis is that the trails found by the navigation



Figure 6.15: Trails found for the query “mark” on the SCSIS site.

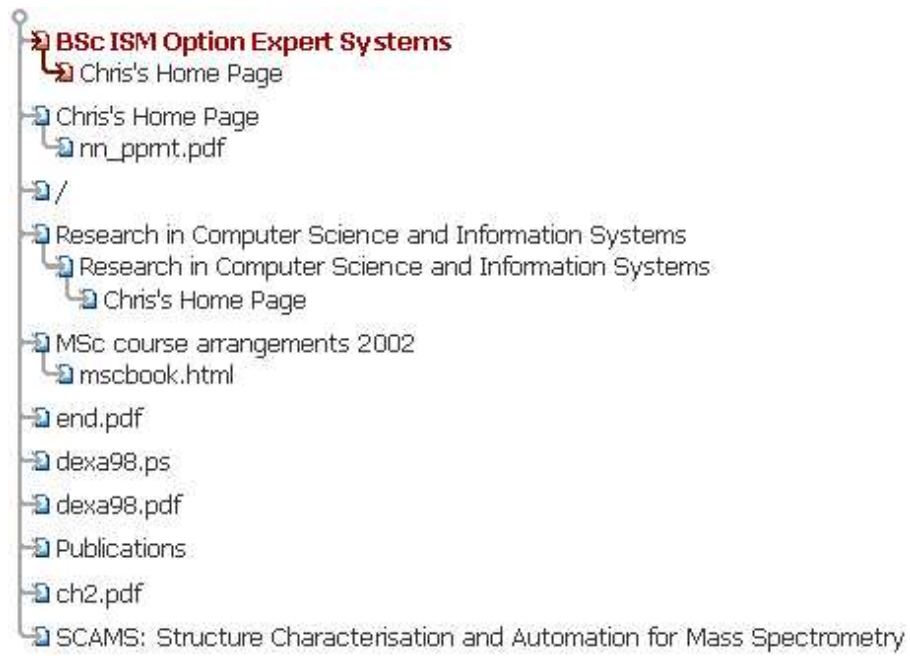


Figure 6.16: Trails found for the query “neural network” on the SCSIS site.

engine are useful, but the overall utility of the system is being limited by problems with related modules. However, to truly test the effectiveness of the system requires an independent test with real users.

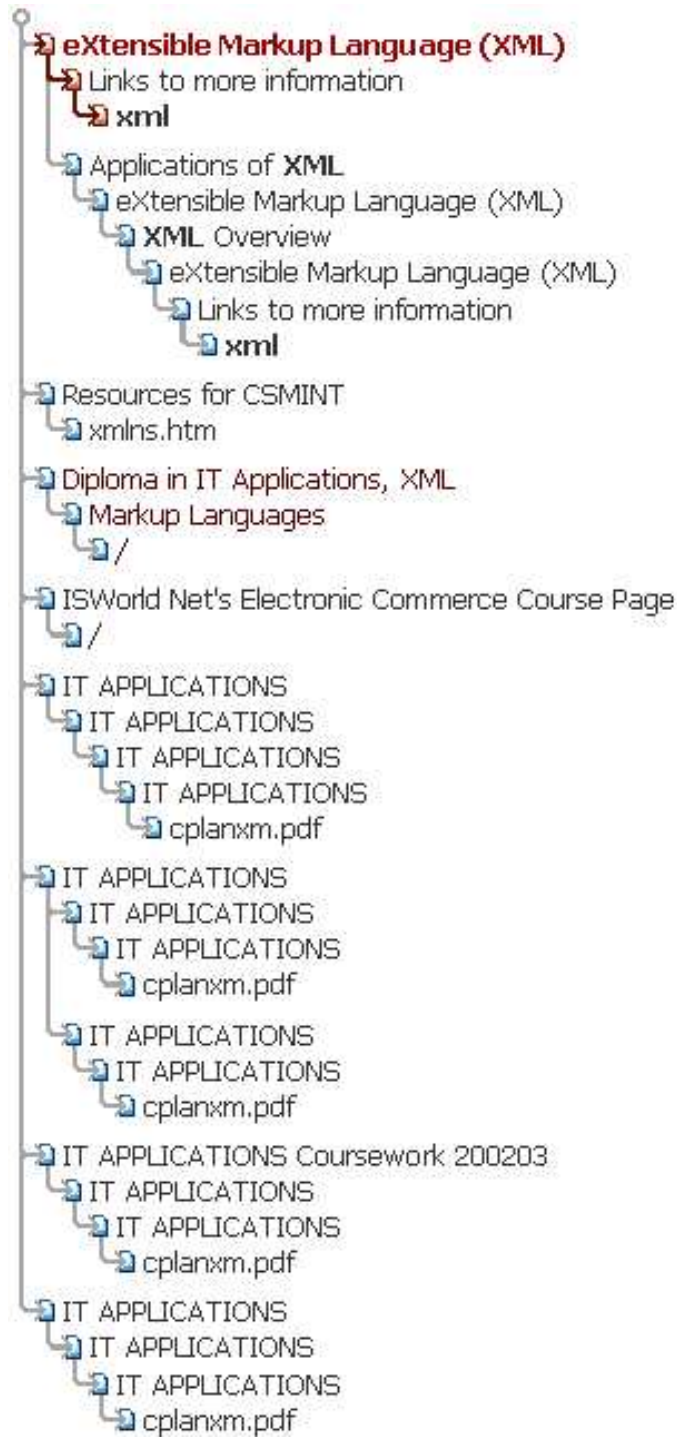


Figure 6.17: Trails found for the query “xml” on the SCSIS site.

## 6.5 Mat-Hassan and Levene's User Study

In order to assess the usefulness of the NavSearch interface and prove the hypothesis that “a trail-based search and navigation engine improves users' navigation efficiency”, Mat-Hassan and Levene conducted a usability study. The results they obtained from the study revealed that users of the navigation engine performed better in solving the question set posed than users of a conventional search engine (Mat-Hassan and Levene 2001).

Users were given two sets of information seeking tasks to complete based upon the pages in UCL's official Web site. Three different search tools were evaluated, one of which was the navigation engine with the NavSearch interface. The others were Compass (UCL's official site search engine) and Google's university search of UCL<sup>7</sup>. Subjects were asked to answer two sets of questions, devised to be at the same level of difficulty, using either NavSearch and Google or NavSearch and Compass. The question sets were formulated so that all the questions fell within one of five types : fact finding, judgement questions, comparison of fact, comparison of judgement and general navigational questions.

Most of the subjects assigned to use Google were more optimistic about the initial likelihood of completing the task, whilst those subjects assigned to use NavSearch were initially more reserved and pessimistic. None of the subjects had had any previous experience with NavSearch and they received no training, although they were encouraged to read the help page, and were given two minutes to familiarize themselves with the interface, during which questions could be asked. Familiarity was the main factor in favour of Google's linear interface model. Mat-Hassan and Levene reported that users “found the interface quite intimidating” considering it a “radical shift” from the conventional layout and format of results.

The interfaces were assessed according to users' completion time, the number of clicks employed, the number of correct answers found by the subjects and the confidence and satisfaction levels expressed by the subjects. When asked to compare NavSearch with Google or Compass, subjects expressed a much higher degree of confidence in their ability to complete future tasks, a higher degree of satisfaction with NavSearch with regards to the completion of tasks and a higher degree of satisfaction completion with regard to navigation and the display of results. Users stated that “showing link relationship helps” and that the system provided “useful trails” which gave “an indication of the pages already looked at and the pages that might be useful to look at”. 96% of the study's subjects chose NavSearch over Google and Compass as their preferred search engine. Mat-Hassan and Levene concluded that “the proposed user interface does indeed provide effective information retrieval assistance”.

---

<sup>7</sup> <http://www.google.com/univ/ucl>

## 6.6 Comparative Testing

The previous two sections have shown the utility and potential of the navigation engine. However, it would be interesting to discover to what degree the trails correspond to the ideal trails which can be found by traversal of the Web site. This section compares the trails found with the navigation engine with those authored by human subjects. Although it is acknowledged that authored trails will neither be complete nor impartial, they can be expected to be of a consistently higher quality than those constructed with existing information retrieval and trail-finding technology.

The purpose of the study is twofold. Firstly, to ascertain the extent of the gap between the human authored and machine constructed trails. Secondly, to identify common features of authored trails which should be incorporated into any future generation of trail-finding system.

### 6.6.1 Evaluation Philosophy

Eight characteristics have been discussed in previous research that may be applicable to the study (Hawking, Craswell, Bailey, and Griffiths 2001; Gordon and Pathak 1999). These characteristics, as taken from Hawking et al. 2001, are:

1. Searches should be motivated by genuine user need.
2. If a search intermediary is employed, the primary searcher's information need should be as fully captured as possible and transmitted in full to the intermediary.
3. A large number of search topics must be used.
4. Most major search engines should be included.
5. The most effective combination of specific features of each search engine should be exploited. I.e. the queries submitted to the engines need not be the same.
6. Relevance judgements must be made by the individual who needs the information.
7. Experiments should be well designed and conducted.
8. The search topics should represent the range of information needs both with respect to subject and to type of results wanted.

Of these the seventh should be a characteristic of any scientific research and the second is irrelevant as no intermediary is used. The first, third and eighth of these criteria are achieved by taking examples from the Birkbeck University of London Web site (see figure 3.10). Queries were taken from log data for the Birkbeck search engine. In total, 18 queries were used, which is less than the number used in either Hawking or Gordon's studies but more than were used in Leighton's study (Hawking, Craswell, Bailey, and Griffiths 2001; Gordon and Pathak 1999; Leighton and Srivastava 1999).

The sixth criteria is not met, as this would severely limit the scope of the evaluation, as noted by Hawking. Three subjects were asked to author trails, based upon queries taken

from three sets of log data<sup>8</sup>. In keeping with Hawking's proposals, the subjects were asked to author trails for queries where they could assess "what it was that the inquirer was actually seeking". Full details of all the trails are presented, including authored and constructed trails. The subjects were not asked to assess the relevance of these trails, or of the individual pages within them. As this thesis describes the only trail-based retrieval system available, any attempt at a comparison with existing systems would be meaningless. The fourth criteria is also not met for this reason.

The fifth criteria is also not met, in keeping with Hawking's observations and with previous studies showing that advanced search features are not commonly used (Hawking, Craswell, Bailey, and Griffiths 2001; Silverstein, Henzinger, Marais, and Moricz 1999; Jansen, Spink, and Saracevic 1998).

In addition the following restrictions were placed on the trail authors:

1. All pages should be on the Birkbeck site or one link away from it. This is in keeping with the abilities of a Web site search system.
2. All trails must be valid with respect to the Birkbeck Web site. This is consistent with the definition of a trail used within this thesis.
3. The navigation engine should not be used to help construct the trails nor to provide any other assistance during the experiment. Any other tools, including the Birkbeck site search and Google were permissible. This prevented unintentional bias in favour of the results returned by the engine.

## 6.6.2 Analysis of Queries

Figure 6.18 shows a summary of the queries chosen, along with the interpretation given to them by the trail authors. Figure 6.19 shows some basic statistics about the authored trails. As with the previous case study, the results and conclusions drawn from each query will be presented in turn:

**access course** The authored trails contain information on courses concerning Microsoft Access. The computed trails, shown in figure 6.20 show pages concerning access for disabled students. This implies that the IR techniques used are inadequate. It is impossible for the computer to fully comprehend the meaning of queries, but it is interesting to note that neither conjunctive queries, phrase matching or proximity would improve the results in this instance. The authored trails were:

1. (a) <http://www.bbk.ac.uk/ccs/>  
(b) <http://www.bbk.ac.uk/ccs/courses/menu.htm>  
(c) <http://www.bbk.ac.uk/ccs/docs/docs.htm#notes>  
(d) <http://www.bbk.ac.uk/ccs/docs/5-69.pdf>

---

<sup>8</sup> Kevin, <http://www.bbk.ac.uk/analog/search/searchlist01Jun03.html>  
Azy, <http://www.bbk.ac.uk/analog/search/searchlist08Jun03.html>  
Bryn, <http://www.bbk.ac.uk/analog/search/searchlist18May03.html>

Query	Description or requirements
access course	Information on any Microsoft Access course.
birbeck logo design	Guidelines on how to use BBK logo such as size, position and permission
design postgraduate	Design courses for postgraduate. It is not clear what kind of design though: Media? Architecture? Industrial? Interior?
exam papers	A list of all past years exam papers.
international students	Information for international students particularly relating to visas, fees, courses and accommodation.
writing up phd	Guidelines or information on writing up a thesis.
a student gym	Is there a gym that can be used by students at Birkbeck?
beginners painting	is there a beginners painting course at Birkbeck? If so, what are the details.
bsc programming	Is there an undergraduate course that will teach computer programming?
dept of philosophy	Information about the department of Philosophy at Birkbeck.
distance learning	Can courses at Birkbeck be completed via distance learning? or: What distance learning support is there at Birkbeck?
mba	Is there a Masters in Business Administration (MBA) course at Birkbeck? If so, what are the details?
access 97	Should address the facilities for using, or the use of Access '97 at Birkbeck.
project management	Should address the project management process at Birkbeck, information on courses in project management or information on the subject of project management.
part time	Should cover either the particulars of part time study at Birkbeck or the general topic of part time study.
research grants history	Should include either research grants in the subject of history, or the history of research grants for the whole of Birkbeck.
social	Should could contain pages regarding social life at Birkbeck or pages relating to social sciences.
ba history	Web pages relating to studying for a BA in History at Birkbeck or pages giving options for further study for those already holding that degree.

Figure 6.18: Queries used as the basis for authoring trails.



Query	Trails	Nodes	Distinct pages
access course	2	7	5
birbeck logo design	0	0	0
design postgraduate	3	9	9
exam papers	2	7	5
international students	1	2	2
writing up phd	3	5	5
a student gym	3	5	5
beginners painting	3	9	5
bsc programming	4	17	8
dept of philosophy	3	8	7
distance learning	7	17	17
mba	3	10	8
access 97	3	10	8
project management	2	14	14
part time	1	20	20
research grants history	1	4	4
social	1	4	3
ba history	1	9	2
AVERAGE	2.4	8.7	7.1

Figure 6.19: Statistics concerning the trails authored for the queries shown in figure 6.18.

2. (a) <http://www.bbk.ac.uk/ccs/>
- (b) <http://www.bbk.ac.uk/ccs/docs/ittraining.html>
- (c) <http://www.bbk.ac.uk/ccs/docs/ittraining.html#courses>

It is interesting to note the use of the local or anchor links (those ending with #foo) in the second trail. There are 17 URLs in the authored trails containing anchors and 6 of the 43 authored trails (over 10%) contain links within a single page. This suggests that the importance of such links has been underestimated in previous research. It is already acknowledged that many portal pages can be split into self-contained sections (Anderson and Horvitz 2002), and this finding suggests the same applies to other pages. Should future studies support this finding, then it would suggest that the strategy of treating all such URLs as equivalent is flawed. It would further suggest that a new IR engine should be developed which identifies linked components within each page.

**birbeck logo design** Strangely, the author could find no pages with which to construct trails, despite using the Birkbeck search and manual navigation from the home page, the registry pages and the “About Birkbeck” pages. The navigation engine produced the trails shown in figure 6.21 which show that pages with such guidelines do exist but are difficult to locate. The experience of the trail authors has shown that manually authoring trails is difficult and time-consuming. This fact brings into question the notion of Bush’s Trail-blazers (Bush 1945) as human agents and may explain the low adoption rate of previous trail authoring systems.

Ironically, the page with the guidelines for use of the logo in printed works is shown in

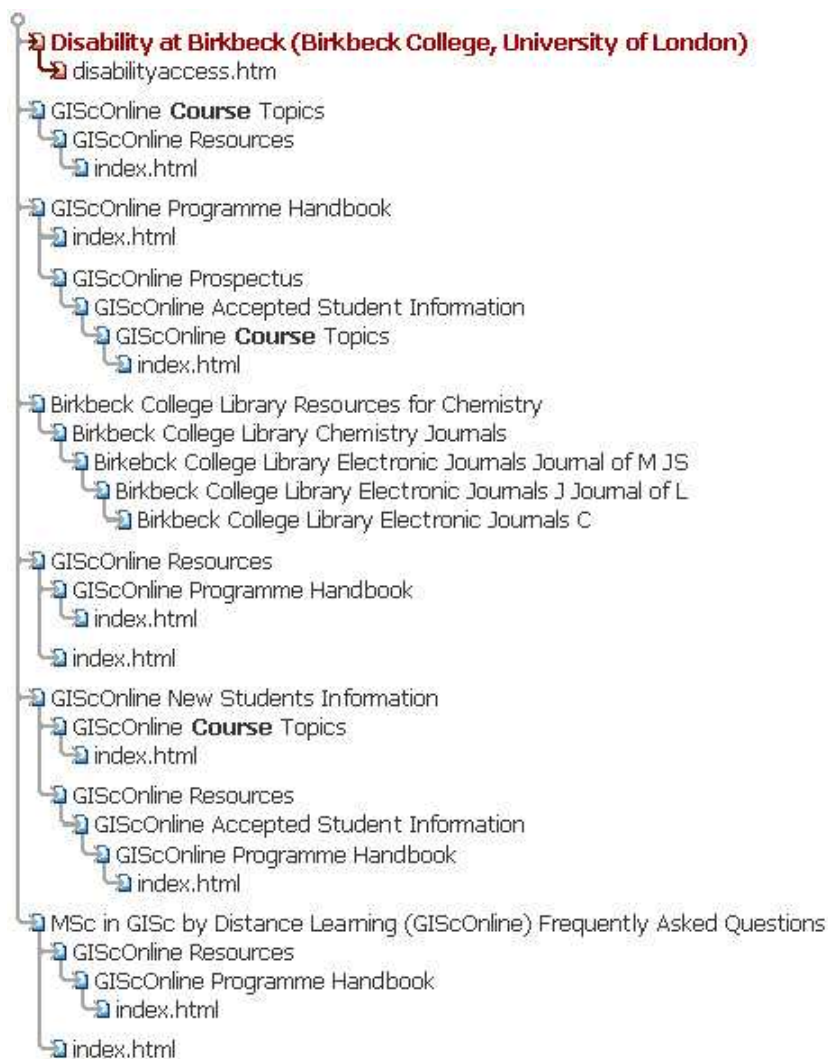


Figure 6.20: Trails found for the query “access course” on the Birkbeck site.

the trail, but is inaccessible to most users of the site, as it is available only to members of staff.

**design postgraduate** The user posing this query was probably interested in a postgraduate course concerning some form of design. However, it is unclear what type of design the user was most interested in. The trail author suggested media design, architecture, industrial design or interior design as possible answers, but there are undoubtedly more. It is important for such ambiguous queries that the results still allow the user to identify possible answers and give guidance for reformulating the query. The following trails, suggested by the author, perform this to a limited degree:

1. (a) <http://www.bbk.ac.uk/study/pg2003/index.html>
2. (a) <http://www.bbk.ac.uk/hafvm/>  
 (b) <http://www.bbk.ac.uk/hafvm/courses.html>  
 (c) <http://www.bbk.ac.uk/study/pg2003/histart/arthisdma.html>
3. (a) <http://www.bbk.ac.uk/academic.html#fce>  
 (b) <http://www.bbk.ac.uk/fce/>  
 (c) <http://www.bbk.ac.uk/study/fce/>  
 (d) <http://www.bbk.ac.uk/study/fce/mediastudies/idxmediastudies.html>  
 (e) <http://www.bbk.ac.uk/study/fce/mediastudies/webd.html>

In contrast, the navigation engine produced those shown in figure 6.22, which show pages on post-graduate courses which happen to mention design issues. The system's reliance on keyword-based retrieval means that the full semantics are not captured.

**exam papers** The author believed that, as with the example in the SCSIS case study, the query was posed by a student looking for a list of past papers for revision. The authored trails (shown below) detail pages on the Birkbeck Electronic Library, as do the trails produced by the navigation engine, as can be seen from the results in figure 6.23. The trails produced by the navigation engine each cover a separate subject area.

1. (a) <http://www.bbk.ac.uk/lib/>  
 (b) <http://www.bbk.ac.uk/lib/eresources.html>  
 (c) <http://bel.bbk.ac.uk/bel2/>
2. (a) <http://www.bbk.ac.uk/lib/>  
 (b) <http://www.bbk.ac.uk/lib/info.html>  
 (c) <http://www.bbk.ac.uk/lib/guide5.html>  
 (d) <http://bel.bbk.ac.uk/bel2/>

**international students** Whilst looking for relevant information on visas, fees, courses and accommodation, the trail author could only find sufficient information of relevance to construct the following single trail:

1. (a) <http://www.bbk.ac.uk/study/index.html>  
 (b) <http://www.bbk.ac.uk/study/international/index.htm>

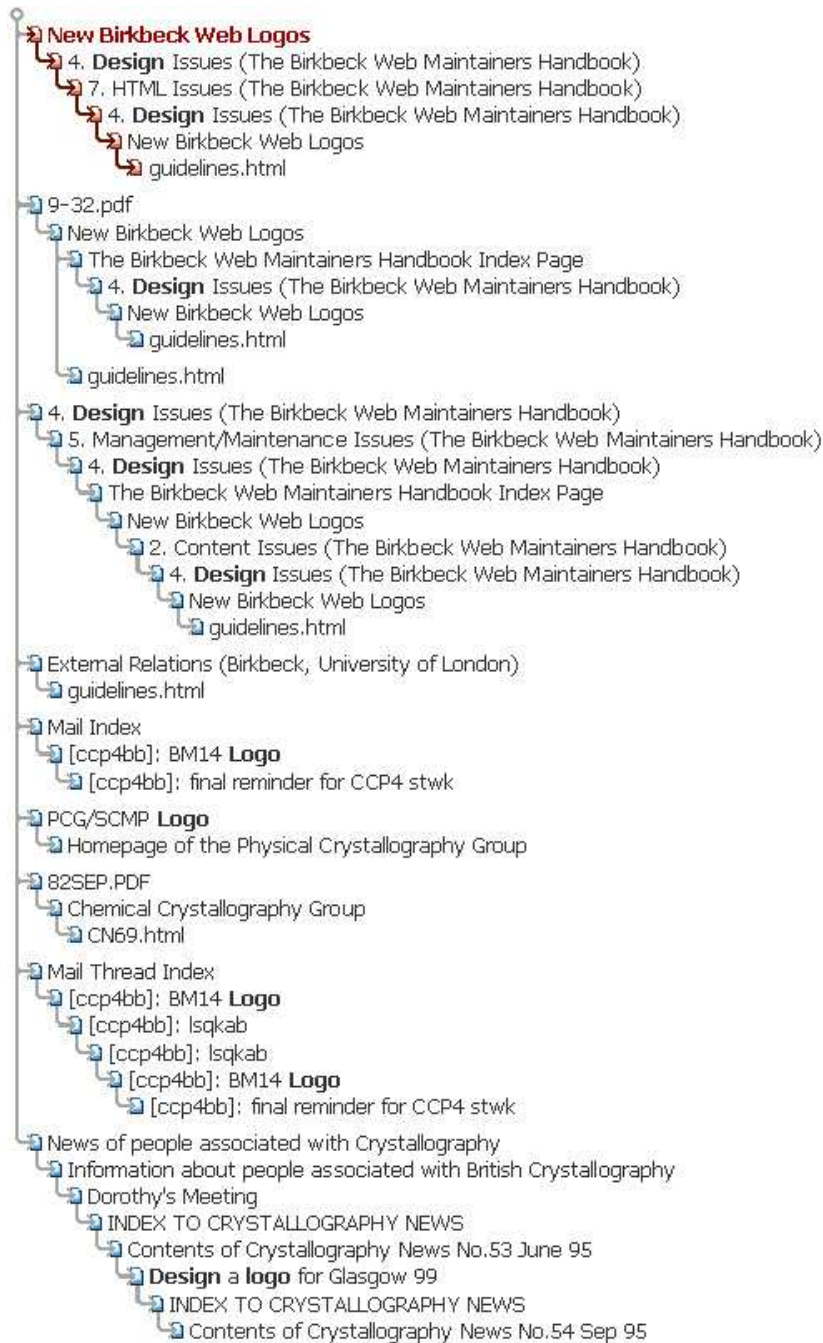


Figure 6.21: Trails found for the query “birbbeck logo design” on the Birkbeck site.

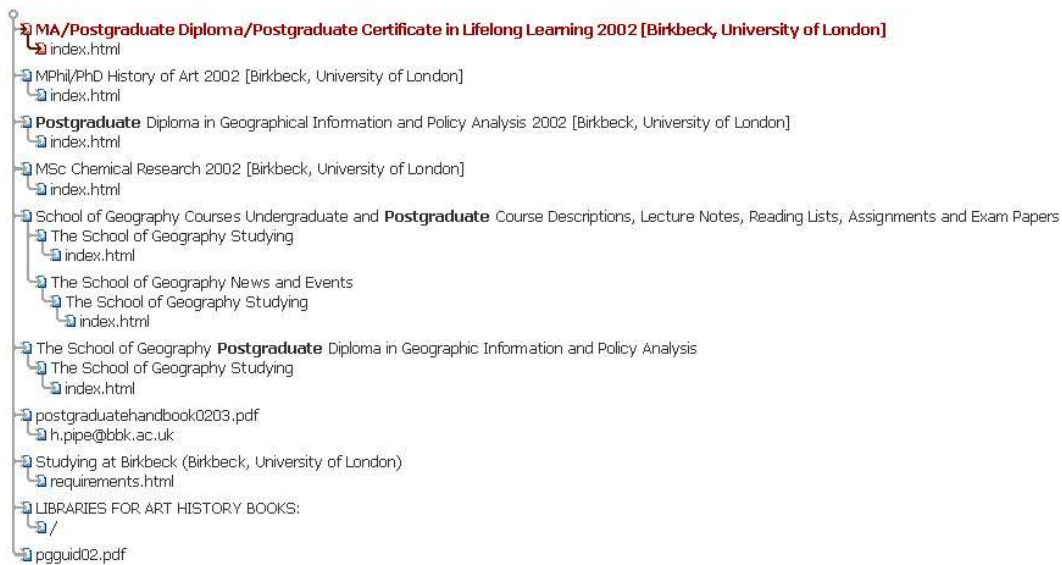


Figure 6.22: Trails found for the query “design postgraduate” on the Birkbeck site.

In contrast the navigation engine produced the trails shown in figure 6.24, which cover much of the needed information but contain very little content specifically for overseas students.

**writing up phd** The authored trails (shown below) provide details on regulations and guidelines for PhD student covering writing up and submission. The trails shown in figure 6.25 were found by the navigation engine and are less relevant, covering aspects of general writing not related to PhD theses. In a university setting, all three of the query terms might be candidates for use as stop words.

It is interesting to note that the last two trails both consisted of singleton trails (trails with only one page). In total, only three singleton trails were generated by the authors.

1. (a) <http://www.dcs.bbk.ac.uk/research/degrees.html>  
 (b) <http://www.dcs.bbk.ac.uk/research/phdregulations.html>  
 (c) <http://www.bbk.ac.uk/reg/regulations/pdf/PhDregs.pdf>
2. (a) <http://www.bbk.ac.uk/llc/al/pg/handbook/dissertation.html>
3. (a) [http://www.bbk.ac.uk/polSOC/courses/ba/sa\\_diss\\_guide.php](http://www.bbk.ac.uk/polSOC/courses/ba/sa_diss_guide.php)

**a student gym** Birkbeck college does not have a student gym. However, the University of London Union (ULU)<sup>9</sup> does. The three authored trails (shown below) explain this. The first shows a link to ULU from the Birkbeck page covering aspects of “London Life”. The link is broken, but the author surmises that the search should be able to work out that ULU is the place to look. The last two trails contain documents (undergraduateguide.pdf and induction1\_newstudents.pdf) which say that ULU has a

<sup>9</sup> <http://www.ululon.ac.uk/>

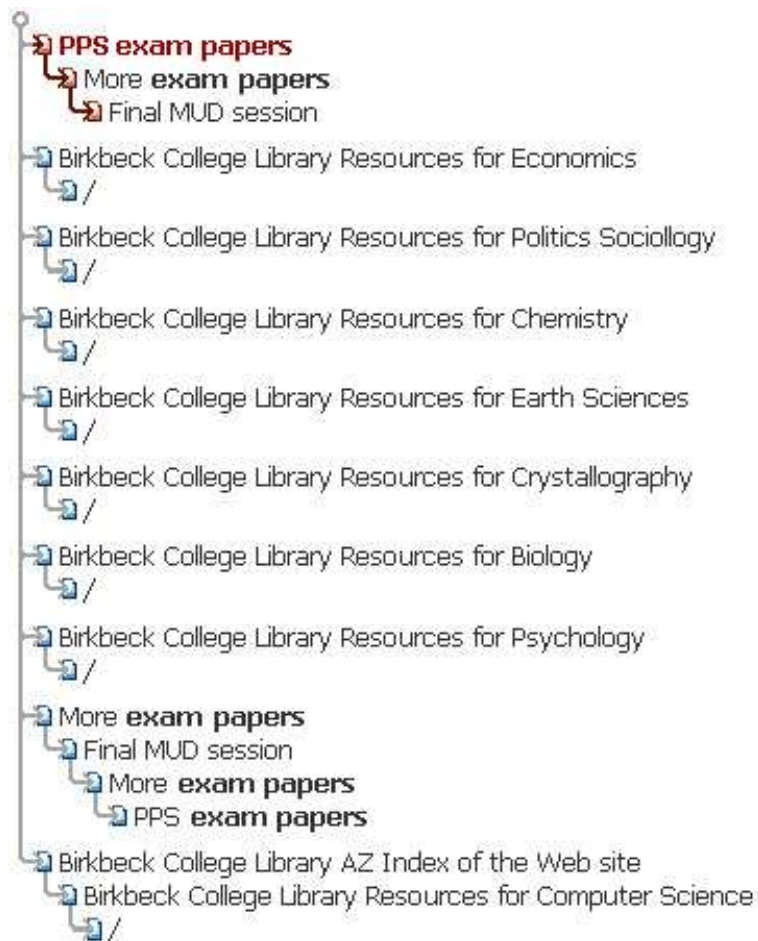


Figure 6.23: Trails found for the query “exam papers” on the Birkbeck site.

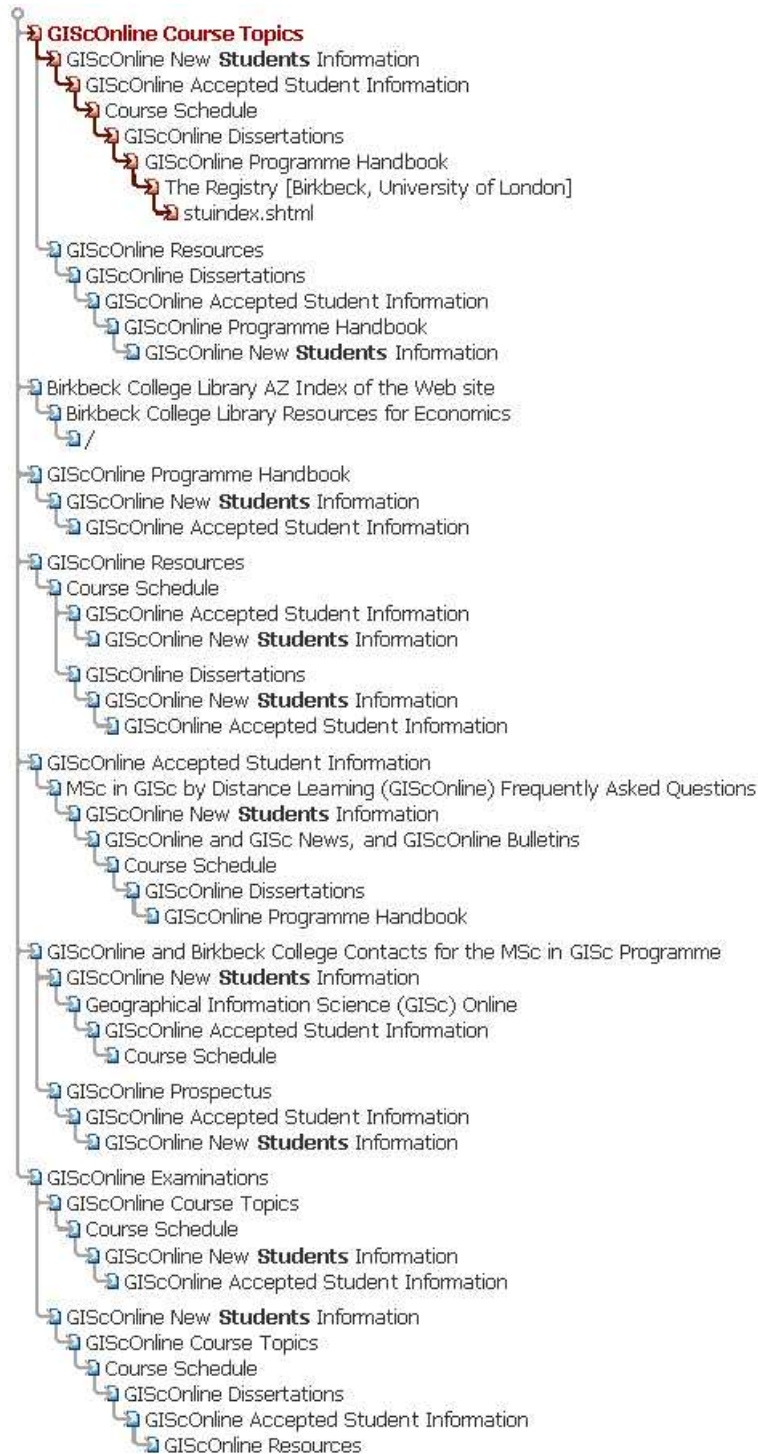


Figure 6.24: Trails found for the query “international students” on the Birkbeck site.

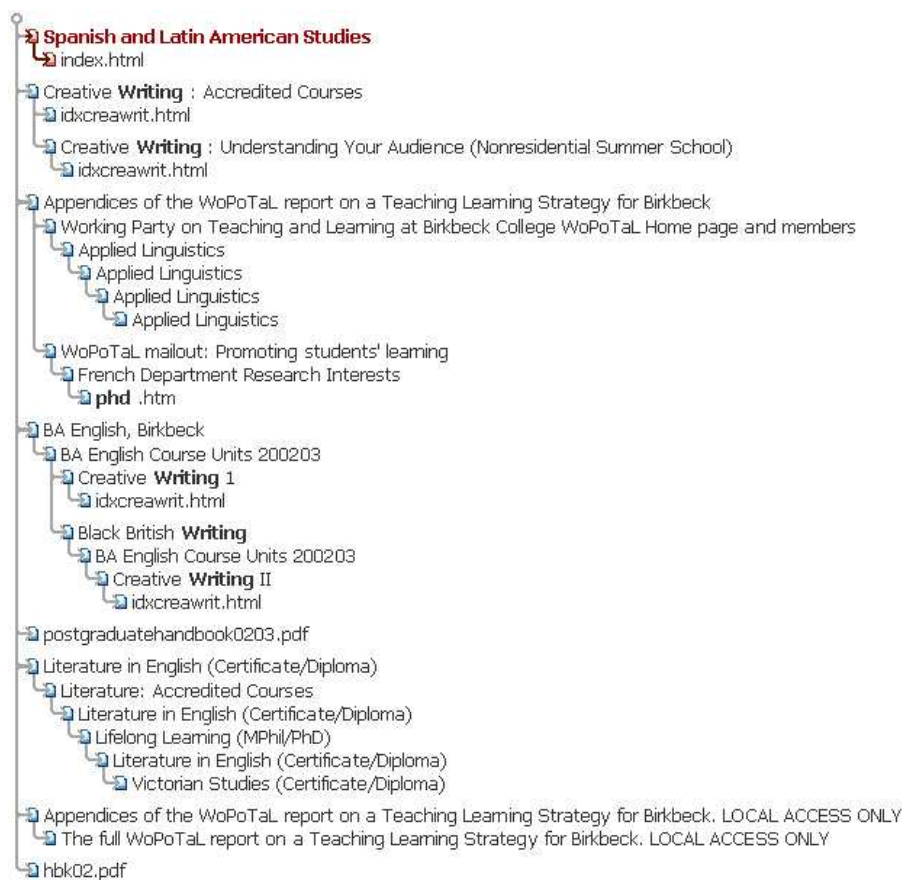


Figure 6.25: Trails found for the query “writing up phd” on the Birkbeck site.



gym. The first page in the second trail puts the second page in the context of the Geography dept.

1. (a) <http://www.bbk.ac.uk/lsp/london.htm>  
 (b) <http://www.ulu.ucl.ac.uk/>
2. (a) <http://www.bbk.ac.uk/geog/study.html>  
 (b) <http://www.bbk.ac.uk/geog/study/guides/undergraduateguide.pdf>
3. (a) [http://www.dcs.bbk.ac.uk/itapps/studenthandbook/induction1\\_newstudents.pdf](http://www.dcs.bbk.ac.uk/itapps/studenthandbook/induction1_newstudents.pdf)

In contrast the trails shown in figure 6.26 highlight the computer's inability to reason in this manner. The poor quality of results is an inevitable consequence of an "AI-complete" problem (Howe 1993). The term AI-complete denotes, by analogy with 'NP-complete', a problem which requires the synthesis of a human-level intelligence to generate a perfect answer. It is impossible for a computer to fully synthesize the intelligence used by a human author in generating trails.

**beginners painting** The trail author surmised that the person who posed the query was intending the question "is there a beginner's painting course at Birkbeck?". This is an interesting case as the answer the searcher really wants is "no". There is no such course available at Birkbeck. There is not even an art course or department which could be considered a close match. The closest offering would be a "history of art" course. The trail author acknowledged that it was "hard to say what combination of pages says "no" most clearly". It is possible that no trails at all might be the best result as no page positively answers the query. However, failure to return any results is obviously ambiguous. The trail author proposed the following trails:

1. (a) <http://www.bbk.ac.uk/index.shtml>  
 (b) <http://www.bbk.ac.uk/study/index.html>  
 (c) <http://www.bbk.ac.uk/foundation/foundation.html>  
 (d) <http://www.bbk.ac.uk/foundation/Subjects.html>
2. (a) <http://www.bbk.ac.uk/index.shtml>  
 (b) <http://www.bbk.ac.uk/study/index.html>
3. (a) <http://www.bbk.ac.uk/index.shtml>  
 (b) <http://www.bbk.ac.uk/study/index.html>  
 (c) <http://www.bbk.ac.uk/study/ug2003/index.html>

By showing the information for foundation degrees the first trail matches the concept of a "beginner's" course. The information on such degrees shows that painting is not an option. Although the second trail is also a prefix to the other trails, it is nonetheless useful in its own right. The last page has a pull-down list of course titles. The searcher can see from this that there is no painting course. The list of courses in the third trail also shows no painting course. Figure 6.27 shows the trails returned by the navigation engine which highlight the computer's inability to reason about how best to express a negative result.

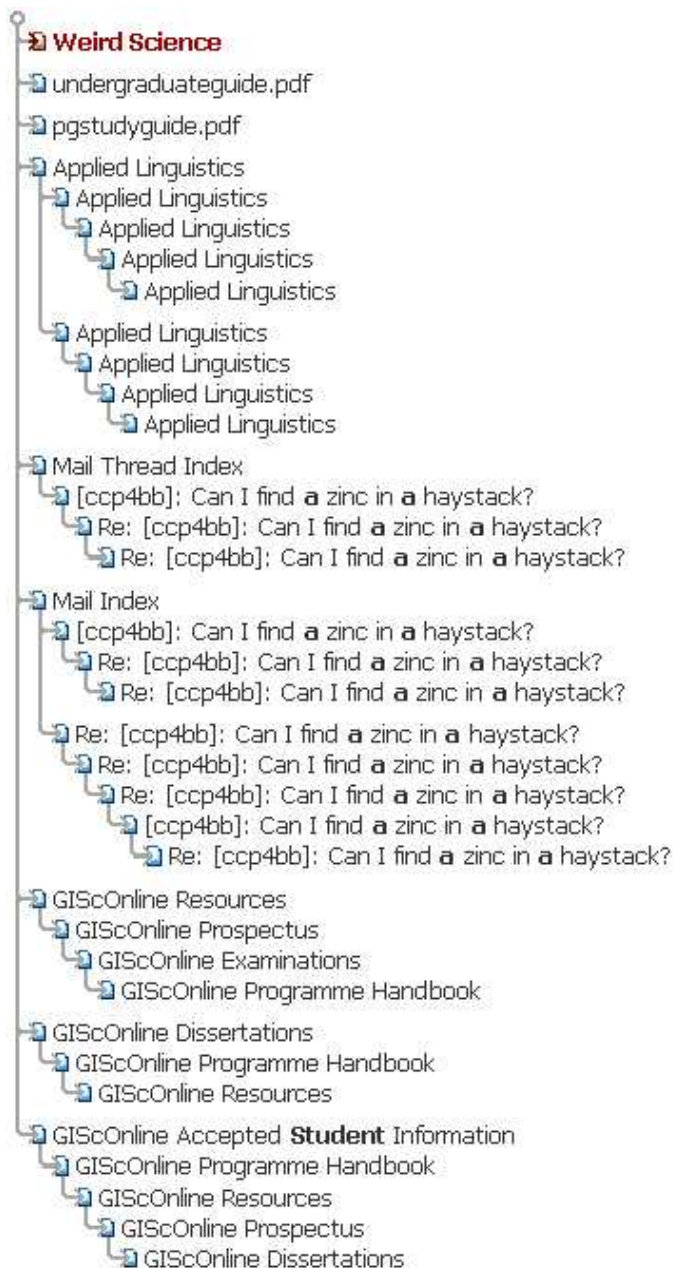


Figure 6.26: Trails found for the query “a student gym” on the Birkbeck site.

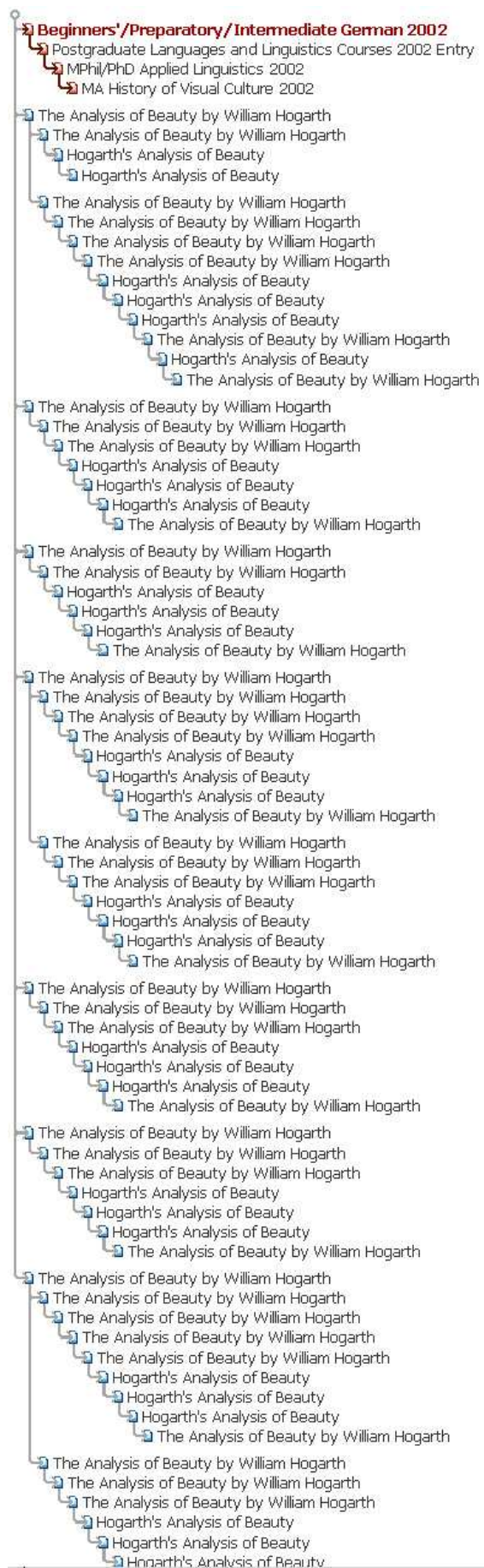


Figure 6.27: Trails found for the query “beginners painting” on the Birkbeck site.

**bsc programming** It is probable that the searcher was after an undergraduate course that would teach computer programming. The following authored trails show SCSIS courses which do exactly that:

1. (a) <http://www.dcs.bbk.ac.uk/>  
 (b) <http://www.dcs.bbk.ac.uk/courses/>  
 (c) <http://www.dcs.bbk.ac.uk/courses/ism/>  
 (d) <http://www.dcs.bbk.ac.uk/courses/ism/ism.html>
2. (a) <http://www.dcs.bbk.ac.uk/>  
 (b) <http://www.dcs.bbk.ac.uk/courses/>  
 (c) <http://www.dcs.bbk.ac.uk/courses/ism/>  
 (d) <http://www.bbk.ac.uk/study/ug2003/compsci/infosysbsc.html>
3. (a) <http://www.bbk.ac.uk>  
 (b) <http://www.bbk.ac.uk/study/index.html>  
 (c) <http://www.bbk.ac.uk/study/ug2003/index.html>  
 (d) <http://www.bbk.ac.uk/study/ug2003/compsci/infosysbsc.html>
4. (a) <http://www.bbk.ac.uk>  
 (b) <http://www.bbk.ac.uk/academic.html>  
 (c) <http://www.dcs.bbk.ac.uk/>  
 (d) <http://www.dcs.bbk.ac.uk/courses/>  
 (e) <http://www.dcs.bbk.ac.uk/courses/ism/>

The computed trails shown in figure 6.28 show details of computer science and physics courses. The physics courses teach computer programming and word processing alongside the expected subjects such as Nuclei and Elementary Particles, Cosmology and Quantum Mechanics. Because there are so many courses teaching computer programming run by the Physics department, the trails dominate those describing the SCSIS courses, although the latter are probably more relevant.

**dept of philosphy** This query supports the previous findings on the inability of users to spell and type consistently and accurately. The authored trails clearly show information about the department of philosophy, whilst the computed trails do not (below and figure 6.29). Although still not perfect, the results do improve when the correct query terms are used (figure 6.30).

1. (a) <http://www.bbk.ac.uk/phil/>  
 (b) <http://www.bbk.ac.uk/phil/staff.html>  
 (c) <http://www.bbk.ac.uk/phil/academics.html>
2. (a) <http://www.bbk.ac.uk/phil/>  
 (b) <http://www.bbk.ac.uk/phil/students.html>  
 (c) [http://www.bbk.ac.uk/phil/ma\\_handbook.html](http://www.bbk.ac.uk/phil/ma_handbook.html) (Masters handbook, but has a section of info. on the department)
3. (a) <http://www.bbk.ac.uk/study/ug2003/>  
 (b) <http://www.bbk.ac.uk/study/ug2003/philosophy/philba.html>

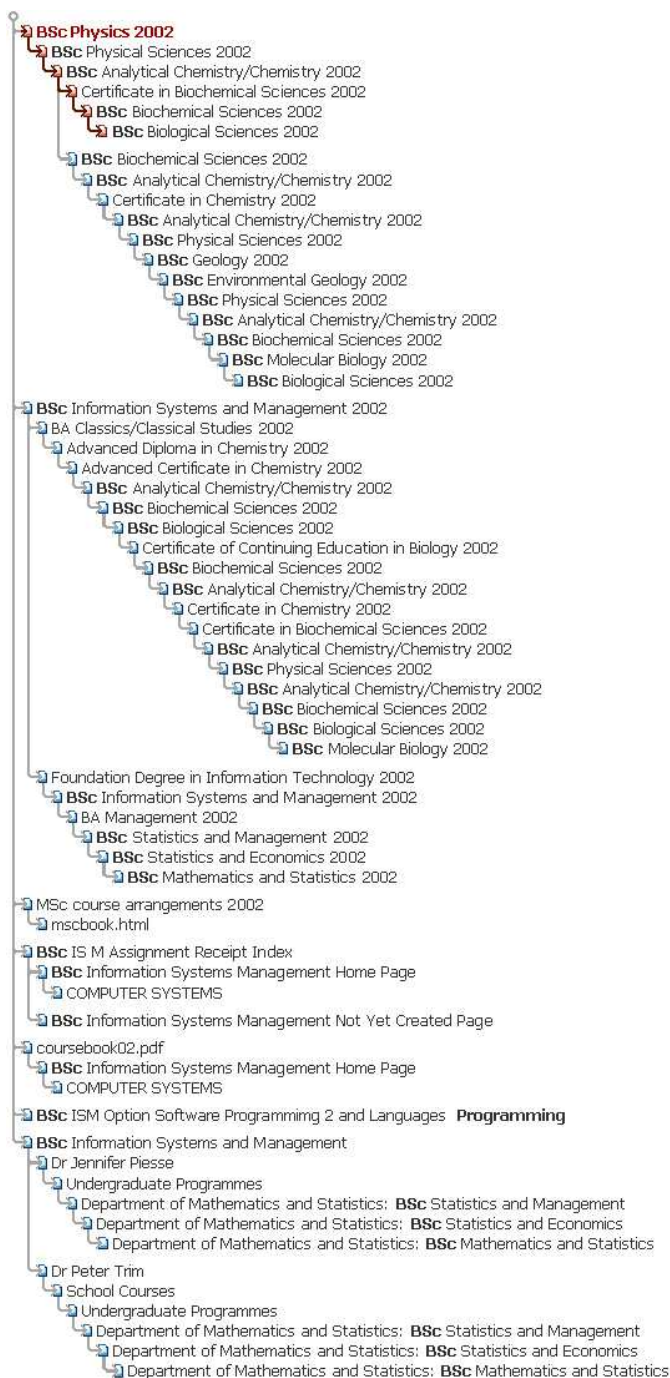


Figure 6.28: Trails found for the query “bsc programming” on the Birkbeck site.

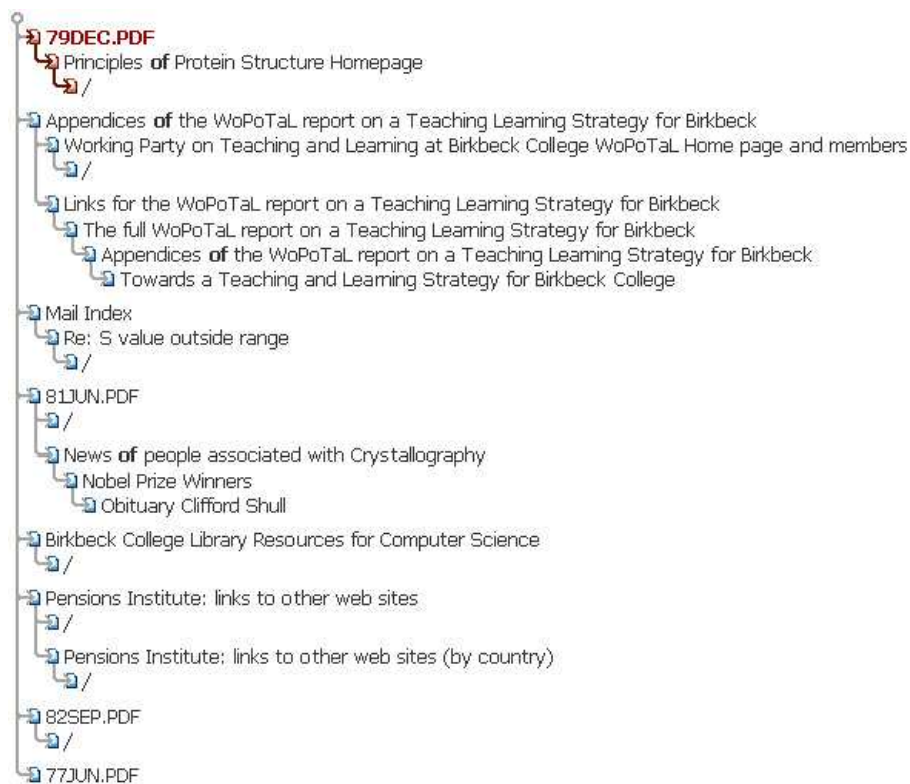


Figure 6.29: Trails found for the query “dept of philosophy” on the Birkbeck site.

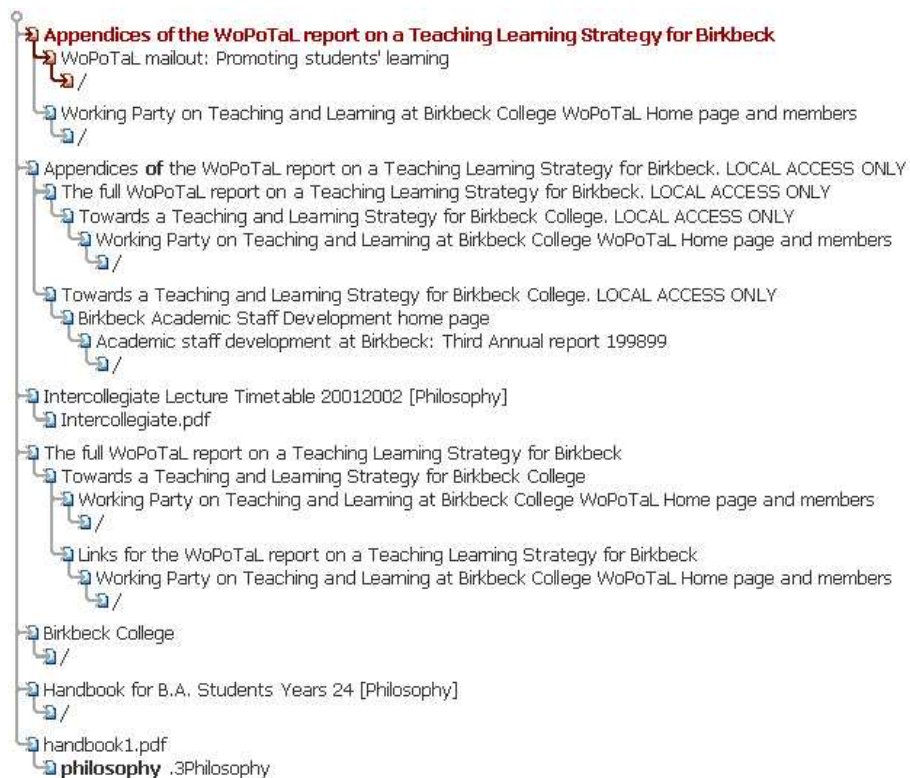


Figure 6.30: Trails found for the query “dept of philosophy” on the Birkbeck site.

**distance learning** The computed trails in figure 6.31 are dominated by the pages on Geography and Geographical Information Science (GISc) courses. A comprehensive set of short trails was authored to show the level of distance learning support available for many subjects:

1. (a) <http://www.bbk.ac.uk/study/>  
 (b) <http://www.bbk.ac.uk/distance/>
2. (a) <http://www.bbk.ac.uk/lib/>  
 (b) <http://www.bbk.ac.uk/lib/info.html>  
 (c) <http://www.bbk.ac.uk/lib/dls/dls2.html>
3. (a) <http://www.bbk.ac.uk/es/dlearn/dlearn.htm>  
 (b) <http://www.bbk.ac.uk/es/dlearn/pg5.htm>  
 (c) <http://www.bbk.ac.uk/es/dlearn/pg4.htm>
4. (a) <http://www.bbk.ac.uk/geog/>  
 (b) <http://www.bbk.ac.uk/gisconline/>  
 (c) <http://www.bbk.ac.uk/gisconline/coursetopics.html>
5. (a) <http://www.bbk.ac.uk/study/fce2002/mediastudies/medsc.html>  
 (b) <http://www.bbk.ac.uk/study/fce2002/mediastudies/medacc.html#c01>
6. (a) <http://www.bbk.ac.uk/study/ug2003/index.htm>  
 (b) <http://www.bbk.ac.uk/study/ug2003/cryst/strucmolacert.html>
7. (a) <http://www.bbk.ac.uk/study/ug2003/earthsci/idxeearthsci.html>  
 (b) <http://www.bbk.ac.uk/study/ug2003/earthsci/geolstracert.html>

**mba** The first two authored trails describe the MBA and “Mini MBA” courses. The last trail describes the MSc Finance course - a viable alternative to an MBA for many people. The last document states that the course aims to “avoid the cookbook treatment common in MBA or less academic masters programmes”.

1. (a) <http://www.bbk.ac.uk/study/fce2002/>  
 (b) <http://www.bbk.ac.uk/study/fce2002/management/idxmanagement.html>  
 (c) <http://www.bbk.ac.uk/study/fce2002/management/manamba.html>
2. (a) <http://www.bbk.ac.uk/study/fce2002/>  
 (b) <http://www.bbk.ac.uk/study/fce2002/management/idxmanagement.html>  
 (c) <http://www.bbk.ac.uk/study/fce2002/management/manad.html>
3. (a) <http://www.econ.bbk.ac.uk/>  
 (b) <http://www.econ.bbk.ac.uk/courses/list.htm>  
 (c) <http://www.econ.bbk.ac.uk/courses/mscfinance/mscfin.htm>  
 (d) <http://www.econ.bbk.ac.uk/courses/mscfinance/AMScFIN02.pdf>

The computed trails, shown in figure 6.32 cover similar ground except for not identifying the MSc Finance course as a potential alternative. This is probably desirable in the general case. Anecdotal evidence has suggested that users of conventional search engines are often frustrated if sponsored links and advertisements for competitive products take an overly prominent position in the search results.



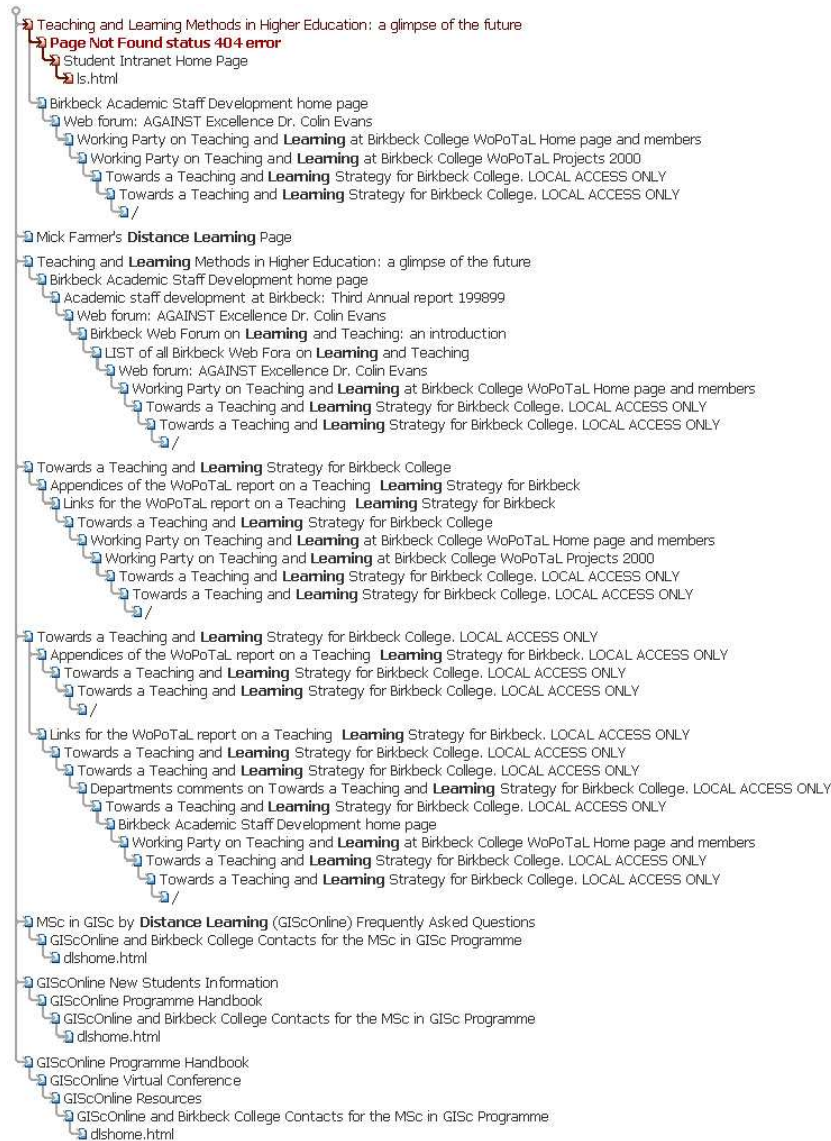


Figure 6.31: Trails found for the query “distance learning” on the Birkbeck site.



Figure 6.32: Trails found for the query “mba” on the Birkbeck site.

**access 97** The trails below address the facilities for using, Microsoft Access '97. The last page of the first trail was included as the “logical next step for a human”. The trail author reasoned that “once the user knows the course on offer, it is likely they’d want the booking instructions”. Interesting comparisons can also be made between these trails and those produced for the query “access course”. The two sets were produced by different authors, yet contain similar pages.

1. (a) <http://www.econ.bbk.ac.uk/helpdesk/stftrain/default.htm>  
 (b) <http://www.econ.bbk.ac.uk/helpdesk/stftrain/package.htm>  
 (c) <http://www.econ.bbk.ac.uk/helpdesk/stftrain/dbase.htm>  
 (d) <http://www.econ.bbk.ac.uk/helpdesk/stftrain/booking.htm>
2. (a) <http://www.bbk.ac.uk/ccs/docs/docs.htm>  
 (b) <http://www.bbk.ac.uk/ccs/docs/docs.htm#ToC5>  
 (c) <http://www.bbk.ac.uk/ccs/docs/5-69.pdf>
3. (a) <http://www.bbk.ac.uk/ccs/docs/docs.htm>  
 (b) <http://www.bbk.ac.uk/ccs/docs/docs.htm#ToC5>  
 (c) <http://www.bbk.ac.uk/ccs/docs/5-70.pdf>

Other comments made by the trail author whilst producing these trails indicate that the use of a global search such as Google is still preferable to the site search facilities offered by Birkbeck. The quality of site search indicates that there is still a large potential market for the navigation engine approach if the technical problems can be addressed.

The trails in figure 6.33 show similar issues to the “access course” results. In addition there is a problem handling numbers such as 97 - a feature which will be addressed in more detail in section 7.10.

**project management** The first authored trail for this query covers the opening pages of a set of lecture notes for the course on Comparative Development Methodologies (CDM). These early pages cover project management tools. Later pages digress into database systems which are of no relevance to the query and are thus excluded. The second trail covers the project management aspects of the “Malet Street Project”.

1. (a) <http://www.dcs.bbk.ac.uk/steve/cdm4/index.htm>
- (b) <http://www.dcs.bbk.ac.uk/steve/cdm4/sld001.htm>
- (c) <http://www.dcs.bbk.ac.uk/steve/cdm4/sld002.htm>
- (d) <http://www.dcs.bbk.ac.uk/steve/cdm4/sld003.htm>
- (e) <http://www.dcs.bbk.ac.uk/steve/cdm4/sld004.htm>
- (f) <http://www.dcs.bbk.ac.uk/steve/cdm4/sld005.htm>
- (g) <http://www.dcs.bbk.ac.uk/steve/cdm4/sld006.htm>
- (h) <http://www.dcs.bbk.ac.uk/steve/cdm4/sld007.htm>
- (i) <http://www.dcs.bbk.ac.uk/steve/cdm4/sld008.htm>
2. (a) <http://www.bbk.ac.uk/msp/intro.html>
- (b) <http://www.bbk.ac.uk/msp/description.html>
- (c) <http://www.bbk.ac.uk/msp/background.html>
- (d) <http://www.bbk.ac.uk/msp/team.html>
- (e) <http://www.mcbainscooper.co.uk/>

The constructed trails, shown in figure 6.34 cover general management issues. Proximity-based IR would help narrow the search in this instance.

**part time** A single trail was authored in response to this query, covering Robin Middlehurst’s talk on “Part-time study: now and in the future”. The trail, which is shown below, is complete and comprehensive. In contrast the trails shown in figure 6.35 are again dominated by GISc pages.

1. (a) <http://www.bbk.ac.uk/asd/joint/joint2.html>
- (b) <http://www.bbk.ac.uk/asd/joint/index.html>
- (c) <http://www.bbk.ac.uk/asd/joint/robin/index.htm>
- (d) <http://www.bbk.ac.uk/asd/joint/robin/sld001.htm>
- (e) <http://www.bbk.ac.uk/asd/joint/robin/sld002.htm>
- (f) <http://www.bbk.ac.uk/asd/joint/robin/sld003.htm>
- (g) <http://www.bbk.ac.uk/asd/joint/robin/sld004.htm>
- (h) <http://www.bbk.ac.uk/asd/joint/robin/sld005.htm>
- (i) <http://www.bbk.ac.uk/asd/joint/robin/sld006.htm>
- (j) <http://www.bbk.ac.uk/asd/joint/robin/sld007.htm>
- (k) <http://www.bbk.ac.uk/asd/joint/robin/sld008.htm>

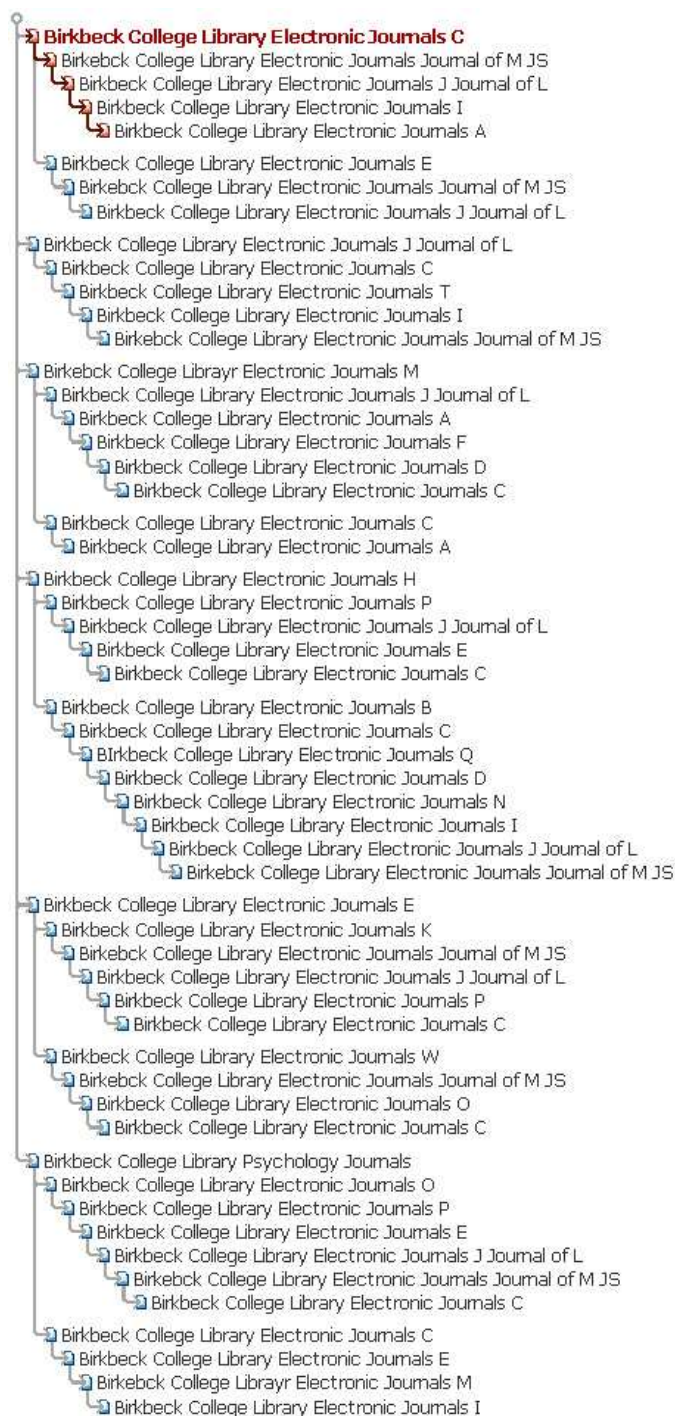


Figure 6.33: Trails found for the query “access 97” on the Birkbeck site.



Figure 6.34: Trails found for the query “project management” on the Birkbeck site.

- (l) <http://www.bbk.ac.uk/asd/joint/robin/sld009.htm>
- (m) <http://www.bbk.ac.uk/asd/joint/robin/sld010.htm>
- (n) <http://www.bbk.ac.uk/asd/joint/robin/sld011.htm>
- (o) <http://www.bbk.ac.uk/asd/joint/robin/sld012.htm>
- (p) <http://www.bbk.ac.uk/asd/joint/robin/sld013.htm>
- (q) <http://www.bbk.ac.uk/asd/joint/robin/sld014.htm>
- (r) <http://www.bbk.ac.uk/asd/joint/robin/sld015.htm>
- (s) <http://www.bbk.ac.uk/asd/joint/robin/sld016.htm>
- (t) <http://www.bbk.ac.uk/asd/joint/robin/sld017.htm>

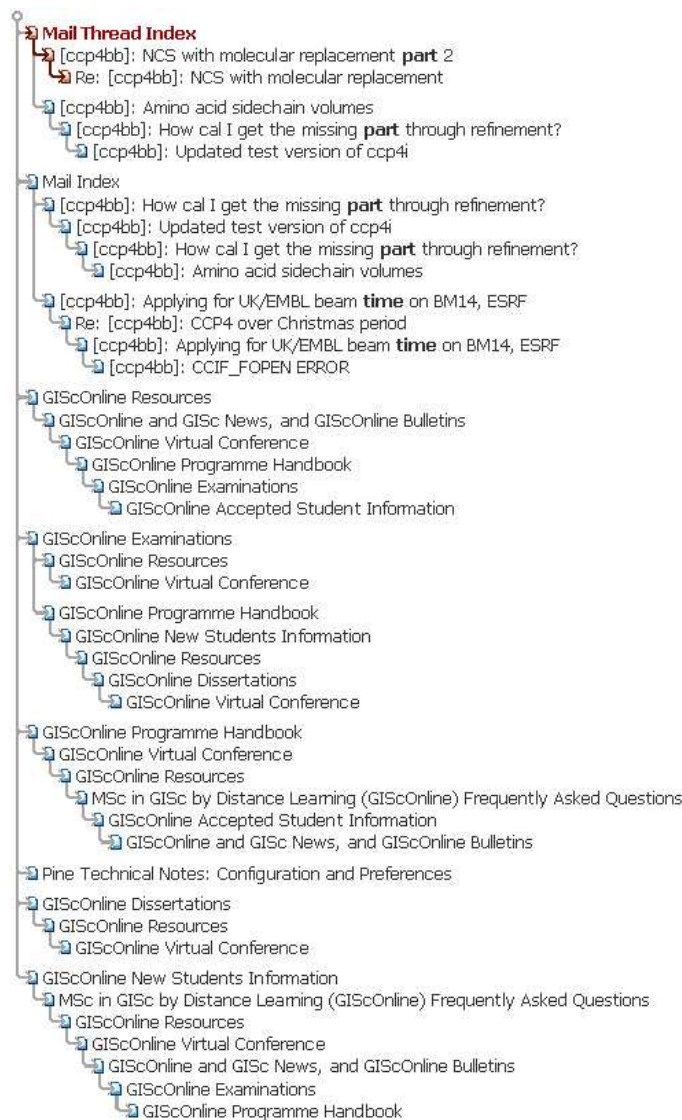


Figure 6.35: Trails found for the query “part time” on the Birkbeck site.

**research grants history** The authored trail shows the page for the Research and Grants Office, in the context of the “Fees and Grants” section of the History, Classics and Archaeology department research page. The first computed trail also highlights this key page, but does not identify the specific section. The computed trails also show the main “Research at Birkbeck” page which also has a section on grants and fees. Furthermore, the computed trails show further details of history research which is likely to be pertinent.

1. (a) <http://www.bbk.ac.uk/hca/research/research.htm>
- (b) <http://www.bbk.ac.uk/hca/research/research.htm#Fees%20and%20Grants>
- (c) <http://www.bbk.ac.uk/hca/noticeboard.htm#Awards>
- (d) <http://www.bbk.ac.uk/res/rgco.html>

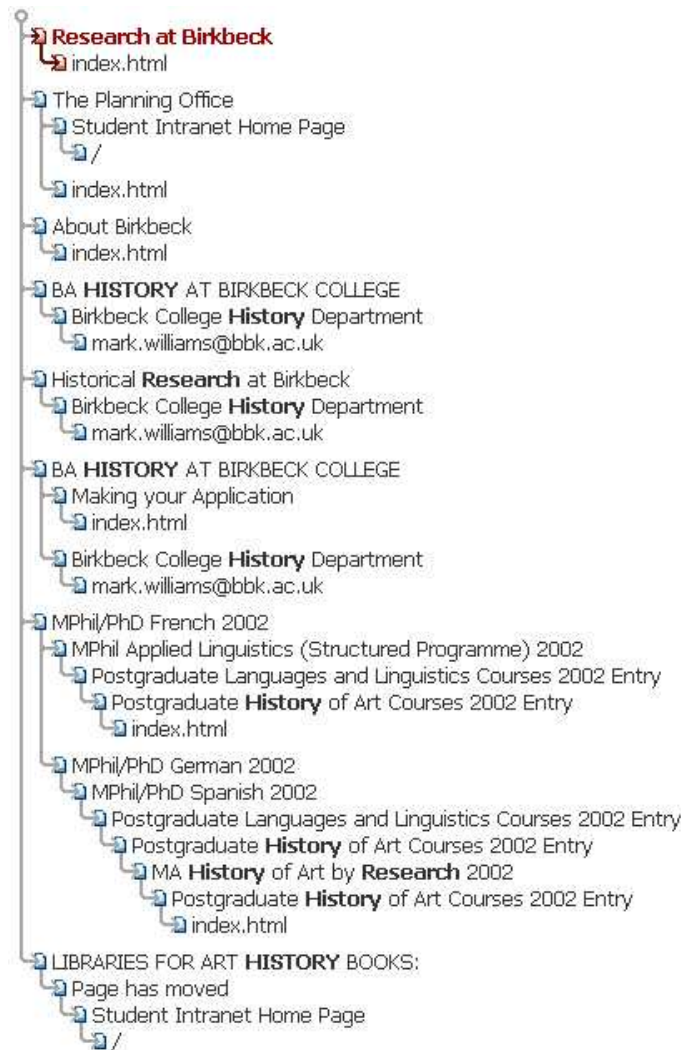


Figure 6.36: Trails found for the query “research grants history” on the Birkbeck site.

**social** The trails for this query could contain pages regarding social life at Birkbeck or pages relating to social sciences. However, the only authored trail contains pages relating to social policy within the Faculty of Continuing Education (FCE). The computed trails shows literature relating to social science in the Birkbeck Library.

1. (a) <http://www.bbk.ac.uk/study/fce/index.html>
- (b) <http://www.bbk.ac.uk/study/fce/socpolicy/idxsocpolicy.html>
- (c) <http://www.bbk.ac.uk/study/fce/socpolicy/socpolcd.html>
- (d) <http://www.bbk.ac.uk/study/fce/socpolicy/socpolcd.html#c01>

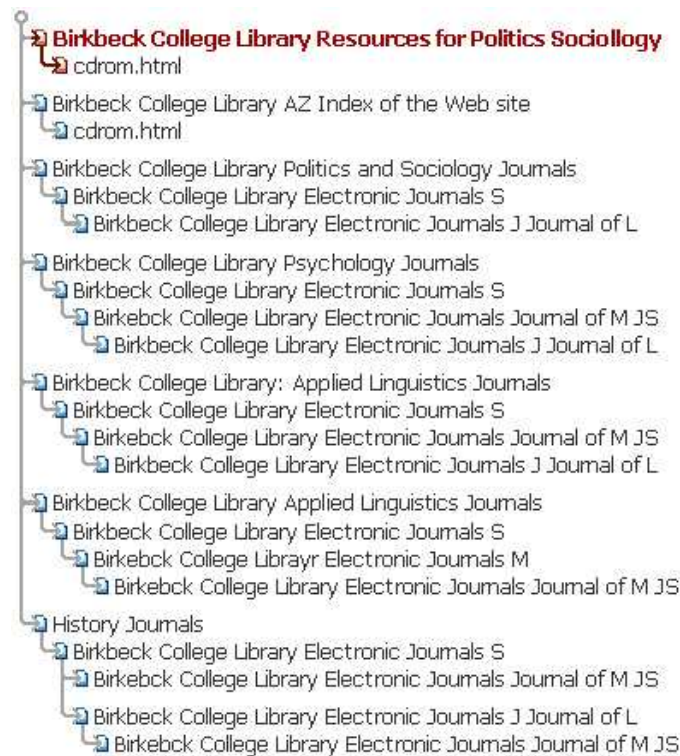


Figure 6.37: Trails found for the query “social” on the Birkbeck site.

**ba history** Only a single trail was authored for this query, and although nine links are present, there are only two pages identified. The computed trails identify different pages covering the same material, and identify alternative undergraduate humanities subjects, including the history of art course, which is most likely to be of interest.

1. (a) <http://www.bbk.ac.uk/hca/courses/bahist.htm>
- (b) <http://www.bbk.ac.uk/hca/courses/bahist.htm#bahist>
- (c) <http://www.bbk.ac.uk/hca/courses/bahist.htm#entrance>
- (d) <http://www.bbk.ac.uk/hca/courses/bahist.htm#apply>
- (e) <http://www.bbk.ac.uk/hca/courses/bahist.htm#degreeprog>
- (f) [http://www.bbk.ac.uk/hca/BA\\_Course\\_Descriptions/BA\\_History\\_courses.htm](http://www.bbk.ac.uk/hca/BA_Course_Descriptions/BA_History_courses.htm)



- (g) [http://www.bbk.ac.uk/hca/BA\\_Course\\_Descriptions/BA\\_History\\_courses.htm#group1](http://www.bbk.ac.uk/hca/BA_Course_Descriptions/BA_History_courses.htm#group1)
- (h) [http://www.bbk.ac.uk/hca/BA\\_Course\\_Descriptions/BA\\_History\\_courses.htm#group2](http://www.bbk.ac.uk/hca/BA_Course_Descriptions/BA_History_courses.htm#group2)
- (i) [http://www.bbk.ac.uk/hca/BA\\_Course\\_Descriptions/BA\\_History\\_courses.htm#group3](http://www.bbk.ac.uk/hca/BA_Course_Descriptions/BA_History_courses.htm#group3)

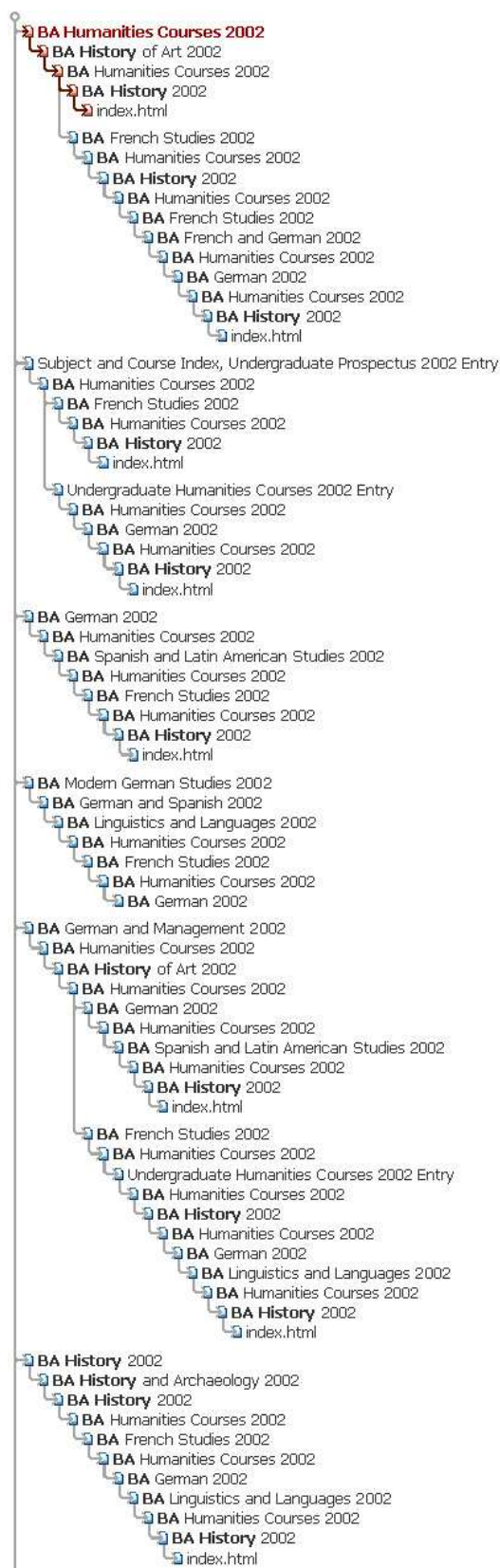


Figure 6.38: Trails found for the query “ba history” on the Birkbeck site.

### 6.6.3 Conclusions

From the results of this comparison, the following conclusions have been drawn:

1. When key pages can be identified which satisfy the users information need and can be judged relevant using IR measures, these appear frequently in both the authored and computed trails. Otherwise, there is little overlap between the content of authored and computed trails, although the quality of both is generally high.
2. It has been suggested that the computed trails are too long and need to be shortened. Figure 6.39 shows the distribution of Trail Lengths for authored and computed trails. For authored trails, the mean, mode and median lengths are 3.65, 3 and 3, respectively. For the computed trails, the mean, mode and median lengths are 4.38, 2 and 4, respectively. From this direct comparison it can be seen that, in general, the computed trails are not significantly longer, but may be so in certain specific cases.
3. From the reports of the trail authors, it can be seen that the effort which must be expended in authoring trails is a significant impediment to the adoption of any system designed exclusively for manipulating authored trails. For this reason, the use of computed generated trails seems highly desirable.
4. In order to be of maximum value, constant improvement of all the components needs to be made to keep up with the current state of the art. For example, recent advances have been made in improving Web page summarization techniques (Over and Yen 2003; Yang and Wang 2003) and in facilitating corporate intranet search (Fagin, Kumar, McCurley, Novak, Sivakumar, Tomlin, and Williamson 2003). Advances have also been made in tackling the logistical challenges of search engine development. Recent progress has been made in identifying techniques for updating indexes (Lim, Wang, Padmanabhan, Vitter, and Agarwal 2003) and for representing large Web graphs (Boldi and Vigna 2003). Combined with better information retrieval, some of these techniques may be effective in improving the overall quality of the system.

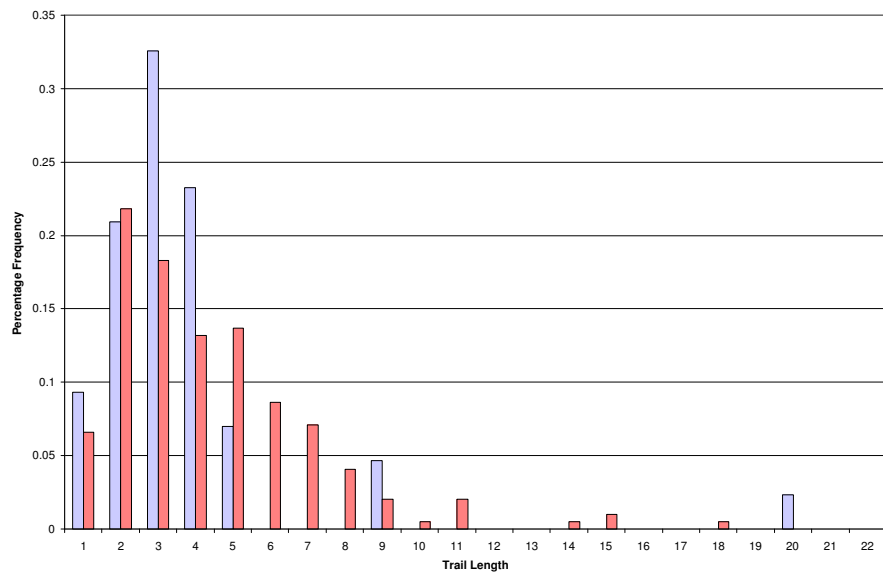


Figure 6.39: Distribution of Trail Lengths. Blue bars show the percentage of authored trails which are of each length. Red bars show the percentage of computed trails of those lengths.

## 6.7 Scaling to the Web

Although site-search is of vital importance, and deserves special attention as an area of research separate from global search engines, it would be highly beneficial to allow full Web-scale trail finding. Unfortunately, the current architecture will not scale to full-size Web data. However, the problem can be broken down. Conventional search engines do not index the full content of the Web. They select some subset to index based on usage statistics, link analysis or the output of dedicated crawling algorithms designed to select high-quality nodes first (Pinkerton 2002; Cho, Garcia-Molina, and Page 1998). A subset of this can be selected on which to perform trail computation. For example, trail information could be computed on high-profile or highly-popular sites such as yahoo.com, cnn.com or bbc.co.uk whilst single-page results are returned for the remaining indexed pages. An alternative strategy is to construct a restricted graph based upon the search results for a given query, over which trails could be constructed. Whilst this approach would suffer less scalability problems, it might suffer similar performance issues to Kleinberg's approach of expanding the search results (Kleinberg 1998).

Whilst multiple servers can easily be used for load balancing without affecting the underlying application structure, it might be necessary to split a corpus across multiple servers. Distribution of the index is needed if and when it becomes too large to efficiently fit on a single machine. Most search engine architectures rely on distribution across a large number of machines, although the exact number of servers varies from dozens to thousands (Risvik and Michelsen 2002). The techniques described in this section offer ideas for how this distribution can be achieved. These techniques have been implemented and are known to work. Unfortunately, a meaningful evaluation of their performance would require a large multi-server network on which comprehensive tests could be performed.

### 6.7.1 Splitting the Index

The two most popular approaches to distribution are to split the collection by keyword or by document. Splitting by document implies that for any given document the scores for all possible keywords are stored on the same physical machine. On the other hand, splitting by keyword means that all the scores for any given keyword are similarly co-located, irrespective of which document they originated in.

One suggested advantage of splitting a corpus by keyword is that a full ranking can be achieved with the results from a single server. Hence questions can be answered by a single server for many queries. The first counter argument to this is that a full ranking should not be used wherever possible. For example, if only the top 10 documents are required, and the query consists of a single keyword then only 10 entries need be looked at. The second counter argument is that for many queries it will be necessary to transfer the entire index of the least-popular term. The effects of this can be minimised by clustering keywords based upon in-query popularity. For example, the entry for "britney" will be on the same physical machine as the entry for "spears". The suggested way to build a split by keyword index is to send each keyword to be processed by a different server, and use a central server to compute normalization factors.

One of the supposed advantages of splitting the collection by document is that the repeated querying of a popular term will not cause the failure of a particular server. The argument against this claim is that popular terms are likely to be included in the most popular queries and are likely to hit both page and disk caches a high percentage of the time. A second claim is that if using split by keyword, the removal of a single server can leave some queries impossible to answer. The argument against this claim is that the server shutdown should remain unnoticed as it will be compensated for by redundant servers. It should also be noted that both Google and AltaVista use an index which is split by document, as do the Stanford Database group in their Berkeley-based system (Hirai, Raghavan, Paepcke, and Garcia-Molina 2000).

The suggested way to build a split by document index is to send each document to be processed by a different server, and maintain a single server for computing *tf.idf* values. For example, all temporary files are processed and sorted before the aggregation begins. This operation is then performed in such a way as to maintain integrity across the collection. The simplest way to do this is to aggregate the collection whilst spawning events for each distinct keyword which tell the control server to update the keyword *idf* score. Once this is done the aggregated file is re-processed to weight the scores accordingly, before finally building the B-tree as before.

An extension to the split-by-document principle is to split the index according to logical clusters, such as Web sites. Whilst there is no advantage in indexing time and a slight disadvantage in query time, the returned trails should be contained in the indexes of a single server or few servers. Trails crossing server boundaries can be constructed by forming a new trailset of the concatenations of other trails.

### 6.7.2 Graph Partitioning

If the system requires an in-memory graph, then the index must be split to match the graph. If the corpus is spread across many domains with relatively low levels of interlinking then it may be possible to use a variation of the domain hashing algorithm used by the Mercator Web crawler (Heydon and Najork 1999a). If this is not the case then two new algorithms are required – one to split the graph, and a second algorithm to combine the results. Natural assumptions for splitting the graph are that the algorithm should:

1. Keep the number of documents on each server as close to equal as possible.
2. Minimise the likelihood of useful trails being missed due to the need to cross servers.

This can be achieved if a partitioning scheme can be identified for the graph  $G = (N, E)$  such that  $G$  is split into a set of  $n$  graphs  $S = \{G_1 = (N_1, E_1), G_2 = (N_2, E_2) \dots G_n = (N_n, E_n)\}$  such the following parameters  $\alpha$  and  $\beta$  are minimized:

1.  $\alpha = \sum_{k=1}^n (|N_k|)^2$
2.  $\beta = (|\{x, y | (x, y) \in E \wedge \neg(\exists k x \in N_k \wedge y \in N_k)\}|)^2$

It is *not* required that the subgraphs be free of overlap. Overlapping graphs may be required to keep the number of split links down. However, it *is* required that all nodes be contained in at

least one subgraph. The first constraint keeps the sets from growing too large, or too skewed. The second constraint reduces the number of missing links to a minimum. Figure 6.41 shows how four partitions of the graph in figure 6.40 would affect the values of  $\alpha$  and  $\beta$ . Experiments will be required to determine the relative importance of these measures.

Flake et al. proposed a method for identifying communities using a variation of the max-flow, min-cut method, where they defined a web *community* as a collection of pages such that each member page has more hyperlinks (in either direction) within the community than to or from pages outside of the community (Flake, Giles, and Lawrence 2000; Flake, Lawrence, Giles, and Coetzee 2002). A variation of this technique should allow the construction of the appropriate graphs, considering that during the partitioning process the graph may be larger than at query time, due to lack of caching, index files, metadata and other memory constraints.

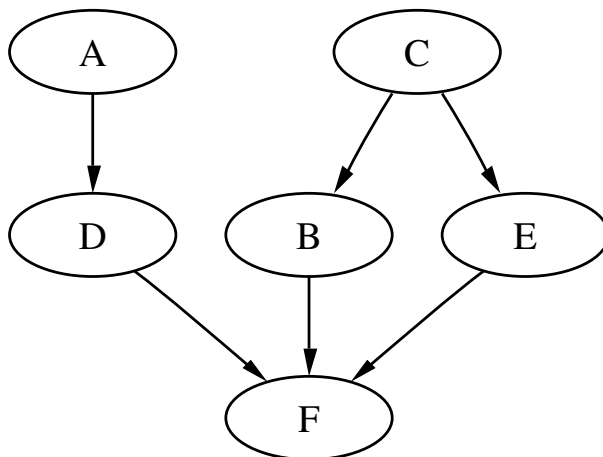


Figure 6.40: Example graph for partitioning.

Node Sets	Edges Cut	$\alpha$	$\beta$
$\{A, D\}$ $\{B, C, F, E\}$	$\{(D, F)\}$	20	1
$\{A, D, F\}$ $\{B, C, E\}$	$\{(F, B), (F, E)\}$	18	4
$\{A, B, C\}$ $\{D, E, F\}$	$\{(A, D), (B, F), (C, E)\}$	18	9
$\{A, D, F\}$ $\{B, C, F, E\}$		25	0

Figure 6.41: Example of the effect of various partitioning schemes on the graph shown in figure 6.40.

### 6.7.3 Merging Results

Trails computed on multiple graphs must be merged together before being presented to the user. In order to achieve this, the concatenations of trails within the two webcases must be considered. To merge two trailsets  $T_1$  and  $T_2$  which are valid across a combined Web Graph

$G = \langle E, V \rangle$ , a new trailset  $T$  should be created such that:

$$T = T_1 \cup \{xy \mid x \in T_1, y \in T_2, (x|_x, y_1) \in E\} \cup \{xyz \mid xy \in T_1, yz \in T_2, y \neq \emptyset\} \\ \cup T_2 \cup \{xy \mid x \in T_2, y \in T_1, (x|_x, y_1) \in E\} \cup \{xyz \mid xy \in T_2, yz \in T_1, y \neq \emptyset\} \quad (6.1)$$

It may not be assumed that any of the trails in  $T$  would have appeared in the candidate set for a combined Web Graph, but the results should provide a reasonable approximation.

It should be noted that subsequent reduction of the trails will remove all repeated subtrails from within a trail as long as the resulting trail remains valid with respect to the Web Graph  $G$ . Hence, only one string need be taken from the set  $\{xyz \mid xy \in T_1, yz \in T_2, y \neq \emptyset\}$ . This makes implementation easier and limits the size of  $T$  to no more than  $3|T_1||T_2|$ .

The importance of a webcase with respect to a query can be defined as a number in the range 0 to 1. The importance of each webcase will depend on many factors:

**Size** The more documents contained in a webcase, the more trails are likely to originate from this webcase and hence the higher the importance ranking.

**Focus** How focused is each webcase on each query term. High *idf* values imply low numbers of matching documents and a low number of documents relevant to the query within the webcase.

**Freshness** More recently updated webcases should be judged as more important.

The first 2 factors combined by weighting according to the occurrence of each term (or log of each term's occurrence) as a ratio/proportion of the total for all webcases.

Two functions, called  $f_1$  and  $f_2$  in the example, are required to calculate the number of trails to be computed on each source and the weight to assign to any given trail based upon that importance and the original trail score. If the trail scores are such that keyword frequencies are normalized across the sources, this task is made significantly easier.

If the webcases have been constructed with identical *tf.idf* values for all terms, then lines 6,7 and 8 can be replaced with the single instruction **sort results by**  $\rho(\text{trail})$ .

Similarly, it is possible to simplify the selection algorithm such that the processing of trail expansion is conducted equally across all webcases, or selected without per-query bias. The test here is whether the time wasted due to processing on less-important webcases is saved in not needing to compute the importance metrics.

Another alternative selection policy is to pre-select the candidate starting points for each webcase and use that as the basis for determining which threads should be run on which servers. This is probably the most sensible policy to use in a local network environment.



**Algorithm 11** (**Merge**(*query*))

```
1. begin
2.   compute importance of each webcase w.r.t. query
3.   for each webcase
4.     if importance(webcase) > threshold
5.       compute  $f_1(n, \text{importance}(\text{webcase}))$  trails from webcase without reduction.
6.       foreach trail  $\in$  results
7.          $\text{weight}_{\text{trail}} = f_2(\rho(\text{trail}), \text{importance}_{\text{webcase}})$ 
8.       end foreach
9.     end if
10.  end foreach
11.  sort results by weight
12.  return results
13. end.
```

Figure 6.42: Algorithm to merge sets of trails from multiple sources.

## 6.8 Concluding Remarks and Future Work

The chapter has demonstrated the usefulness of applying the navigation engine to the discovery of Memex-like trails to Web site navigation. The development of this application represents a key contribution in helping to solve the navigation problem.

The demonstration of graphical interfaces, examples of the trail-based approach for solving the navigation problem, and presentation of theoretical techniques for allowing scalable, distributed, trail-based search on the Web have been backed up by extensive evaluations. As a result of these evaluations, the following conclusions were reached:

1. The quality of the returned trails is sufficiently high to be of genuine use to users. The findings by Mat-Hassan and Levene show that the combination of engine and interface is highly effective in increasing satisfaction and reducing the task completion time.
2. The resource discovery problem was defined in section 2.4 as the problem of finding a given document or resource answering an information need. Furthermore, the navigation problem was defined in section 2.8 as that of stopping people from getting “lost in hyperspace”. The suggestion of trails is analogous to the provision of guided tours and helps solve the navigation problem. The use of trails also provides context and thus also helps to solve the resource discovery problem by allowing users to more easily distinguish between relevant and irrelevant documents.
3. Relying on standard IR evaluation techniques is not feasible for assessing trail-based search and navigation systems. Such techniques are unlikely to be viable for assessing information-unit based systems or similar novel interfaces. Standard relevance testing, anecdotal evidence and TREC-style evaluations all miss contextual information which is shown to be useful by the nature of the authored trails.
4. Small sections of pages may be useful for answering queries. Future trail-based systems should use IR techniques capable of splitting pages into separate components and identifying anchor links of interest.
5. Manual authoring of trails is difficult, time-consuming and error-prone. Bush’s original concept of human trail-blazers should be updated in favour of automated trail-finding systems, such as that proposed herein.
6. The work on the navigation engine is far from complete. Several important additions need to be made to create a complete system:
  - (a) Better IR techniques.
  - (b) A more sophisticated system for generating short titles.
  - (c) Integration with CMSs including an API to allow partial updates and instantaneous manipulation of the graph.
  - (d) Near duplicate detection.
  - (e) Distribution, including implementation and evaluation of the partitioning and trail merging schemes described in section 6.7.

All of these represent research problems which have received considerable attention. Integrating the results of recent research into the navigation engine is a minor research issue, but a significant software engineering challenge and is beyond the scope of this thesis.

## Chapter 7

# Search and Navigation in Database Systems

A generic tool could perhaps be made to allow any database which uses a commercial DBMS to be displayed as a hypertext view.

Berners-Lee 1989

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation).

Codd 1970

## 7.1 Introduction

Codd's statement that users "must be protected from having to know how the data is organized in the machine (the internal representation)." (Codd 1970) referred to the physical structures used by experts and programmers. For many users of modern systems, being protected from the internal structures of pointers and hashes is insufficient. They also need to be spared the requirement of knowing the logical structures of a company or of its databases. For example, customers searching for information on a particular product should not be expected to know the address at which the relevant data is held, but neither should they be expected to know part numbers or table names in order to access this data, as required when using the Structured Query Language (SQL).

Relational DataBase Management Systems (RDBMSs) contain much of today's corporate data, often with dynamic content generation layers providing Web views. Such data comprises a large chunk of what is known as the "hidden" or "deep" Web. The word "hidden" means that, from a practical point of view, this data is hidden from conventional search engines. The word "deep" is intended for greater accuracy, meaning that the data can only be accessed through a specialised query interface. It is estimated that the deep Web contains 500 times more information than is visible to conventional search engines (Bergman 2000). Search engines traditionally crawl data from Web sites by downloading pages and constructing indexes which reference these pages by URL. It has been shown in the previous chapters how this technique can be extended to support trail finding across Web sites. However, this technique works well only for static pages and fixed content in the "publicly indexable Web". Crawling data through hidden Web interfaces is possible (Raghavan and Garcia-Molina 2001), but expensive in terms of resources and unreliable in that there can be no guarantees that the data is complete or representative of the database.

One way for users to access data in the deep Web is through a site-specific search engine, such as the query interface at Amazon.com. One can imagine that Amazon have a relational database storing all their catalogue information, over which the full-text query facility was developed. Research shows that users actively use such interfaces and expect major Web sites to support unstructured search facilities (Nielsen 1997). These interfaces are more natural than the SQL syntax supported directly by the database. However, the full-text search will result in a loss of expressiveness relative to the full expressive power of SQL, which is an issue that will be partially explored. One can argue that many end users do not need access to the full expressive power of SQL. Studies of keyword-based search engines on the Web have shown that users type short queries, rarely use advanced features and are typically bad at query reformulation (Silverstein, Henzinger, Marais, and Moricz 1999; Jansen, Spink, and Saracevic 1998; Spink, Bateman, and Jansen 1998; Spink, Jansen, Wolfram, and Saracevic 2002). It is likely that profiles for users of database search facilities will reveal similar behaviour.

Building on the previous work on Web-based trails, a tool called *DbSurfer* has been developed which provides an interface for extracting data from relational databases. This data is extracted in the form of an inverted index and a graph of foreign key dependencies, which can together be used to construct trails of information, solving the *join discovery problem* and allowing free text search on the contents. The free text search and database navigation facilities can be used directly, or can be used as the foundation for a customized interface. The system differs from previous efforts in the algorithms used, in the presentation mechanism

and in the use of primary-key only database queries at query-time to maintain a fast response for users. Examples are shown on data from the Digital Bibliography and Library Project (DBLP) data set.

The other major repositories of data in the hidden Web are CMSs such as Vignette or Documentum. These systems help corporations with content creation, management, publishing and presentation. Often these contain facilities for intranet document management, which will track versions, changes and linkage information. In other systems, a content management system is linked to a separate system for document control. For example, Microsoft's Content Management Server can access documents stored in SharePoint Portal Server, as shown in figure 7.1. Documents in these systems often lack hypertext links between the documents which could be used to construct meaningful trails. However, the documents are often classified using a hierarchy of categories. Some strategies have been developed for linking documents based upon the categories under which they appear, which can then be used to construct trails. A prototype system, called *Vertical Trailer* has been developed to test these principles.

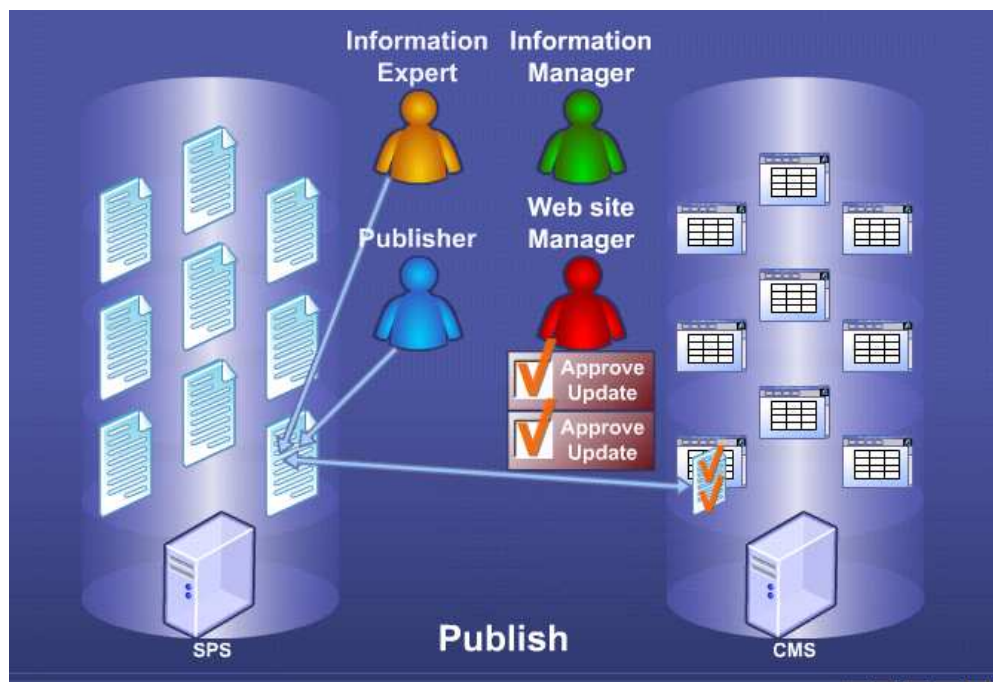


Figure 7.1: Microsoft's Content Management Server (CMS) can create pages which reference documents in a SharePoint Server (SPS), where documents are referenced by a hierarchy of categories.

The rest of this chapter is organized as follows:

**Section 7.2** describes methods of indexing the contents of relational databases to enable keyword search and trail discovery. The textual content of the database is stored in an inverted file using the architecture and data structures discussed previously.

**Section 7.3** describes related extensions to the navigation engine architecture. These exten-

sions allow DbSurfer to index database content and present data from the database in a generic web page format.

**Section 7.4** discusses the work done to incorporate XML indexing into the system. All the data extracted from relational databases is converted to XML before indexing and alternative sources of XML data can take advantage of the facilities provided.

**Section 7.5** discusses how DbSurfer complements the preceding work to provide expressive queries for solving user's information needs.

**Section 7.6** gives examples of this technique using DBLP data. This is the same data set used to test the BANKS System (Hulgeri, Bhaltoia, Nakhe, Chakrabarti, and Sudarshan 2001).

**Section 7.7** gives an overview of preliminary work into the evaluation of DbSurfer and related systems.

**Section 7.8** presents techniques for applying trail-finding techniques in data management systems where documents are classified in hierarchies, taxonomies and ontologies.

**Section 7.9** discusses alternative systems for free-text search in databases. These include BANKS, Mragyati and DbXplorer.

**Section 7.10** concludes with directions for future research.

## 7.2 Indexing Relational Databases

### 7.2.1 Translating a Relation to a Full-Text Index

A single relation (or table) is a set of rows, each of which can be addressed by some primary key. To index these rows, the data is extracted from each row in turn and a *virtual document* or Web page is constructed, which is indexed by the parser. Chapter 5 described how this parser will recognize Web content and handle document formats such as Postscript, PDF, Microsoft Office, Shockwave Flash and RPM package formats. Files of these types may be stored as Binary Large Objects (BLOBs) in a database, but may never be indexed by RDBMS structured indexing facilities. The textual content of this document is extracted and stored in an inverted file (Harman, Fox, Baeza-Yates, and Lee 1992). During the parsing stage, URLs are retrieved which reference other Web pages. These URLs may be sent to the crawler and the pages indexed in the same index. The inverted file is indexed such that the posting lists contain normalized *tf.idf* entries as described previously.

Whilst these virtual documents are transient and exist only for the time it takes for the data to be indexed, the entries in the posting lists provide references to a servlet which will reproduce a customized page for each row entry. This is achieved by extracting the data, converting it to XML using a Simple API for XML (SAX) generator and applying an eXtensible Stylesheet Language Transformations (XSLT) stylesheet to the resulting page (Harold and Means 2001). Binary data is handled with a separate servlet accessed via links from these pages. The data for these pages is always accessed via a primary key, so the page display is almost instantaneous. This is essential for providing the quick responses that users insist on (Nielsen 2000). It is a practical impossibility to guarantee response times on large databases when queries may contain full table scans and much work goes into avoiding them in traditional e-commerce systems (Gurry and Corrigan 1996).

The primary key may not be a convenient index to embed in a URL. For example, it may be a composite key with a large number of attributes or even a binary object. To cover these possibilities and make the system robust, a second identifier may be created to identify this key, giving a two step lookup process. This index is held externally to the database. Oracle databases contain a unique *rowid* for each table which can be indexed, preventing this two-stage process. Similar optimizations exist for other databases, but these have yet to be fully exploited.

### 7.2.2 Generating the Link Graph

Answers to users' queries may not be contained in a single table. Often the results are spread over several tables which must be joined together. Such queries can be answered with the help of a *link graph*. The link graph is constructed by examining the foreign key constraints of the database (either by the accessing the data dictionary table or via the Java DataBase Connectivity (JDBC) APIs) and the data entries themselves. Each (*table, attribute*) pair where there is a recognized referential constraint generates a bi-directional link. Each row entry is converted to a URL and the indexes for these URLs are added to the link graph. The set of links between Web pages and between database rows and Web pages is also added to this graph. This is equivalent to the Web graph used for site search and is stored in the same



manner. The strength of this approach is that it allows transparent access to the database in a manner which is compatible with access to any other Web page and allows for relational data to be joined with relevant Web data. This mechanism can be extended to include secure access for users on an internal network.

### 7.2.3 Computing Joins with Trails

A natural join of two tables or relations  $R$  and  $S$  is the relation containing tuples that result from concatenating every tuple of  $R$  with every tuple in  $S$  where the values for some set of common *join attributes*,  $b$  are equal in both  $R$  and  $S$ . The natural join (Codd 1970) of  $R$  with  $S$  is defined as the set:

$$R \bowtie S = \{(a, b, c) | (a, b) \in R \wedge (b, c) \in S\}$$

The *join discovery problem* is defined as being that of determining a network, tree or sequence of tuples which may be joined to best provide an answer which satisfies a users information need. In both the relational algebra and in SQL, the joins are specified directly by the user. In the case of SQL, the relations  $R$  and  $S$  are specified in the **from** clause of the **select** statement whilst the set of join attributes  $b$  is specified in the **where** clause, which also restricts the set of tuples to those with elements matching specified values. This places the burden of responsibility firmly on the user. In a keyword-based tool, the computer must be responsible for identifying these connections.

In a correctly configured RDBMS schema, the permissible joins may be inferred by the integrity constraints on the relations of the schema. It has been shown how these can be used to construct a graph of related tuples. The inverted file can be used to identify those tuples which are most likely to be relevant to the user's query, and the potential gain metric can be used to identify those nodes which are highly connected. Given this information, the *navigation engine* can be utilized to construct joins. Each node on a trail found by the Best Trail algorithm directly specifies a primary key value to identify a tuple in  $R$  or  $S$ , and a link between two nodes implies a natural join exists between the single item relations containing these tuples. The concept of trails is well established in the hypertext community, but DbSurfer represents the first system to allow their construction across tables in relational databases.

### 7.3 Extending the Navigation System

Conventional Web search engines usually use an architecture pattern comprising three components – a robot or crawler, an indexer and a query engine (Pinkerton 1994; Brin and Page 1998; Risvik and Michelsen 2002). It has been shown how the design can be extended with the trail finding system. In addition, the crawler can be augmented with the database indexer described above. A key difference between the DbSurfer and a conventional search engine is that a search engine traditionally returns links to pages which are logically and physically separated from the pages of the servers performing the query operations, whereas the links returned by the DbSurfer refer mostly to the row display servlet described. Figure 7.2 shows the detailed architecture. The data from the database is retrieved by the DbReader when the index is built and by the display servlet when examining the constructed trails. The database indexer (or reader) works by connecting to the database, selecting all the accessible tables and views available, and asking the administrator which of these should be indexed. The program will then extract the referential constraints for all of the selected tables and build a lookup table. This is kept separate from the main index and used by both the indexer and the display servlet.

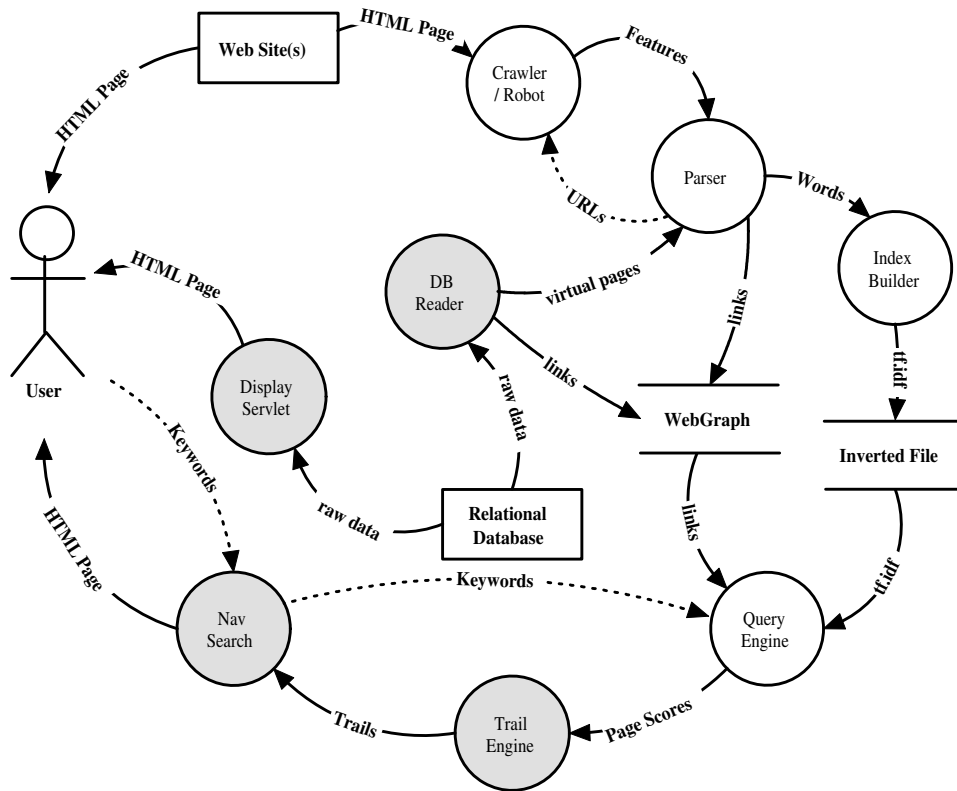


Figure 7.2: Architecture of DbSurfer. Closed boxes represent the external sources of data which the user is interested in; open-ended boxes represent internal data stores; unshaded circles represent processes typically associated with search engines; shaded circles represent processes unique to DbSurfer; solid arrows represent data flow and dotted arrows represent flows of important information (URLs and Queries). Simple keyed “get” instructions (for example in HTTP requests) are omitted for clarity.

## 7.4 Semi-Structured Data and XML

A relational database can be viewed as a special case of a more general model of semistructured data and XML (Abiteboul, Buneman, and Suciu 2000). Hence it might not be surprising that XML data can be handled using DbSurfer. Indeed, that is all that DbSurfer does! The virtual documents alluded to in section 7.2 are XML representations of relational tuples. Figure 7.3 shows an example of this from a row in the DBLP schema discussed in section 7.6. The superfluous `row` element has been added for compatibility with the emerging SQL/XML standard (Eisenberg and Melton 2002). The proposed standard includes generation of an XML Schema which is neither constructed nor required at present.

1. `<PUBLICATION>`
2.   `<row>`
3.     `<JOURNAL> Advances in Computers </JOURNAL>`
4.     `<KEY> journals/ac/Dam66 </KEY>`
5.     `<PAGES> 239-290 </PAGES>`
6.     `<TITLE> Computer Driven Displays and Their Use in Man/Machine Interaction. </TITLE>`
7.     `<TYPE> article </TYPE>`
8.     `<URL> http://dblp.uni-trier.de/db/journals/ac/ac7.html#Dam66 </URL>`
9.     `<VOLUME> 7 </VOLUME>`
10.    `<YEAR> 1966 </YEAR>`
11.    `</row>`
12. `</PUBLICATION>`

Figure 7.3: Example XML entry extracted from the DBLP Schema.

Attribute names are indexed as individual keywords so that a query “Anatomy of a search engine author” should return trails from the Anatomy paper to the entries for Sergey Brin and Larry Page. In addition, all keywords are associated with the containing tags. XML documents discovered on Web sites are automatically recognized as such and can be indexed in the same way, as can XML documents stored in the database, thus increasing coverage.

## 7.5 Query Expressiveness

The navigation engine query syntax has been extended to support an attribute container operation using the “=” sign. The construct  $x = y$  means that an attribute  $y$  must be contained in an XML tag  $x$ . For example, the query “Simon” might return publications relating to Simon’s probabilistic model as well as articles by authors named Simon. The query `author=simon` would restrict the returned entries to those contained in an XML element `<author>`, which translates to those in the author table. i.e. publications written by authors named Simon. The search engine query operations such as `+`, `-` and `link:` are still supported with this extension. Thus a query “Computers -type=phdthesis -type=mastersthesis” would return books, journals and articles on Computers, but no theses. This syntax does require some knowledge of either table or attribute names, but exists as an option to allow those with such knowledge to gain greater control.

This means that trails can be provided which answer disjunctive queries (the default), with preference for results containing as many keywords as possible. The return of trails containing only specific keywords can be forced, as can the retrieval of trails whose pages exclude certain keywords. The attribute syntax can also be used to provide more complex selection. For example, the query “Computers -type=phdthesis -type=mastersthesis” would be equivalent (using the DBLP webcase) to the SQL query

```
select * from publication
  where type <> 'phdthesis'
  and type <> 'mastersthesis'
```

This does not represent a major saving. However, a researcher who is trying to find the year of publication of Brin and Page’s search engine paper (Brin and Page 1998) could find the answer with a query such as “sergey anatomy”, whereas the full SQL required would be:

```
select year from publication, writes, author
  where lower(author.name) like '%sergey%'
  and lower(publication.title) like '%anatomy%'
  and writes.publication = publication.key
  and writes.author = author.id
```

The DbSurfer expression represents a significant saving in time and complexity for the user, whilst still returning the desired result. Using Oracle’s `explain plan` function (Gurry and Corrigan 1996) to examine the actions of the Oracle database when performing this query reveals that 8 operations are required to complete this query including a full table scan. Other relational databases are likely to offer similar performance. In comparison, the DbSurfer results require no database accesses to compute the trails, and require only 3 index-only accesses to examine the relevant entries, showing that DbSurfer can provide results which provide savings in database activity as well as user input.

## 7.6 Examples

In order to highlight the differences between the varying keyword-based systems for indexing relational database content, Hulgeri's lead has been followed in indexing the content of a relational database containing DBLP data (Hulgeri, Bhaltoia, Nakhe, Chakrabarti, and Sudarshan 2001; Ley 2002). The DBLP data is downloaded as an XML file<sup>1</sup> which is then parsed to create the schema shown in figure 7.4. There are four tables in the schema. The `publication` table holds details of all the journal, article and book entries; the `author` table contains details of each individual author and the `writes` table links these together. The `citation` table links publications with those which reference them.

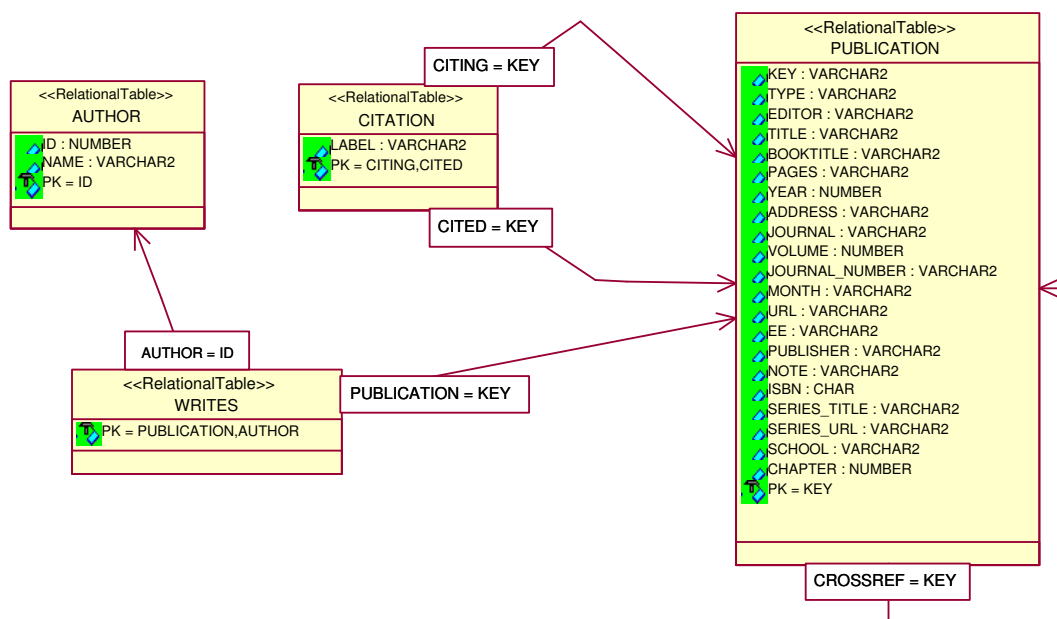


Figure 7.4: UML diagram showing the DBLP Schema. The publication table stores details of all the journal, article and book entries, indexed by the attribute `key`. The `citation` table refers to two publication entries using the foreign keys `cited` and `citing`. Finally, the `Author` table is indexed on the primary key `id`, and is linked to the `publication` table by the `writes` table, whose foreign keys are `publication`, which refers to the `key` attribute in the `publication` table and `author` which refers to the `id` field in the `author` table.

The DBLP interface is available to the public as a demonstration of DbSurfer's potential. This can be reached from the homepage for Birkbeck College School of Computer Science's Web Navigation Group<sup>2</sup>. Figure 7.5 shows results for the query "sergey anatomy". The first trail shows the entries for Sergey Brin and his much-cited paper "Anatomy of a Large Scale Hypertextual Web Search Engine" (Brin and Page 1998). In this example, the remaining

<sup>1</sup> <http://dblp.uni-trier.de/xml/>

<sup>2</sup> <http://nzone.dcs.bbk.ac.uk/>

trails are single-node trails describing other authors called Sergey and other papers with anatomy in the title. Figure 7.6 shows results for the query “vannevar bush”. The first trail is a singleton node showing the author entry for Vannevar Bush. The second shows his paper “As we may think” (Bush 1945) in the context of a citation by a later work. The third trail shows two papers describing work related to Vannevar Bush and Memex, both by James M. Nyce.

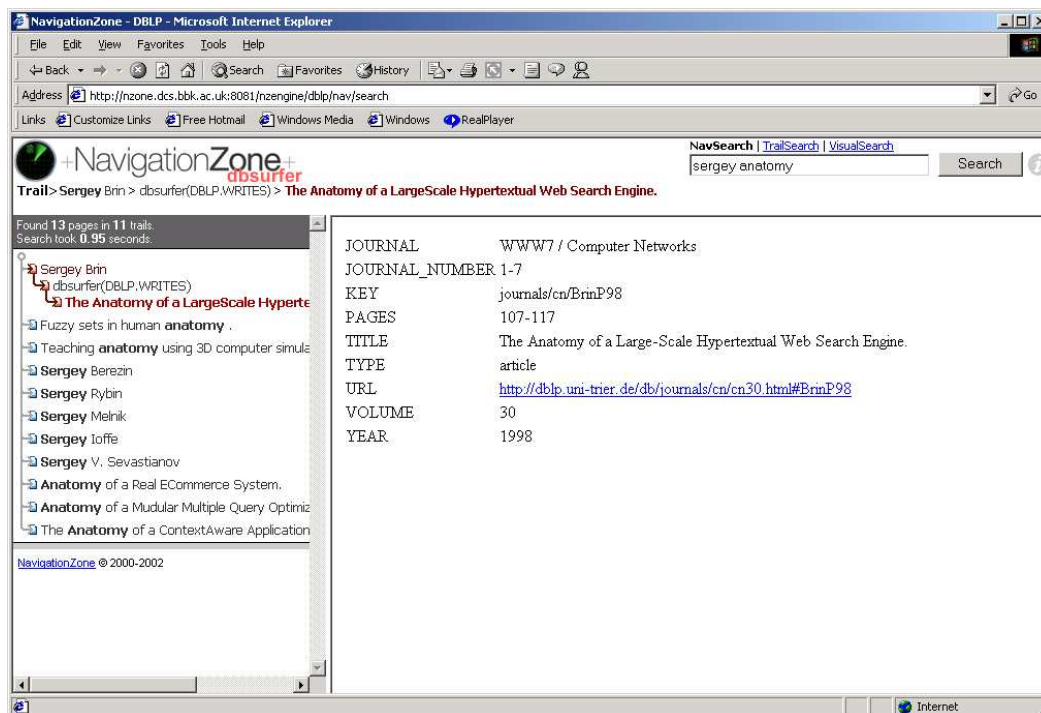


Figure 7.5: Example results using DbSurfer for the query “sergey anatomy”.

It should be noted that the DBLP already has a search system designed specifically for researchers’ needs. The DbSurfer system cannot hope to replace a custom system with its default setup. The reason for choosing the DBLP as a demonstration is to allow better testing and comparison with similar database-indexing systems. However, DbSurfer would allow the rapid deployment of a search and navigation interface in situations where no such interface exists. Also, DbSurfer can allow the development of a custom system by using XSLT stylesheets to format results. In many cases, missing features and aggregation of results can be added by constructing views at the database level.

The screenshot shows a Microsoft Internet Explorer browser window titled "NavigationZone - DBLP - Microsoft Internet Explorer". The address bar contains the URL: `http://nzone.dcs.bbk.ac.uk:8081/nzengine/dblp/nav/search`. The search bar at the top right contains the query "vannevar bush".

The search results are displayed in two columns:

- Left Column (Trail):** A hierarchical list of search results. The top result is "Vannevar Bush", which is expanded to show "Bush 1945", "As We May Think.", "dbsurfer(DBLP.WRITES)", and "Vannevar Bush". Other results include "From Memex to Hypertext: Understanding the...", "Innovation, pragmatism, and technological c...", "James M. Nyce", "Paul Kahn", "Paul Bush", "Martin Bush", "Eric Bush", "Bush Jones", "J. A. Bush", and "C. A. Bush".
- Right Column (Record):** A detailed record for the selected result "As We May Think.". The record includes the following fields:
 

JOURNAL	The Atlantic Monthly
JOURNAL_NUMBER	1
KEY	journals/theatlantic/Bush45
PAGES	101-108
TITLE	As We May Think.
TYPE	article
URL	<a href="http://dblp.uni-trier.de/db/journals/theatlantic/theatlantic176.html#Bush45">http://dblp.uni-trier.de/db/journals/theatlantic/theatlantic176.html#Bush45</a>
VOLUME	176
YEAR	1945

The footer of the page reads "NavigationZone © 2000-2002".

Figure 7.6: Example results using DbSurfer for the query “vannevar bush”.



## 7.7 Evaluation

To evaluate the relative performance of DbSurfer, two experiments were performed, on a server with 1GHz dual Pentium III processors.

In the first experiment, 20 papers were selected from those found in the DBLP corpus, with the highest ranks in the ResearchIndex (CiteSeer) “most accessed documents” list. From this list, 20 queries were constructed by taking the surname of the first author and 1, 2 or 3 significant keywords with which a user might expect to identify that paper. These queries were submitted to DbSurfer for evaluation. They were also submitted to Hulgeri’s system for Browsing AND Keyword Search in relational databases (BANKS) (Hulgeri, Bhaltoia, Nakhe, Chakrabarti, and Sudarshan 2001) and CiteSeer (Lawrence, Bollacker, and Giles 1999) for comparison. The results are shown in figure 7.7. The key result is that DbSurfer performs well (and outperforms BANKS and CiteSeer) in finding requested references. The table shows reciprocal ranks for the desired paper, in terms of the trail, page or cluster containing the relevant citation. Only the first page of results was considered in each case, but this should have minimal impact on the results. Times are shown as reported by each of the systems concerned and are not strictly comparable, but are intended to be indicative of the general level of performance. Times are missing only for those queries for which the BANKS system failed to return any results.

This result is encouraging, but may be misleading in places. The poor retrieval performance of BANKS is largely due to its poor coverage as it indexes only a subset of the DBLP data set. The poor performance of CiteSeer is due to an unnecessary dependence on boolean operations which were not tested<sup>3</sup> The 21.38 second response time for the query “nilsson routers” is due to bad configuration and behaviour of the JVM garbage collector. However, a top-and-tailed average time of 1.2 seconds is still disappointingly short of the sub-second response time expected. More worrying is that a third of queries failed to return the desired document in any of the returned trails. However, over half the desired documents were identified in the best trail for each query, suggesting that the trail-finding scheme can be highly effective.

The second experiment provided a closer analysis of the times taken to compute the results. By isolating two papers and requesting them with a decreasing number of keywords, the times taken to perform each component operation could be analysed. Computing scores for nodes takes around 50% of the total processing time, with the trail finding taking around 30%, computing the text summaries around 15%, filtering redundant information around 2%, with the remainder being taken up by system overhead, XML transformation and presentation. Increasing the number of keywords causes a limited increase in the time to compute page scores, but this impact is dwarfed by other factors. One other interesting result is that as the number of keywords increases so does the fraction of nodes in the returned trails which are distinct for the entire trailset. Only extensive user testing will confirm whether this is a positive feature.

---

<sup>3</sup> See section 6.6 for a justification of the evaluation criteria.

Query	DbSurfer		Banks		Citeseer
	1/Rank	Time	1/Rank	Time	1/Rank
crescenzi ip lookup	0.00	0.40	0.00	11.77	0.13
web database florescu	0.33	1.33	0.00		0.00
brin anatomy	1.00	0.35	0.00		0.00
digital libraries lawrence	0.00	1.74	0.00		0.00
waldvogel ip routing	1.00	0.77	0.00		0.33
rivest cryptosystems	1.00	1.22	1.00	2.93	0.25
web mining cooley	0.00	1.59	0.00		1.00
broch routing	1.00	1.43	0.00	0.93	0.06
deerwester latent semantic analysis	1.00	1.13	0.00	12.27	0.20
agrawal mining	0.33	2.20	0.00		0.00
bryant boolean function	1.00	0.70	0.00		0.00
nilsson routers	0.00	21.38	0.00	0.93	1.00
rcs tichy	1.00	0.93	0.00	1.32	1.00
traffic leland	1.00	0.99	0.00		0.00
joachims support vector	0.00	1.17	0.00	10.27	0.06
traffic paxson	1.00	0.69	0.00		0.00
time elman	0.00	1.78	0.00	1.84	0.00
workflow georgakopoulos	1.00	1.60	0.00		1.00
ferragina b-tree	1.00	1.31	0.00	13.18	0.20
fraley clusters	0.00	0.72	0.00		0.00
Average	0.58	2.17	0.05	6.16	0.26

Figure 7.7: Comparison of reciprocal rank and total time taken for 20 citation-seeking queries on DbSurfer, BANKS and CiteSeer.

## 7.8 Hierarchies, Taxonomies and Ontologies

The trails generated from the link structure of a hypertext perform well for corpuses where such structure is prevalent, but fails in situations where documents are not strongly inter-linked. The join dependencies in a relational database also provide suitable linkage information to allow the construction of trails.

Many documents exist in separate proprietary databases such as document management systems or CMSs where neither hypertext link information nor relational constraint exist. Such documents may however, be classified in a hierarchical, *ontology*, *taxonomy* or *classification tree*. Examples of well known classification trees on the Web include DMOZ and Yahoo. Examples of other taxonomies include the Dewey Decimal Classification or the classification for animal and plant life-forms. Some corporate information systems and CMSs, such as Microsoft's SharePoint (Microsoft Corporation 2001) or Verity's K2 (Raghavan 2001) provide the facility to associate documents with categories stored in a hierarchy. Recently there have been attempts to provide a standards-based XML representation for this kind of information using the Web Ontologies Language (Smith, McGuinness, Volz, and Welty 2002).

Some strategies for linking documents have been developed based upon the categories under which they appear. These links can then be used to construct trails. A prototype system, called *Vertical Trailer* has been developed to test these principles. The system is working, but is still to be subjected to experimental evaluation of its effectiveness.

Vertical Trailer takes as its input an XML file describing two basic structures. The first of these describes a taxonomy (also referred to as a classification tree or hierarchy) where categories may be associated with subcategories. For example, the category "finance" might have subcategories of "taxation", "acquisitions" and "invoices". An example taxonomy is shown in figure 7.8. The second structure is a classification map identifying those categories in the taxonomy to which documents in the corpus belong, an example of which is shown in figure 7.8.

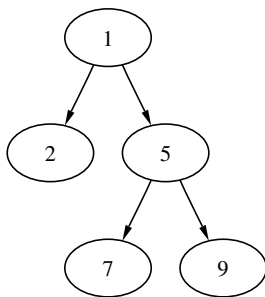


Figure 7.8: Example of a taxonomy

This input can be treated as a 4-tuple,  $(C, D, T, M)$  where  $C = \{c_1, c_2 \dots c_n\}$  is a set of categories in the classification tree.  $D = \{d_1, d_2 \dots d_n\}$  is a set of documents;  $T : C \rightarrow \wp(C)$  is a function defining the tree, taking a category and returning a list of subcategories; and  $M : D \rightarrow \wp(C)$  is a function defining the position of a document in the classification tree, taking a document and returning a set of categories. All categories and documents are associated with a unique identifier (ID), allocated sequentially as described in chapter 3.

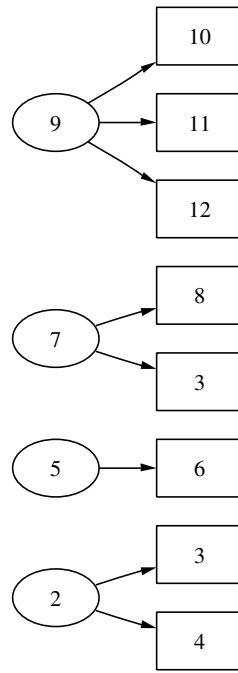


Figure 7.9: Example of documents mapped to the taxonomy shown in Figure 7.8

These are allocated such that no two documents may have the same ID, no two categories may have the same ID and no document may share an ID with any category. The numbers in the example diagrams all refer to such identifiers. This information is now used to construct a new graph over which trails can be generated.

### 7.8.1 Generating the Link Graph

Consider a set of categories arranged in a hierarchy and a set of documents classified according to this taxonomy. Each category is associated with a number of subcategories and each document may be associated with a number of categories. The predicate  $Parent : C \rightarrow \{\top, \perp\}$  denotes that  $y$  is a subcategory of  $x$  and is defined by the formula  $Parent(x, y) = y \in T(x)$ . From these definitions, a new graph can be defined on which to run the Best Trail algorithm, incorporating links for various relationships. A summary of these relationships is given in figure 7.10.

Condition	Link
$Parent(C_i, C_j)$	$C_i \rightarrow C_j$
$D_j \in C_i$	$C_i \rightarrow D_j$
$D_i \in C_k \wedge D_j \in C_k \wedge D_i \neq D_j$	$D_i \rightarrow D_j$
$D_i \in C_k \wedge Parent(C_k, C_j)$	$D_i \rightarrow C_j$
$D_i \in C_k \wedge D_j \in C_m \wedge Parent(C_k, C_m) \wedge D_i \neq D_j$	$D_i \rightarrow D_j$

Figure 7.10: Conditions under which links are added to the Vertical Trailer graph.

A link,  $C_i \rightarrow C_j$ , is created between each category and its subcategories. Links are also generated between each category,  $C_i$  and each document,  $D_j$ , assigned to it. This defines the basic tree through which trails can be constructed.

Additional links are optional, but appear to improve the quality of results. For example, a link is generated between any document,  $D_i$ , and the subcategory,  $C_j$ , of the category,  $M(D_i)$ , to which that document is assigned. This allows a single trail to flow through both the document space and the taxonomy space. Generating links between any pair of documents,  $D_i$  and  $D_j$ , which are in the same category allows trails to flow through many documents at each level of the taxonomy.

Given a graph consisting of these links, and an appropriate text index of the document's content, the Best Trail algorithm can be used to construct trails as shown previously. Brief investigations have been carried out into the effectiveness of all these strategies and whilst much work remains in testing the overall effectiveness of the system, preliminary results suggest that the trails are most useful when graphs are generated containing each of these four types.

The last link type investigated was an attempt to link every document,  $D_i$ , to each document,  $D_j$  in a subcategory of the category,  $M(D_i)$ , to which  $D_i$  was assigned. This allows the Best Trail algorithm to construct shorter trails leading only through relevant documents, but omitting the categories removes valuable contextual information and is not recommended.

### 7.8.2 Graph Construction Algorithms

The links for the first two relationships are found during the parsing of the XML file representing the input (algorithm 12). Lines 2 to 6 map to the parsing of the category structure, whilst lines 7 to 11 correspond to the parsing of the document classifications. The *writeLink* method is used to create an initial set of outlinks,  $E$ , forming a temporary graph,  $G = (N, E)$  where  $N = C \cup D$ . An example of such a graph is shown in figure 7.12.

#### Algorithm 12 (Parser( $C, D, T, M$ ))

```

1. begin
2.   foreach  $c \in C$ 
3.     foreach  $s \in T(c)$ 
4.       writeLink( $c, s$ )
5.     end foreach
6.   end foreach
7.   foreach  $d \in D$ 
8.     foreach  $c \in M(d)$ 
9.       writeLink( $c, d$ )
10.    end foreach
11.  end foreach
12. end.

```

Figure 7.11: Algorithm to create a directed graph from a document classification hierarchy.

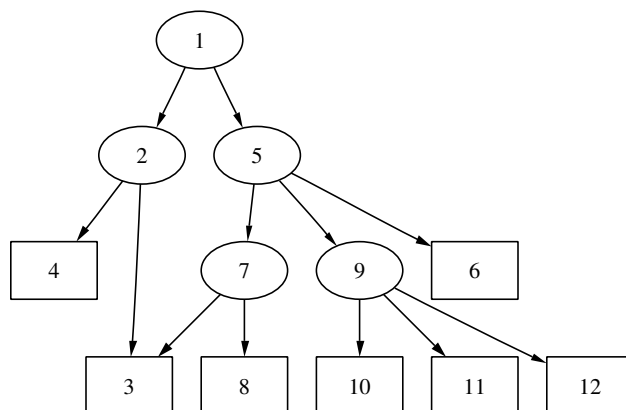


Figure 7.12: Directed Acyclic Graph (DAG) formed from the union of the taxonomy and classification map.

**Algorithm 13** (SameClass( $C, D, N, E$ ))

```

1. begin
2.   foreach  $n \in N$ 
3.     if  $n \in C$ 
4.       foreach  $\langle n, o \rangle \in E$ 
5.         if  $o \in D$ 
6.           foreach  $\langle n, p \rangle \in E$ 
7.             if  $p \in D$ 
8.               writeLink( $o, p$ )
9.             end if
10.          end foreach
11.        end if
12.      end foreach
13.    end if
14.  end foreach
15. end.
  
```

Figure 7.13: Algorithm to add links between documents with the same classification.

Algorithms 13 and 14 use this graph to compute the remaining links. The results to the questions,  $n \in C?$  and  $n \in D?$  can be solved using a lookup table constructed during the parsing stage. Hence, if an average outdegree of  $\frac{E}{N}$  is assumed, then by definition algorithm 12 takes times  $O(E)$  both on average and in the worst case, whilst algorithm 13 takes time  $O(\frac{E^2}{N})$  in the average case. This tends to the worst case of  $O(E^2)$  as the link density increases or as the outdegrees of nodes in graph become less uniform. Algorithm 13 takes time  $O(\frac{E^3}{N^2})$  which tends to a worst case of  $O(E^3)$  under similar circumstances.

**Algorithm 14 (SubClasses( $DB$ ))**

```

1. begin
2.   foreach  $n \in N$ 
3.     if  $n \in D$ 
4.       foreach  $\langle c, n \rangle \in E$ 
5.         foreach  $\langle c, s \rangle \in E$ 
6.           if  $s \in C$ 
7.             writeLink( $n, s$ )
8.           foreach  $\langle s, d \rangle \in E$ 
9.             if  $d \in D$ 
10.              writeLink( $n, d$ )
9.           end if
10.          end foreach
11.         end if
12.       end foreach
13.     end foreach
14.   end if
15. end foreach
11. end.

```

Figure 7.14: Algorithm to add parent-child links to a classification hierarchy graph.

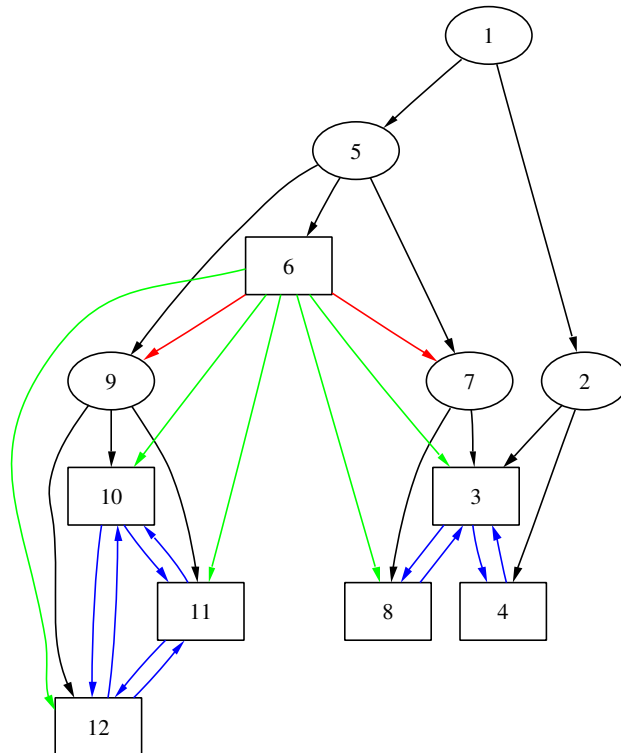


Figure 7.15: Graph used for trail discovery. Blue, red and green arrows denote links for the third, fourth and fifth conditions in figure 7.10, respectively.



## 7.9 Related Work

Recent work at Microsoft Research, at the Indian Institute of Technology, Bombay and at the University of California has resulted in several systems similar to DbSurfer in many ways. However, DbSurfer differs greatly in the design of the algorithms and in the style of the returned results. DbSurfer also offers the opportunity for intergrating both Web site and database content with a common interface and for searching both transparently.

BANKS was developed by the Indian Institute of Technology (Hulgeri, Bhaltoia, Nakhe, Chakrabarti, and Sudarshan 2001). Each result in the BANKS systems is a tree from a selected node, ordered by a relevance function which factors in node and link weights. Mragyati<sup>4</sup>, also developed at the Indian Institute of Technology, uses a similar approach in which keyword queries are converted to SQL at query time (Sarda and Jain 2001). This approach has some notable advantages. It guarantees that all data being searched on is fresh, whereas DbSurfer only ensures that the displayed data is fresh – the data in the inverted file will need to be periodically updated to ensure that it is not “stale”. The authors claim that the approach “is scalable, as it does not build an in-memory graph”. This is a legitimate criticism of DbSurfer’s approach. However, allowing almost arbitrary selection of attributes for querying and relying on the databases own indexes restricts the indexing of binary fields to those supported by the database (usually in non-standard components) and makes full-table scans probable, introducing a new problem in scalability and response time.

DBXplorer (Agrawal, Chaudhuri, and Das 2002) was developed by Microsoft Research, and like BANKS and Mragyati, it uses join trees to compute an SQL statement to access the data. The algorithm to compute these differs, as does the implementation, which was developed for Microsoft’s Internet Information Services (IIS) and SQL Server – the others were implemented in Java. DbSurfer does not require access to the database to discover the trails, only to display the data when user clicks on a link in that trail.

DISCOVER is the latest offering and shares many similarities to Mragyati, BANKS and DbXplorer, but uses a greedy algorithm to discover the *minimal joining network* (Hristidis and Papakonstantinou 2002). It also takes greater advantage of the database’s internal keyword search facilities by using Oracle’s Context cartridge for the text indexing.

Goldman et al. have also introduced a system for keyword search (Goldman, Shivajumar, Venkatasubramanian, and Garcia-Molina 1998). Their system works by finding results for queries of the form  $x$  near  $y$  (e.g. find movie near travolta cage). Two sets of entries are found – and the contents of the first set are returned based upon their proximity to members of the second set. In comparison to DbSurfer, there is no support for navigation of the database (manual or assisted) nor any display of the context of the results.

The join discovery problem is related to the problem tackled by the *universal relation (UR) model* (Ullman 1989; Levene 1992). The idea underlying the UR model is to allow querying of the database solely through its attributes without explicitly specifying the join paths. The expressive querying power of such a system is essentially that of a union of conjunctive queries (Sagiv 1983). DbSurfer takes this approach further by allowing the user to specify values (keywords) without stating their attributes and providing relevance based filtering.

---

<sup>4</sup> Sanskrit for “Search” or “Hunt”

Goldman and Widom 2000 outlines an approach for a related problem, of allowing structured database queries on the Web. WSQ/DSQ (pronounced “wisk-disk”) is a combination of two systems for Web-Supported and Database-Supported Queries. WSQ allows structured queries on Web data, by creating two virtual tables,  $WebPages(SearchExp, T_1, T_2 \dots T_n, URLRank, Date)$  and  $WebCount(SearchExp, T_1, T_2 \dots T_n, Count)$ , both of which can be queried alongside normal RDBMS tables. A similar approach to Goldman and Widom’s is adopted by Squeal, which provides **page**, **tag**, **att**, **link** and **parse** tables which can be queried using SQL (Spertus and Stein 2000).

## 7.10 Future Work and Concluding Remarks

Two systems have been described – DbSurfer and Vertical Trailer.

DbSurfer extracts data from relational databases in the form of an inverted index and a graph of foreign key dependencies. This data can then be used to construct trails of information, solving the *join discovery problem* and allowing free text search on the contents. These facilities can be used directly, or as the foundation for a customized interface.

Vertical Trailer takes XML files describing taxonomy and classification maps, which can be produced from systems such as CMSs. These are then used to construct a graph on which the Best Trail algorithm can be run.

The two systems have a great deal of potential for providing search and navigation facilities in various environments. However, several improvements are possible:

### 7.10.1 Queries

DbSurfer does not handle numbers very well – it does not handle range queries and works only using the text representation. This situation could be improved by following ideas presented in Agrawal and Srikant 2002. The system described recognizes numbers in both documents and queries and looks for close matches.

This strategy could be extended to other formats by mapping them to numeric values. Dates and times can be mapped to various numeric schemes, such as the number of time units elapsed since some predefined event or time. The number of milliseconds since 1st Jan 1970 00:00 GMT is a common example of such a system. Repetitive dates and intervals such as the 1980s, 40 days or Q3 require more thought. Mapping between equivalent unit types would also improve the quality of the results. For example, recognizing the equivalence of the measures 1 foot, 12 inches or 300mm. This type information could be transmitted as part of an XML Schema.

Colour equivalence and proximity could also be handled in this way allowing, for example, red, scarlet and vermilion to be recognized as being closely related. This might be accomplished using a multi-dimensional representation, such as Cyan Magenta Yellow Key (CMYK), Hue Separation Value (HSV) or Red Green Blue (RGB) (Foley, van Dam, Feiner, and Hughes 1990).

Given attribute-value pairs in the inverted file, some aggregate functions can be implemented by combining values at query time. An alternative strategy is to index views created at the database level, but this requires a good understanding of the values which are likely to be aggregated.

### 7.10.2 Presentation

Some presentation issues exist for the row display servlet. Backlink handling, for example, is an issue. When navigating the database structure it should be possible to examine those rows which reference a given attribute. This can be achieved by using a separate servlet to generate the list of rows, which might operate by submitting another query with the

command `link:currenturl`. This would return a list of rows which reference the current page's underlying row. With the appropriate query modification, this could be extended to restrict entries to the user's requirements.

Another issue is the handling of multipart keys. Each foreign key field is currently displayed as an outlook. However, this method of display does not extend to multipart or composite keys. In particular, it will not work for composite keys where one of the component attributes is a foreign key for some other table. In such a situation it is unclear where the destination of such a link should be.

### 7.10.3 Security

Security is a major issue. By constructing a single index the fine-grained access controls employed by the database are removed. Since all indexing is done through a single user account, the access rights for all DbSurfer users are equivalent to the access rights of that user. One possible way to restore some of the fine-grained security may be to allow each user to view the data only under the database username and password which they supply. Such a system might be implemented using container managed security which is part of the J2EE standard. This would require some very simple server configuration and a view on the data dictionary tables of the underlying RDBMS. However, this is not a complete solution as it would only affect the display servlet. This would need to be expanded so that rows which could not be displayed were never presented to the user. This would have a noticeable impact on performance. However, failure to do this would have two negative implications. Firstly, the system would present users with data which they could not access (this being analogous to returning 404s in a Web search engine <sup>5</sup>). Secondly, it might be possible to infer information without the rows being displayed. For example, if a company had an invoices table indexed, simply the presence of an entry (for example `payee=enron` or `reason=takeover`) might be considered damaging. Until these issues are resolved, the efficient indexing of secure data for unstructured search will be highly problematic.

### 7.10.4 Closing the Loop

DbSurfer and Vertical Trailer provide trail-based free-text search and navigation facilities in relational databases and classification based CMSs respectively. A complementary problem to the issues addressed by these systems is enhancing the database to provide these facilities internally. Approaches exist for unstructured search in many databases (Masier and Simmen 2001; Hamilton and Nayak 2001; Raghavan 2001; Koch and Loney 2002). One such system is OracleText (Koch and Loney 2002), which is supplied with Oracle9i and provides keyword indexing facilities for individual field entries in a table. Indexes are constructed using a SQL Data Definition Language (DDL) command such as :

```
create index i_emp
on emp(ename)
```

---

<sup>5</sup>HTTP return code 404 means Page not found

```
indextype is ctxsys.ctxcat
```

The data can be retrieved using an SQL command such as :

```
select * from emp  
  where contains (ename, 'fred') > 0
```

Two types of index are supported – *context* and *ctxcat*. Context indexes are suitable for large text fragments and have indexes which must be manually updated. An external index such as that provided by DbSurfer could feasibly replace the search facilities provided by context indexing whilst adding navigation and join-discovery facilities. Ctxcat indexes provide transactional synchronization of index and table data and are thus considerably more difficult to replace.

Context indexes can be replaced by defining functions in the database which make calls to the navigation engine using the Simple Object Access Protocol (SOAP). SOAP is an XML-based system for Remote Procedure Calls (RPC). These calls would be made to return trails which would then be converted into tables of objects, with an appropriate schema such as that described in Heather and Rossiter 1990. It is theoretically possible to replace ctxcat indexes using trigger mechanisms attached to the table entries, but the development of a generic approach for applying this technique has yet to be established. Figure 7.16 shows how the new system would fit into the existing framework (figure 7.2), thus “closing the loop”. Data will be able to flow from the database, through the database indexer to the navigation engine, then from the navigation engine back to the database.

### 7.10.5 Concluding Remarks

Two systems have been presented for handling data extracted from databases. DbSurfer is a powerful system for keyword search and navigation through relational databases. DbSurfer solves the join discovery algorithm by discovering Memex-like trails through the graph of foreign-to-primary key dependencies. This allows queries to be answered efficiently without relying on a translation to SQL. Vertical Trailer is a prototype system for trail discovery on data from document management systems where documents are associated with categories. Both systems find trails in non-hypertext areas where the use of such paths has previously received little attention.

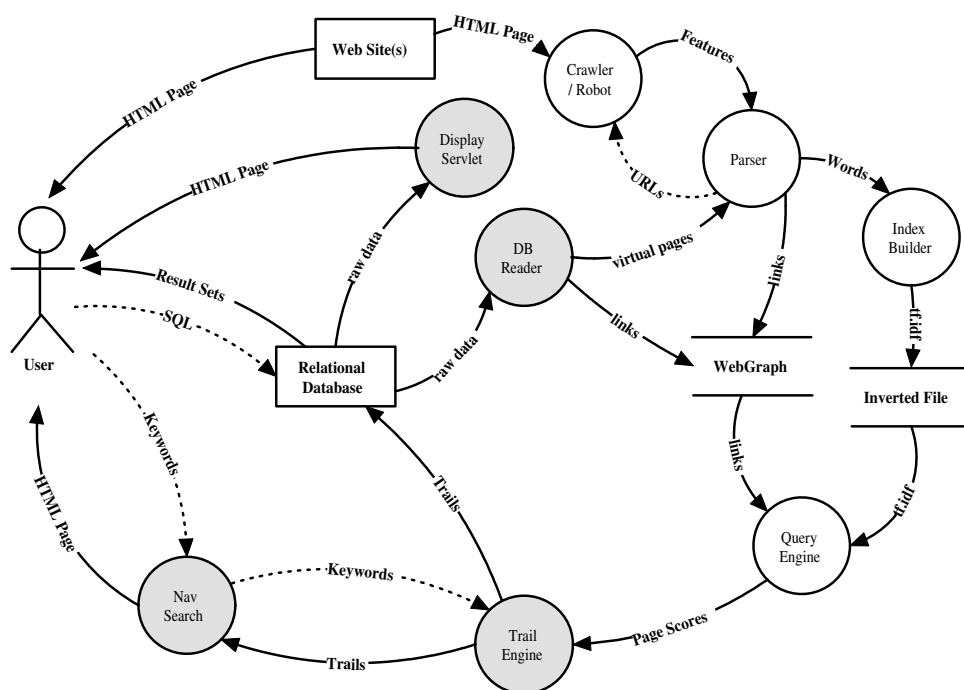


Figure 7.16: Revised architecture of DbSurfer.

## Chapter 8

# Trails and Program Comprehension

If debugging is the art of removing bugs,  
then programming must be the art of inserting them.

Unknown

An apprentice carpenter may want only a hammer and saw, but a master craftsman employs many precision tools. Computer programming likewise requires sophisticated tools to cope with the complexity of real applications, and only practice with these tools will build skill in their use.

Robert L. Kruse, Data Structures and Program Design

Don't get suckered in by the comments – they can be terribly misleading. Debug only code.

Dave Storer

## 8.1 Introduction

The previous chapters have discussed the navigation problem in hypertext and Web sites. A similar problem exists in automatically-generated corpuses such as program documentation. Program documentation, like other online support systems, provides the opportunity for enhanced productivity. However, users often take longer finding the required information in such systems than they would in conventional paper-based documentation (Tomasi and Mehlenbacher 1999).

Javadoc (Friendly 1995) is one such system, designed to document the APIs of programs and classes written in the Java programming language. This chapter describes AutoDoc – an indexing tool for Javadocs which provides the same functionality as provided for Web sites.

If Javadoc-style program documentation, which is derived from source code, can be indexed, it seems logical that the source code itself can be indexed. Source code has a long history of hypertext representation, having being shown in Engelbart’s 1968 demo. Using Nelson’s definition, it could even be considered a hypertext! It is certainly “interconnected in such a complex way that it could not conveniently be presented or represented on paper” (Nelson 1965). This chapter investigates these connections, and presents AutoCode – a companion tool for AutoDoc, which indexes Java source code and presents trails between the classes.

The rest of this chapter is organized as follows:

**Section 8.2** gives important background information – describing various concepts associated with Object Oriented Programming (OOP) and Java, including the various types of coupling which connect the classes, and the Javadoc program used to build the on-line documentation.

**Section 8.3** presents AutoDoc, with examples on the JDK 1.4 Javadocs.

**Section 8.4** describes AutoCode – including the principles and architecture of AutoCode; examples of AutoCode on the JDK 1.4 source code and details of a user study.

**Section 8.5** describes experiments into the graph structure of code, revealing a Web-like, scale-free topology replete with power-law distributions.

**Section 8.6** summarises a study in which the the Potential Gain was used as a metric to identify *key* classes and potential candidates for refactoring. This shows the utility of the AutoCode data and of the work described in chapter 4.

**Section 8.7** concludes with ideas for future work.



## 8.2 OOP, Java and Object Coupling

### 8.2.1 Object Oriented Programming

The phrase Object Oriented Programming (OOP) describes the use of programming languages and techniques based on the concept of an “object” (Howe 1993). A program is written in an Object Oriented (OO) language by specifying “classes”. A class is an Abstract Data Type (ADT) – a type whose internal form is hidden behind a set of access functions or “methods”. Objects are instances of the class type which are created and inspected by calls to these methods. In theory, this allows the implementation of the type to be changed without requiring changes to the API, and consequent changes to external code which manipulates the objects. Well encapsulated classes allow operations only via these methods. Poorly encapsulated classes expose the internal structure, state and implementation in their APIs.

The OOP style of programming and philosophy of encapsulation can be implemented in non-OO languages. OO languages feature improved support for OO constructs, and enforce certain styles of programming considered beneficial. The history of OO languages started with SIMULA-67 around 1970. Smalltalk, another creation of Xerox PARC became highly popular before the advent of C++ and Java made OO pervasive.

### 8.2.2 Java

The name “Java” can be said to encapsulate the amalgamation of four different things – a programming language, a virtual machine (the JVM), a security model and a huge class library (Zawinski 2000). The Java language is a simple, architecture-neutral, OO language with a syntax similar to that of C++. It was developed by James Gosling of Sun Microsystems during the early 1990s. The Java compiler generates architecture-neutral object files consisting of bytecode instructions which the JVM translates into native machine code, providing a separate platform for deploying applications which can run on Unix, Microsoft Windows or Macintosh systems.

The Java class libraries provide implementations of common functions required for many programs. There are classes for I/O, sound, printing, communication with relational databases, graphics, UI design, security and interprocess communication. The Java class library which ships with JDK 1.4 represents some 1.4 million lines of code spread over 6 000 classes.

Further information on Java can be found in the `comp.lang.java` newsgroup, on the `java.sun.com` Web site or in Flanagan 1997.

### 8.2.3 Coupling

Classes and objects in OO systems do not work in isolation. The classes are “coupled” to each other by various dependencies. The term “coupling” also represents the degree to which these components depend on one another. Loose coupling is desirable for code maintenance and comprehension but tight coupling may be necessary for performance reasons. Coupling is increased when the data exchanged between components becomes larger or more complex. Too much coupling is indicative of a poorly thought out design and there is evidence to suggest

that it can lead to more fault-prone software (Briand, Devanbu, and Melo 1997; Briand, Daly, and Wust 1999; Harrison, Counsell, and Nithi 1998). The Java language couples classes together with the following relationships:

**Inheritance** A class is a blueprint or prototype that defines the variables and the methods common to all objects of a certain kind. Inheritance provides the ability to derive new classes from existing classes. A “derived class” or “subclass” inherits state and behavior from a “base class” or “superclass” in the form of instance variables and methods. A subclass *A* **extends** a class *B*, adding new methods and instance variables. Subclass-superclass relationships are often modelled by saying that *A is-a B*. For example a `FileInputStream` is-a `InputStream`.

Multiple inheritance is the ability of a subclass to extend and inherit state and behaviour from more than one superclass. As Java does not support multiple inheritance and provides an implicit base class for all objects, in the form of the `java.lang.Object` class, the subclass-superclass relationships form a strictly tree-shaped “class hierarchy”.

**Interface** An interface is a collection of method and constant declarations, without implementations. It acts as a contract between classes. When a class implements an interface, it promises to implement all of the methods declared in that interface. Although Java does not support multiple inheritance of classes, it does support multiple interfaces, encouraging delegation. A class *A* **implements** an interface *B*.

**Parameter** In Java, all objects are manipulated by references. A reference to an object of class type *A* may be passed to a method of an object of class type *B*. This creates a dependency between the classes *B* and *A*.

**Return Type** Similarly, a method of an object of class type *B* may return a reference to an object of class type *A*. This object may have been created by the method, or may have existed previously. This behaviour also creates a dependency between *B* and *A*.

**Aggregation** References to objects may also be stored in an object’s data fields. A coupling between classes of types *A* and *B* may be created when the definition of class *A* states that a reference to a class of type *B* is stored. This allows classes to be aggregated together to build up more powerful classes and is often categorized by the expression that *A has-a B*.

#### 8.2.4 Refactoring

The term *refactoring* refers to a technique for improving code quality by making changes to the internal structure of the software without changing its external behaviour. Refactoring can be used to improve software design by, for example, moving code between classes, extracting code into new methods or classes or altering the position of classes in an inheritance hierarchy. Refactoring therefore leads the programmer to work more deeply on understanding what the code does and is thus an aid to maintenance and reuse (Johnson and Foote 1988). The potential benefits of carrying out refactoring are reduced duplication of code, improved readability, faster development and fewer bugs.

In terms of seminal refactoring literature, the work of Opdyke and Johnson describes a number of software refactorings (Opdyke 1992; Johnson and Opdyke 1993b; Johnson and Opdyke 1993a). Fowler, Beck, Brant, Opdyke, and Roberts 1999 describes seventy-two types of refactoring and illustrates each type with examples and UML notation. Recent empirical work in the refactoring area and its automation is found in Tokuda and Batory 2001 and in Najjar, Counsell, Loizou, and Mannock 2003, the opportunities, benefits and problems of refactoring class constructors was investigated.

### 8.2.5 The Jakarta Project

The Jakarta Project creates and maintains open source solutions on the Java platform for distribution to the public. Products are developed and distributed through various subprojects, each of which has its own team of developers. Two of the most notable projects are Ant and Tomcat.

Apache Ant (Apache Software Foundation 2003; Bailliez et al. 2002) is a Java-based build tool. It behaves in a similar way to `Make` but without some of the associated problems. Make tools are inherently shell-based. They evaluate a set of dependencies, then execute shell commands. Ant is different in that, instead of using shell commands, it uses XML-based configuration files, which define various tasks to be executed. Ant is extended by writing Java classes which define these tasks. This system sacrifices expressive power for portability. The source code for Apache Ant contains 145 000 lines of code spread over 500 classes.

Tomcat is the servlet container used in the official reference implementation for Java Servlets and JavaServer Pages (JSPs). Java servlets are typically small programs that are used to dynamically generate Web content. For example, a servlet may collect data from a database, manipulate the data and format the data as a Web page. Servlets are seen as more portable than CGI scripts, easier to secure and avoid slowdown issues due to application loading. The JSP APIs allow developers to create HTML or XML pages that combine static page templates with dynamic content by embedding special JSP tags. The source code for Jakarta Tomcat contains 150 000 lines spread over 370 classes.

### 8.2.6 Java Documentation Systems

The documentation process is an important part of any software development. Extensive documentation of any program is essential if it is to be maintained and enhanced. To meet this demand, many companies and organizations are using tools capable of producing Hypertext documentation, typically in HTML format. Such documentation is typically generated from special markup in the code. One of the best known examples of such a system is the Javadoc (Sun Microsystems Inc. 2001; Friendly 1995) tool shipped with all versions of the Java Development Kit (JDK). Exploration of the Google index reveals over 800 examples of documentation sets created with Javadoc with over 200 000 HTML pages between them. Given that many such archives are firewalled or restricted from robots, the true number is likely to be much higher.

Javadoc comments are written in HTML and precede class, field, constructor and method declarations. They usually consist of a description followed by block tags such as `@param`,

`@return`, and `@see` (which document a method's parameters return-type and related items, respectively).

Javadoc supports extensions in the form of Doclets. These can alter the style of the displayed documents, including source code annotation and syntax highlighting; write data in new formats, such as PDF,  $\text{\LaTeX}$  or XML; perform sanity and style checks or compare versions. Doclets define certain methods which enable them to run with the Javadoc program.

DocFather is a search utility developed by Siteforum Inc. It was originally created by Frank Schrufer and Dirk Schlenzig and first released in 1996. It was developed because it took too long to find the desired content by conventional browsing and no search utility was available for Javadocs. It provides search facilities for Java documentation and highlighting of keywords in the returned pages, but provides no summaries for search results, no contextual information in the display of results and no guide to the structure of the code of related classes.

Doxygen is a documentation system for C++, C, Java and Interface Definition Language (IDL). It also generates on-line documentation from documented source files and features support for generating output in HTML, Rich Text Format (RTF), PostScript, PDF and Unix man pages.

Further information on documentation systems and related techniques for program comprehension and source code analysis can be found in the proceedings of conferences such as the Source Code Analysis and Manipulation workshop (SCAM), the International Conference on Software Maintenance (ICSM), the International Workshop on Program Comprehension (IWPC) and the Working Conference on Reverse Engineering (WCRE).

### 8.3 AutoDoc

Tools such as DocFather help in finding information but fail to provide continuous support or context. Presentation of the entire documentation structure in a graph or a hyperbolic tree presents problems of information overload or graph confusing.

Logistical problems of regular indexing and providing total coverage are exaggerated with the generated content of Javadocs as the entire corpus is likely to change with any one update, rendering incremental crawling techniques useless. Fortunately the small size of the corpuses means they lend themselves well to complete reindexing, although duplicate detection schemes can reduce the overhead.

Program documentation corpora are often highly interlinked. Links are created in Javadocs due to package structure, class inheritance and type references. This is of great benefit if the information can be harvested correctly, as a link exists in most cases where the relationship between the classes suggests one should. Hence, a greater number of potential trails also exist. Filtering this increased density of link information provides the challenge.

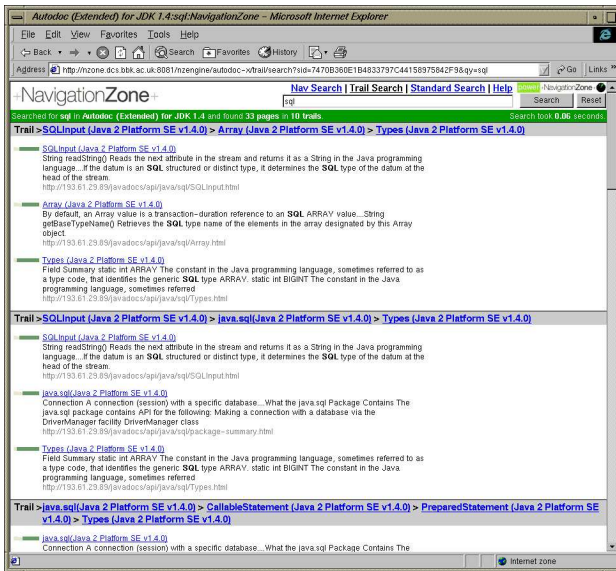
This is achieved in the AutoDoc tool – a search and navigation tool created by combining the Best Trail algorithm with a set of heuristics customized for on-line documentation. Rules are set which force zero-relevance of known pages, alter title and heading display and eliminate starting points. The improvements and heuristics made to customize AutoDoc have now been incorporated into the Web site navigation tools.

Figure 8.1 shows the AutoDoc results for the query SQL on the JDK 1.4 Javadocs using each of the tree interfaces discussed in chapter 6. More examples of AutoDoc can be found at Web Navigation Group's homepage <sup>1</sup>.

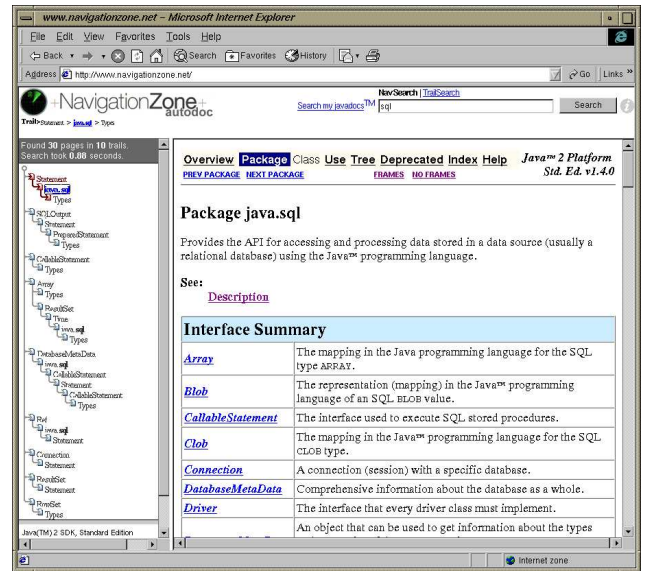
AutoDoc allows increased productivity while programming, being quicker and more flexible than using a book or manually navigating the on-line documentation. It is thus ideal for experienced users who want answers quickly, as well as providing a smooth introduction to the language for novice programmers. It also helps to expand developers' knowledge by providing contextual information such as relevant classes in the hierarchy, implemented interfaces and external specifications, giving pedagogic value as a teaching tool.

---

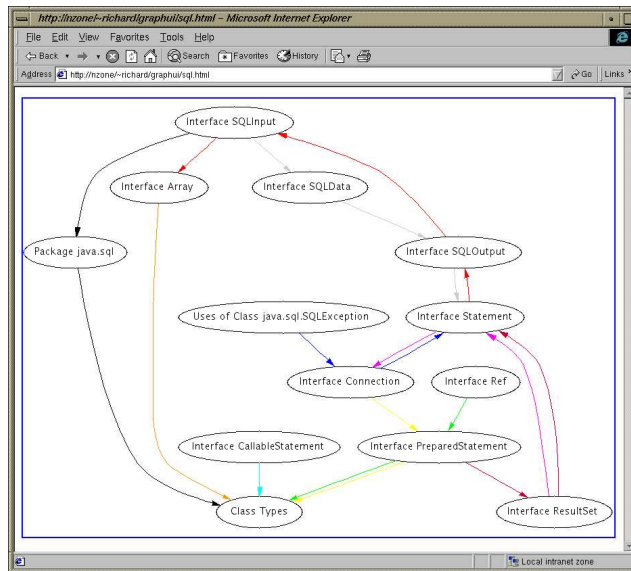
<sup>1</sup><http://nzone.dcs.bbk.ac.uk/>



(a) TrailSearch



(b) NavSearch



(c) VisualSearch

Figure 8.1: Three user interfaces for AutoDoc, showing results for the query “sql”. The results show the JDBC classes. By examining the links between pages on the trails, the user can gain a better understanding of the connections between the classes.

## 8.4 AutoCode

AutoCode is a more advanced tool for visualizing and searching within Java source code. It works using a version of the Best Trail Algorithm which supports multiple graphs.

### 8.4.1 Architecture

AutoCode indexes the Java code using a doclet containing the complete post-process architecture described in chapter 5. This communicates with the external robot and parsers via CORBA. AutoCode also provides new post-process classes to interact with the doclet APIs and to implement multiple graph support. Figure 8.2 shows how this integrates with the existing trail engine components (figure 5.1) and how the elements with which the user interacts remain largely unchanged.

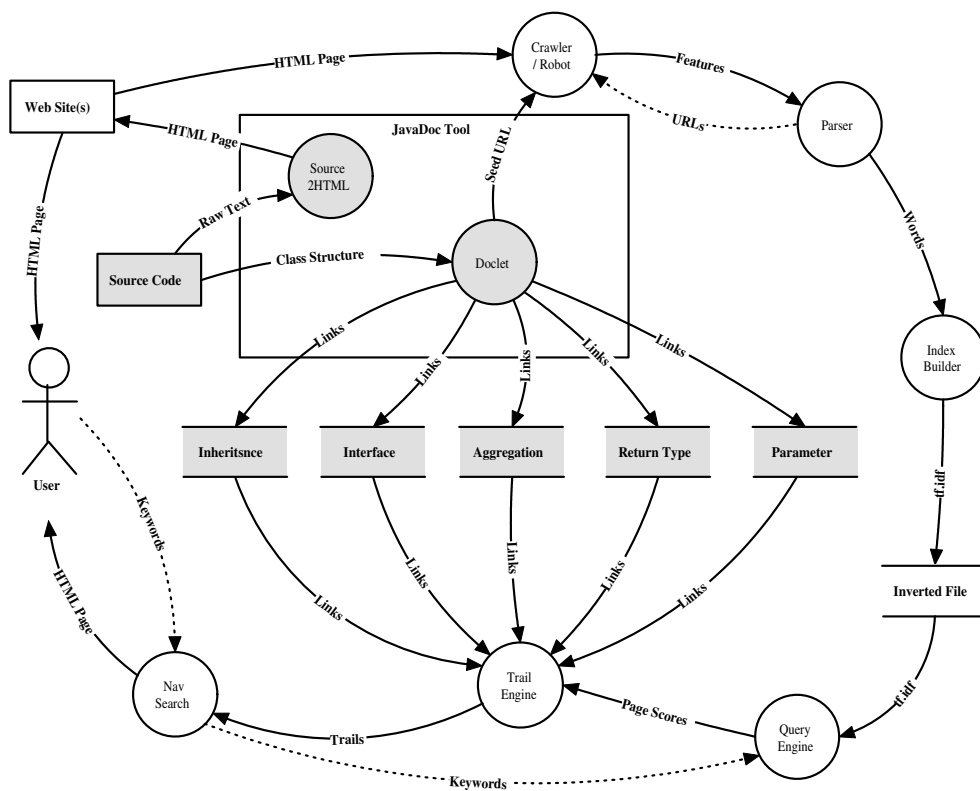


Figure 8.2: Architecture of AutoCode. Closed boxes represent external data sources, open boxes represent internal data stores, shaded circles represent processes specific to AutoCode and unshaded circles represent common processes. Solid arrows represent data flow and dotted arrows represent flows of important information (URLs and Queries). Simple keyed “get” instructions (for example in HTTP requests) are omitted for clarity.

AutoCode creates five graphs. One graph is created for each of the coupling types described in section 8.2 – Inheritance, Interface, Aggregation, Parameter and Return Type. An illustration of how these graphs can be derived from source code can be seen in figure 8.3.

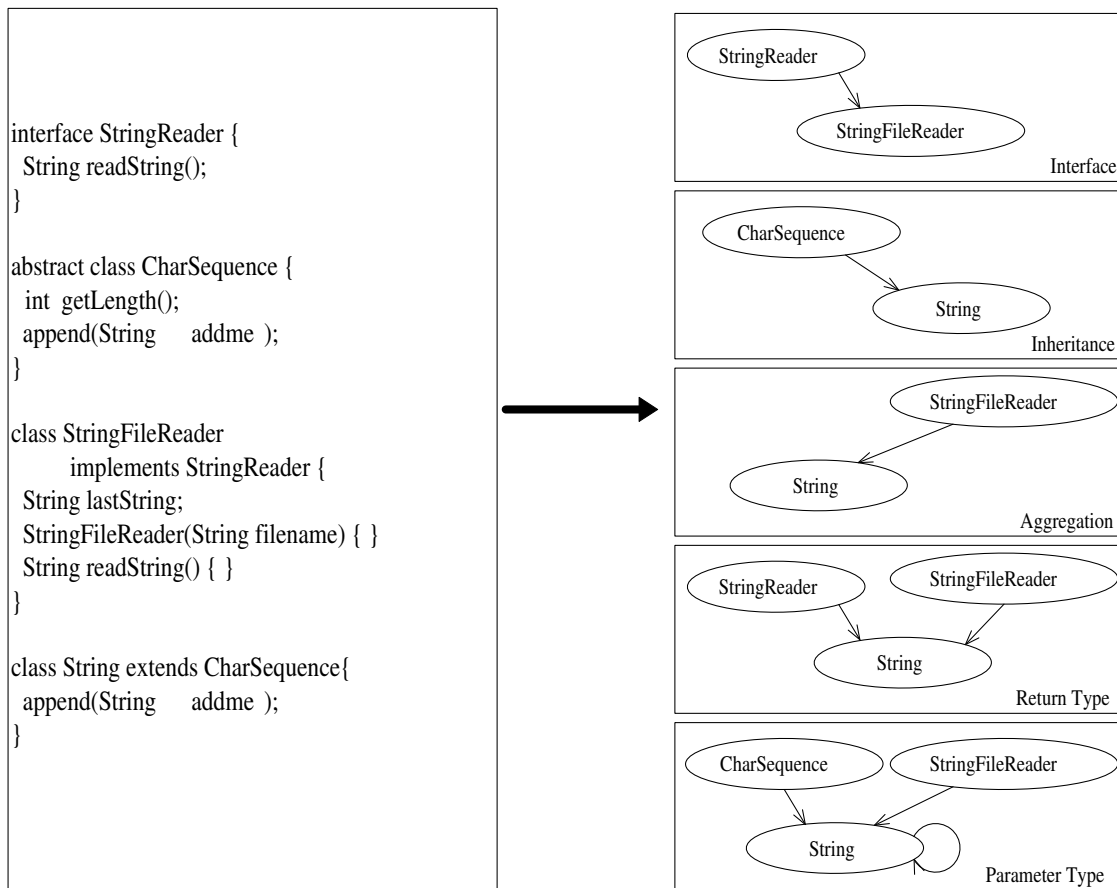


Figure 8.3: Illustration of coupling types and their graph representations.



In order to make use of these graphs, AutoCode requires a `TrailAlgorithm` implementation with multiple graph support. `MultiGraphBestTrail` extends the `BestTrail` implementation (see figure 5.3). It replaces the `addStartingPoint` method to expand navigation trees from each starting point on each of the five graphs. AutoCode also subclasses `ActiveWebcase` to provide multiple graph support. A `WebcaseStub` extends `ActiveWebcase`, delegating responsibility to a second `ActiveWebcase` and overriding methods for obtaining graphs and `TrailNodes`, returning different graphs to each `Best Trail` instance.

AutoCode uses an enhanced version of `NavSearch` with support for coloured trails. The Java source code is shown in the main window with the trails highlighted using the colour scheme shown in figure 8.1.

Colour	Link Type
Green	Parameter
Cyan	Return-type
Gold	Interface
Purple	Aggregation
Orange	Inheritance

Table 8.1: Colour scheme for trail highlighting.

### 8.4.2 Source Code Display

Trail-finding on the graphs of coupling relationships provides automated navigation and shows the context of classes. However, this is considerably less effective if the display of the source code is poor, and if there is no support for manual navigation.

Many tools exist for converting Java source code to HTML. The `SourceToHTMLConverter` classes which come with Javadoc provide neither syntax highlighting nor linking. Steven R. Brandt's `Java2HTML`<sup>2</sup> `VasJava2HTML`<sup>3</sup> and `JMarkup`<sup>4</sup> all provide syntax highlighting but don't provide links between classes. The Gnu is Not UNIX (GNU) project's `Source-highlight` works with multiple languages (C++, Prolog, Perl, etc.) but suffers the same flaw.

`XRef-Java2HTML` does provide links but to the Javadoc documentation, not to the source files. It also provides no public API. Jason Shattu's `Java2HTML`<sup>5</sup> does provide a public API and makes links to both Javadocs and between classes in source code. It also provides effective syntax highlighting. This application was chosen and used to provide the main page display for AutoCode. Calls to the API are made as part of the doctet process.

### 8.4.3 Examples

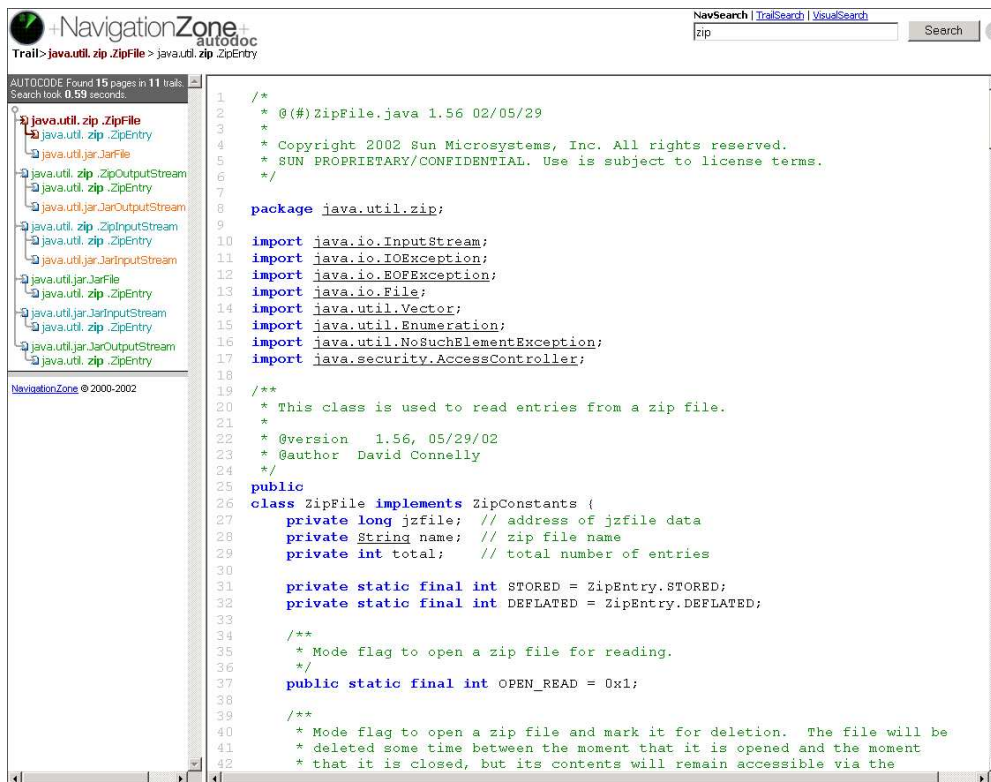
Figure 8.4 shows the results for the query “zip” on the autocode index of the JDK 1.4 source code. Figure 8.5 shows the trails more clearly. It can be easily seen that one or more methods

<sup>2</sup> <http://www.javaregex.com/cgi-bin/pat/java2html.asp>

<sup>3</sup> <http://www.chez.com/vasile/java2/Vas.Java2HTML.html>

<sup>4</sup> <http://szeiger.de/java/JMarkup.java.html>

<sup>5</sup> <http://java2html.com/>



```
NavigationZone+
autodoc
Trail> java.util.zip.ZipFile > java.util.zip.ZipEntry
NavSearch | TrailSearch | VisualSearch
zip Search
AUTOCODE Found 15 pages in 11 trails
Search took 0.59 seconds.
0
1 java.util.zip.ZipFile
2 java.util.zip.ZipEntry
3 java.util.jar.JarFile
4 java.util.zip.ZipOutputStream
5 java.util.zip.ZipEntry
6 java.util.jar.JarOutputStream
7 java.util.zip.ZipInputStream
8 java.util.zip.ZipEntry
9 java.util.jar.JarInputStream
10 java.util.jar.JarFile
11 java.util.zip.ZipEntry
12 java.util.jar.JarInputStream
13 java.util.jar.JarOutputStream
14 java.util.zip.ZipEntry
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
1 /*
2  * @(#)ZipFile.java 1.56 02/05/29
3  *
4  * Copyright 2002 Sun Microsystems, Inc. All rights reserved.
5  * SUN PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
6  */
7
8 package java.util.zip;
9
10 import java.io.InputStream;
11 import java.io.IOException;
12 import java.io.EOFException;
13 import java.io.File;
14 import java.util.Vector;
15 import java.util.Enumeration;
16 import java.util.NoSuchElementException;
17 import java.security.AccessController;
18
19 /**
20  * This class is used to read entries from a zip file.
21  *
22  * @version 1.56, 05/29/02
23  * @author David Connelly
24  */
25 public
26 class ZipFile implements ZipConstants {
27     private long jzfile; // address of jzfile data
28     private String name; // zip file name
29     private int total; // total number of entries
30
31     private static final int STORED = ZipEntry.STORED;
32     private static final int DEFLATED = ZipEntry.DEFLATED;
33
34     /**
35      * Mode flag to open a zip file for reading.
36      */
37     public static final int OPEN_READ = 0x1;
38
39     /**
40      * Mode flag to open a zip file and mark it for deletion. The file will be
41      * deleted some time between the moment that it is opened and the moment
42      * that it is closed, but its contents will remain accessible via the
```

Figure 8.4: Results for the query “zip” on the JDK 1.4 source code.



Figure 8.5: Trails returned for the query “zip” on the JDK 1.4 source code.

in the `ZipFile` class must return `ZipEntry` as a parameter. `ZipOutputStream`s also have methods which take `ZipEntry`s as parameters and `ZipInputStream`s have methods which return them. `JarFile` extends `ZipFile`. Jar files are Zip files Typically containing java classes and a special `manifest.mf` entry, hence it is not surprising that `JarFiles`, `JarInputStream`s and `JarOutputStream`s behave similarly to their zip counterparts.



Figure 8.6: Trails returned for the query “writer” on the JDK 1.4 source code.

Figure 8.6 shows the trails returned for the query “writer” on the JDK 1.4 source code. The first trail shows that `OutputStreamWriter` is a subclass of `Writer`. The second shows that `StreamResult` has a method which requires a `Writer` as a parameter. The third shows that there is a member variable of type `Writer` in the `SerializerToXML` class. The fourth and fifth trails show that `XmlDocument` has a method which requires a reference to a `Writer` object as a parameter and a method which can return one via the `XmlWriteContext` class.

The remaining trails are concerned with the `imageio` classes. They show that the `ImageTranscoder` interface is implemented by the abstract `ImageWriter` class, and that classes such as `ImageWriter` must provide at least one method taking an `ImageWriteParam` as a parameter. Actually, two methods require `ImageWriteParam` references – `convertImageMetadata` and `convertStreamMetadata`.

## 8.5 Power Law Distributions in Class Relationships

Power law distributions have been found in many natural and social phenomena and in the structure of the Web (Broder, Kumar, Maghoul, Raghavan, Rajagopalan, Stata, Tomkins, and Wiener 2000; Pandurangan, Raghavan, and Upfal 2002; Adamic 2002). Only recently has attention turned to the power law distributions found in program code and in particular in Java software.

A study has been performed on the AutoCode graphs – the primary motivation for which was to confirm the hypothesis that the graphs are scale-free and, in this sense, Web-like. Given the similarities in graph structure, it seems reasonable that similar techniques can be applied to both corpora. To verify the relationships, data was collected from the JDK libraries, Ant and Tomcat. Power laws were identified using the techniques described in section 2.3. The identification of twelve separate power law distributions shows that even when the network is decomposed by coupling type, the power laws still remain prevalent.

The primary motivation of this research is to show the connection between the structures found in the topology of the Web and the structures found in program code. By examining power law distributions, the study shows that the coupling networks are *scale-free* and, in this sense, Web-like.

The relationships discovered can help explain the structure of source code at a low level of abstraction. Identifying such patterns allows the consequences of developing larger and more complex software to be more easily predicted. For example, the data could be used to predict how many classes might contain greater than a hundred methods in a set of classes ten times larger than that of the JDK. Alternatively, it could be used to predict the maximum number of constructors of any class in that system. This may have implications for software maintenance and comprehensibility in terms of time spent and effort expended (Najjar, Counsell, Loizou, and Mannock 2003).

A further motivation is to enable models of code development that will allow developers to create synthetic code bases containing large numbers of computer-generated classes. For example, given an appropriate means of generating synthetic data, a developer could generate a data set of a much larger number of classes. This would enable them to test the consequences of developing a large system before development begins.

Finally, the work has implications for the graph traversal algorithms used in reachability analysis and garbage collection. Just as internet networks are robust against random removal of nodes (Albert, Jeong, and Barabási 2000), it is likely that random removal of classes will have little effect on the proportion of code which can be reached and thus executed.

### 8.5.1 Related Studies

O'Donoghue et al. have performed a run-time analysis of Java bytecode sequences obtained using a customized version of the Kaffe JVM (O'Donoghue, Leddy, Power, and Waldron 2002). Their experiments showed that the frequencies with which bigrams are interpreted by the virtual machine follows a power law.

Potanin et al. have conducted experiments using a query-based analysis tool called Fox,

which is an enhanced version of Bill Foote's Heap Analysis Tool (HAT). Their research has confirmed power laws in the indegree and outdegrees of the run-time object graphs of several programs (Potanin 2002; Potanin, Noble, Freat, and Biddle 2003).

Valverde et al. have shown that the indegree and outdegrees of nodes in a network of class diagrams also follow power laws, leading to a scale-free network topology similar to that of the Web (Valverde, Ferrer-Cancho, and Sole 2002). Since these diagrams have a one-to-one mapping with the source code structure, they imply that these laws are a feature of object-oriented program code.

## 8.5.2 Results

### Methods, Fields and Constructors

The majority of this study concerns coupling relationships between classes. However, three power laws were identified without type information. These relate to the fundamental building blocks of classes – the number of fields in each class, the number of methods in each class and the number of class constructors. Figure 8.7 shows log-log plots highlighting each of these relationships.

For the distribution of the number of methods, the exponents are 1.202, 1.013 and 0.766 for JDK, Ant and Tomcat, respectively. This implies that in the JDK there is a higher proportion of classes with very few methods when compared with the other two systems. This might imply fewer key classes in this system. For the distribution of the number of fields, the exponents are 0.912, 1.124 and 0.931 for JDK, Ant and Tomcat, respectively. The difference in the magnitude of the exponents would indicate no strong relationship between the the number of methods and the number of fields. It could be imagined that a large number of fields implies a larger number of methods to operate on those fields. The observations from this study lead to the hypothesis that it is infeasible to predict the number of methods from the number of fields and vice-versa. This hypothesis is supported by lack of correlation between the numbers of methods, fields and constructors (figure 8.8). The correlation matrix in figure 8.9 shows that no strong correlation exists between any of these measures.

For the distribution of the number of constructors, the exponents are 3.058, 3.363 and 2.949 for JDK, Ant and Tomcat, respectively. This implies that classes with a large number of constructors are rarely found in systems of this scale. For example, the JDK system contains only three classes with more than ten constructors. Previous work into refactoring of constructors found similar evidence for five medium-sized Java systems (Najjar, Counsell, Loizou, and Mannock 2003). Only one class was found to have ten constructors. This class was part of the Swing library.

### Coupling Power Laws

The frequency with which classes are used as superclasses to other classes can be calculated by examining the distribution of outlinks in the superclass-subclass graph. Figure 8.10 shows a bucketed log-log plot of the number of descendants of the classes in the JDK. The results show that the distribution follows a power law with exponent 0.906. The exponents for

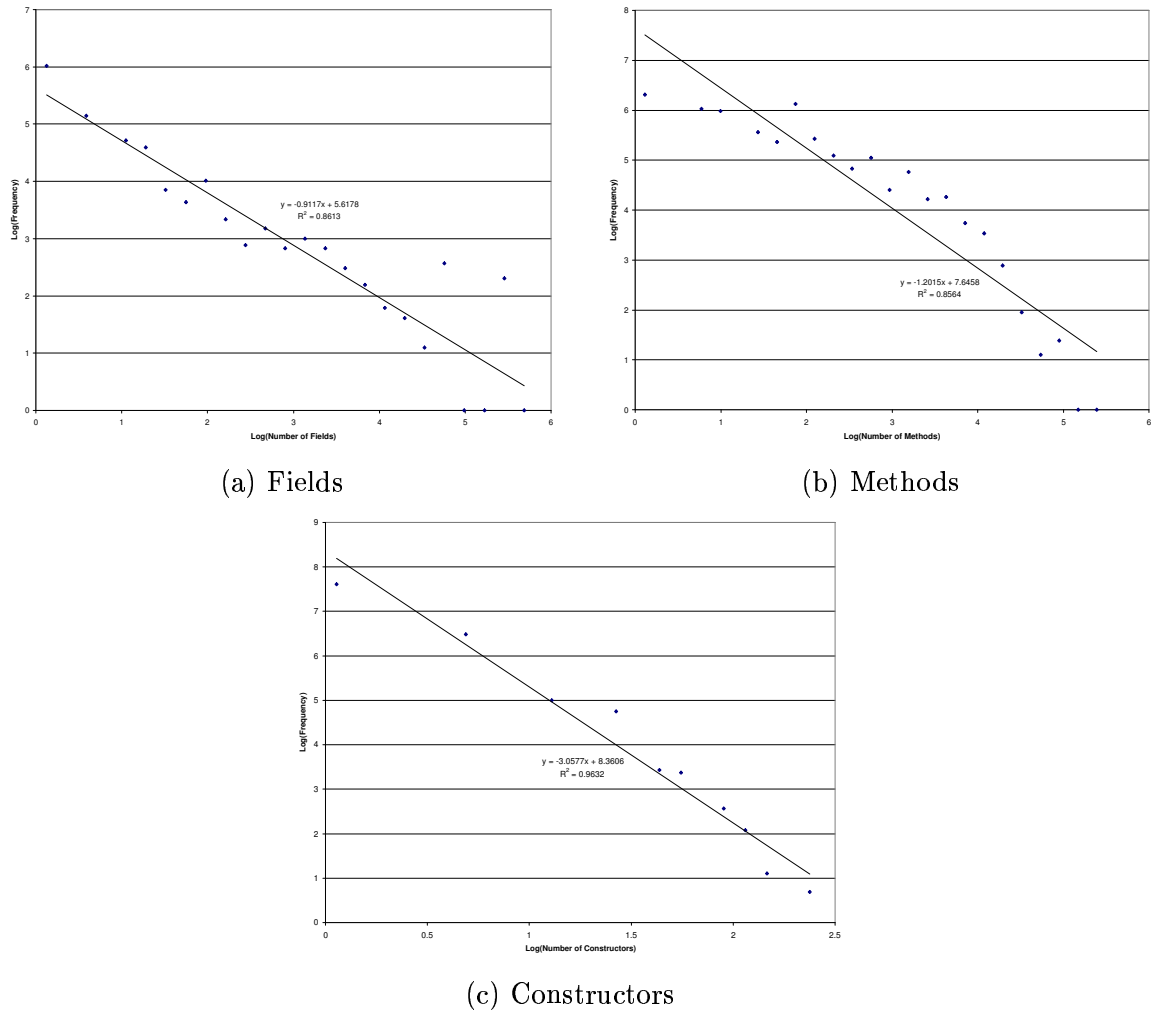
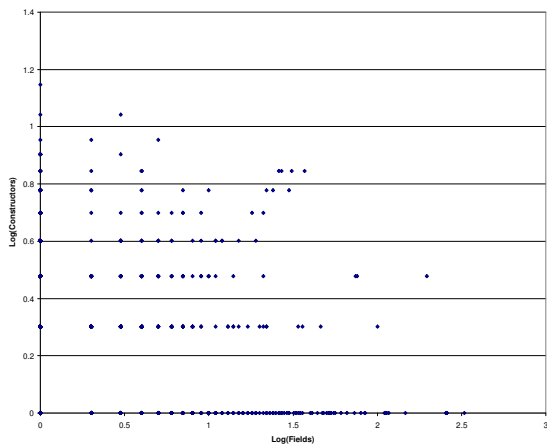
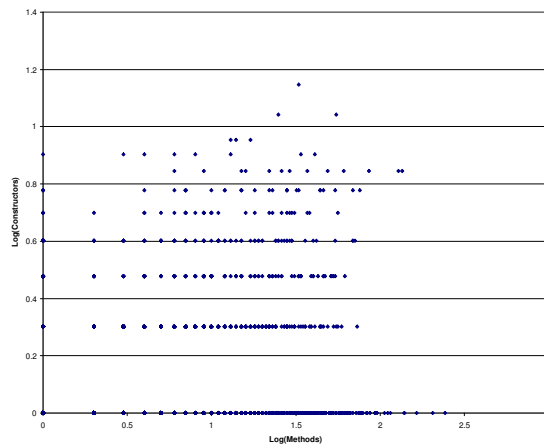


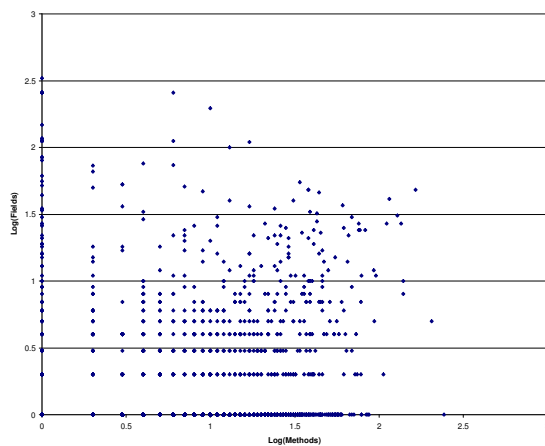
Figure 8.7: Log-log plots showing power law distributions in the number of (a) fields, (b) methods and (c) constructors of classes in the JDK class libraries.



(a) Fields vs. Constructors



(b) Methods vs. Constructors



(c) Methods vs. Fields

Figure 8.8: Log-log plots showing the relationships between (a) the number of fields and the number of constructors, (b) the number of methods and the number of constructors and (c) the number of methods and the number of fields for classes in the JDK.

	Methods	Fields	Constructors
Methods	1		
Fields	0.0506	1	
Constructors	0.157	0.010	1

Figure 8.9: Correlation matrix for class members in the JDK



Apache Ant and Jakarta Tomcat are 0.810 and 1.310, respectively. The high value for Tomcat implies that more classes in that system have relatively few descendants, whilst a small number of classes are extended by many descendants. In other words, the functionality of the system is distributed more evenly than in the other two systems. In contrast, for the Ant system, much of the functionality is contained in subclasses of key classes such as `Task` and `BaseParamFilterReader`. Hence the functionality is more concentrated in fewer classes in this system.

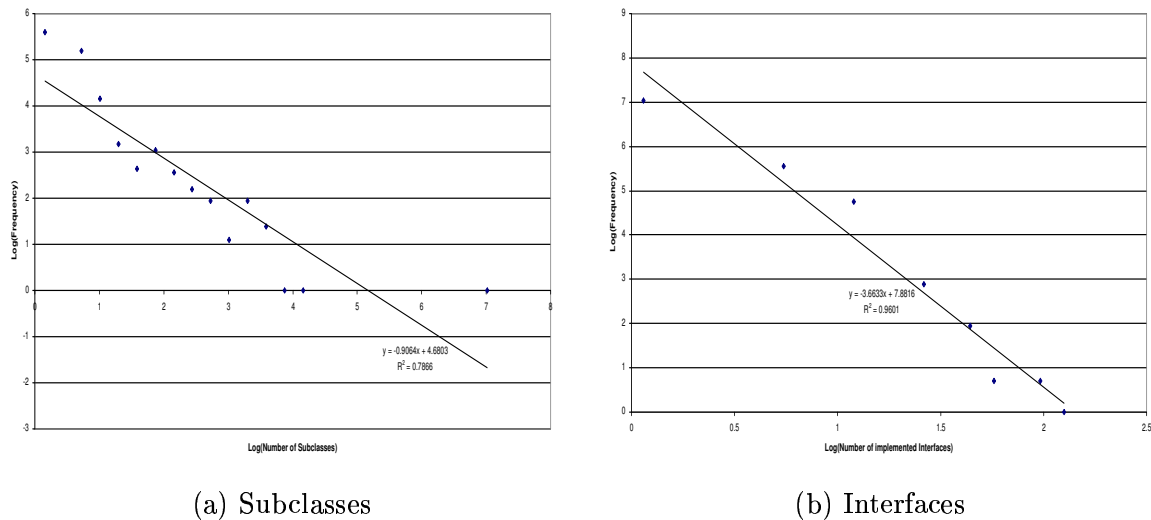


Figure 8.10: Log-Log plots showing power law distributions in (a) the number of subclasses of each class and (b) the number of interfaces implemented by classes, both based on data from the JDK class library.

The same techniques can be used to show that the distribution of the number of classes implementing an interface follows a power law – with exponents 1.130, 1.118 and 1.636 for JDK, Ant and Tomcat, respectively. This makes sense if the use of interfaces as a surrogate for multiple inheritance is considered; a similar distribution for interface implementations as for subclasses was to be expected under this assumption.

The distribution in the number of interfaces implemented by a class also follows a power law, with a much higher exponent of 3.663, as can be seen from figure 8.10. This exponent was calculated for the JDK. Insufficient data was available to calculate the exponents for the other two systems. This result can be explained by virtue of very few classes implementing a large number of interfaces. Those that do implement a large number of interfaces tend to delegate the responsibility for the methods of these interfaces to members of the same interface.

Two further power law distributions can be seen in the relationship between classes as member variables. The first is a power law distribution in the number of other classes referenced as member variables within a given class. For example, in figure 8.3, `StringFileReader` references one class, `String`, via the field `lastString`. The exponents of the distributions are 0.876, 1.267 and 1.152 for JDK, Ant and Tomcat, respectively. The low value for JDK reflects a comparatively uniform distribution of coupling via aggregation in this system. One explanation for the low JDK value may be that the roles of various packages in the system

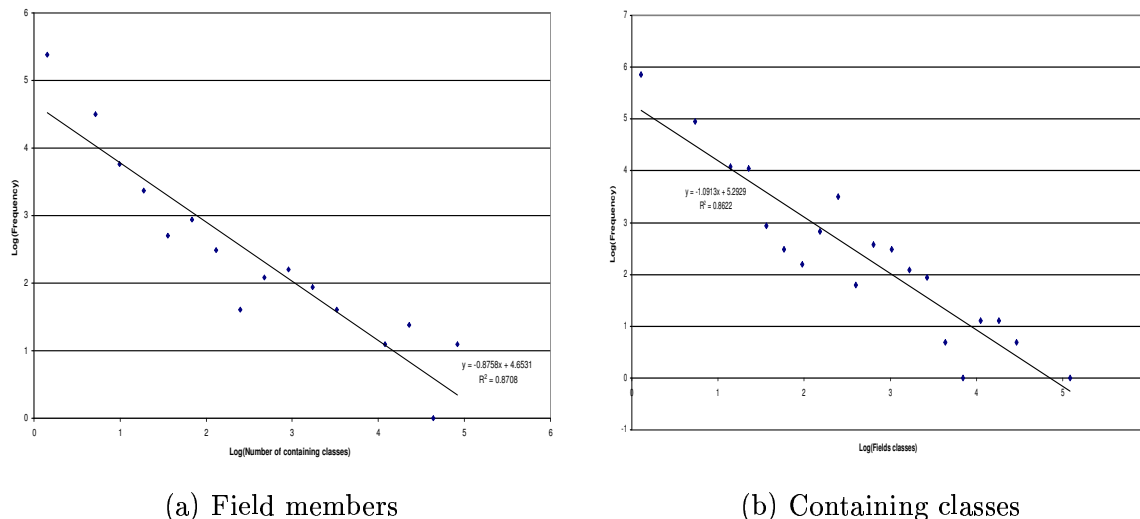


Figure 8.11: Log-Log plots showing power law distributions in (a) the number of classes referenced as field variables and (b) in the number of classes which contain references to classes as field variables.

do not overlap and hence there are multiple focal points for aggregation, as opposed to a centralized structure.

The second distribution is in the number of classes which reference a given class as a member variable. For example, in figure 8.3, `String` is referenced by one class, `StringFileReader`. The exponents of these distributions are 1.091, 1.371 and 1.934 for JDK, Ant and Tomcat, respectively. Interestingly the JDK again has the lowest exponent value supporting the previous hypothesis about multiple focal points for aggregation.

Both of these power-laws can be seen from the plots in figure 8.11. It is noticeable that the values for the first distribution are lower than the corresponding values for the second. This can be explained by the tendency in object-oriented code for many classes to be grouped together as members of another class. In contrast, it is comparatively rare for a class to be referenced as a member in many classes.

Four more class features were analyzed for power law distributions, namely the indegrees and outdegrees induced by parameter types and return types for each of the three systems. All showed scale-free topology. The Ant system has comparatively high values for all the exponents in these relationships. Inspection of the classes in this system and subsequent analysis revealed no strong correlation between usage of return types and parameters. This could be considered a surprising result, since it might be expected that parameters and return types were linked. No obvious explanation could be found for the differences in exponents between the systems.

The exponent values for all three systems can be found in figures 8.12, 8.13 and 8.14. The  $r^2$  values denote Pearson product-moment correlation. The high  $r^2$  values for JDK reflect the larger number of classes in this system. As a result, more consistency in the data may be expected. The  $r^2$  values are relatively low but still support the theory.

Relationship	Exponent	Lower 95%	Upper 95%	$r^2$
Number of Methods	1.202	0.972	1.431	0.856
Number of Fields	0.912	0.746	1.078	0.861
Number of Constructors	3.058	2.570	3.545	0.960
Subclasses	0.906	0.623	1.189	0.787
Implemented Interfaces	3.663	2.918	4.409	0.960
Interface Implementations	1.130	0.933	1.329	0.907
References to class as a member	0.876	0.682	1.069	0.871
Members of class type	1.091	0.875	1.307	0.862
References to class as a parameter	0.858	0.787	0.929	0.973
Parameter-type class references	1.183	1.050	1.316	0.948
References to class as return type	0.957	0.882	1.032	0.978
Methods returning classes	1.522	1.324	1.720	0.939

Figure 8.12: 95% confidence intervals for power law exponents in JDK.

Relationship	Exponent	Lower 95%	Upper 95%	$r^2$
Number of Methods	0.766	0.564	0.968	0.768
Number of Fields	0.931	0.702	1.160	0.834
Number of Constructors	2.949	2.394	3.503	0.990
Subclasses	1.310	0.714	1.906	0.828
Interface Implementations	1.636	0.865	2.407	0.856
References to class as a member	1.152	1.629	0.675	0.853
Members of class type	1.934	1.432	2.037	0.970
References to class as a parameter	0.711	0.375	1.046	0.595
Parameter-type class references	1.191	0.842	1.540	0.793
References to class as return type	1.043	0.666	1.420	0.751
Methods returning classes	1.362	0.883	1.840	0.801

Figure 8.13: 95% confidence intervals for power law exponents in Tomcat.

Relationship	Exponent	Lower 95%	Upper 95%	$r^2$
Number of Methods	1.013	0.799	1.228	0.854
Number of Fields	1.124	0.919	1.378	0.901
Number of Constructors	3.363	2.771	3.953	0.984
Subclasses	0.810	0.452	1.169	0.667
Interface Implementations	1.118	0.585	1.652	0.814
References to class as a member	1.267	0.410	2.124	0.881
Members of class type	1.371	0.446	2.296	0.881
References to class as a parameter	0.960	0.555	1.365	0.669
Parameter-type class references	1.480	1.110	1.850	0.864
References to class as return type	1.342	0.715	1.969	0.753
Methods returning classes	1.820	1.348	2.293	0.922

Figure 8.14: 95% confidence intervals for power law exponents in Ant.

## 8.6 Potential Gain as a Refactoring Metric

Decisions on which classes to refactor are fraught with difficulty. The problem of identifying candidate classes becomes acute when confronted with large systems comprising hundreds or thousands of classes. This section summarises work presented in Wheeldon and Counsell 2003a showing how the Potential Gain metric can be used to identify *key* classes and hence candidates for refactoring. The conjecture was made that certain classes in every OO system are of such importance (in terms of the features they possess) that they should be a priority for refactoring effort and that these could be identified by using the Potential Gain metric.

The motivation for the research stemmed from a number of sources: Firstly, developers will inevitably want to quickly identify key classes either to avoid refactoring and re-testing effort to such classes or because those classes are the ones which exert considerable influence over the system as a whole. Secondly, to investigate any differences between library-based systems and other systems in terms of coupling. Thirdly, to prove the utility of the AutoCode graph data (section 8.4) of the potential gain metric (chapter 4) in fields beyond Web metrics.

The Potential Gain is defined for classes in terms of the coupling paths between them and gives an indication of the inter-connectivity of a class with other classes. For example, a high Potential Gain value for a class at a point in the inheritance hierarchy means that the class has a relatively large number of descendants. A very low value indicates that it is a leaf node - in other words, it has no subclasses. In terms of aggregation classes either *use* other classes or are *used by* other classes (or both). Hereafter, these will be referred to as *normal* aggregation and *reverse* aggregation, respectively.

The original justification for the use of the reciprocal and geometric decay functions (section 4.2) was based on the assumption that the utility of browsing a page diminishes with the distance of the page from the starting URL. The justification for these measures in the context of coupling is based on the fact that the influence that two classes have on each other diminishes as the distance (in terms of coupling) increases. This is a features of many patterns - such as *Mediator* and *Facade* (Gamma, Helm, Johnson, and Vlissides 1995) which reduce communication and dependencies by introducing “middle-men”.

In addition to the Potential Gain, the AutoCode doclet also reported the number of *methods* in each class, the number of class *attributes*, and the *depth* or level of each class in the inheritance hierarchy (Chidamber and Kemerer 1991). Table 8.15 shows, for the JDK system, these statistics for each of the top fifteen classes when ranked in descending order by their reverse aggregation Potential Gain values. The presence of `Hashtable` and `Vector` as commonly used objects suggests that the Collections framework introduced in JDK 1.2 has not been fully adopted within the JDK. The top fifteen classes were then ranked in descending order according to their normal aggregation Potential Gain value. These fifteen classes were then compared with the reverse aggregation classes in figure 8.15. Several of the classes in figure 8.15 are highly self-referencing. Considering which classes score highly on both metrics, helps eliminate such classes. Eliminating such classes leaves `String`, `Object`, `Hashtable`, `Vector` and `Class` clearly identifiable as extensively used, key classes. Figure 8.16 shows the top fifteen classes when ranked on descending inheritance Potential Gain values. It is interesting to note that only one class from figure 8.15 appears in the top fifteen JDK classes from figure 8.16. This class was `java.lang.Object`, as might be expected; a high inheritance

Potential Gain value implies that a class has many subclasses.

Classname	Methods	Attributes	Constructors	Depth
PageAttributes.MediaType	0	223	1	2
String	58	7	12	1
Character.UnicodeBlock	1	87	1	2
HTML.Attribute	1	83	1	1
HTML.Tag	4	82	3	1
MediaSizeName	2	75	1	3
Color	29	35	7	1
Object	12	0	1	0
CSS.Attribute	3	62	1	1
AccessibleRole	0	56	1	2
Hashtable	24	14	4	2
Vector	43	4	4	3
Class	69	17	1	1
TypeCode	19	0	1	1
ObjectStreamField	12	6	4	1

Figure 8.15: The fifteen classes with the highest reverse aggregation Potential Gain values: JDK

Classname	Methods	Attributes	Constructors	Depth
Object	12	0	1	0
Throwable	17	5	4	1
Exception	0	1	4	2
Component	254	80	1	1
ComponentUI	11	0	1	1
Container	106	18	1	2
AbstractAction	12	3	3	1
AccessibleContext	24	24	1	1
JComponent	178	69	1	3
Expression	17	1	1	1
RuntimeException	0	1	4	3
Component.AccessibleAWTComponent	39	2	1	2
Buffer	21	5	1	1
EventObject	2	1	1	1
ORB	60	5	1	1

Figure 8.16: The fifteen classes with the highest inheritance Potential Gain values: JDK

Further analysis was made on the JDK, Tomcat and Ant systems, with particular attention given to “bad smells” such as *Large Class* and *Primitive Obsession* (Fowler, Beck, Brant, Opdyke, and Roberts 1999), and to *core* refactorings such as *Extract Method*, *Move Field* and *Move Method* upon which many other refactorings are based.

In addition to providing additional justification for the use of the potential gain metric and the AutoCode coupling graphs, four principal results emerged from the research. Firstly, that metrics from research domains other than software engineering can be used to aid developers and researchers in the refactoring process. Secondly, that there are substantial differences between each of the three systems investigated. Thirdly, that only the Ant system exhibited the expected properties of key classes in terms of inheritance and aggregation. Finally, the analysis supported the hypothesis that interfaces are commonly used as a surrogate for

multiple implementation inheritance.

## 8.7 Concluding Remarks and Future Work

This chapter has described the application of the trail-finding technology introduced in part II to the problem of program comprehension. Two systems have been presented for search and navigation in Javadocs and Java source code. The development of the Autocode system has shown the effectiveness of the trail-finding approach when many of the problems which are found in Web search are removed - links have types, short titles are easily generated, pages have a consistent and logical layout and are well connected. However, several issues still remain.

Documentation and source code display systems similar to those used for Java exist for other languages, with Object Oriented languages such as C++ and C# gaining particular benefit from the mapping between classes and Web pages. It is intended that both AutoDoc and AutoCode be extended to support these languages.

Interesting questions may be asked of documentation and source code corpora: Should links based on the usage of objects in code be treated in the same way as those based upon human judgements? Are these created with the same distributions of preferential linking as found in web sites? How does this affect the performance of metrics which assume that links denote quality or authority, such as Kleinberg's Hubs & Authorities (HITS) or Brin and Page's PageRank?

### 8.7.1 OO Coupling

As a first step to answering the second of these questions, the graph structure induced by the OO coupling has been studied. Twelve new power-laws have been identified. The exponents of these power laws are given for the JDK (figure 8.12), Tomcat (figure 8.13) and Ant(figure 8.14). One conclusion from this work is the belief that these regularities are common across all non-trivial object-oriented programs.

Another conclusion is that the different types of coupling examined are independent. This finding contradicts the hypothesis that high usage in one form of coupling can be used to predict high usage in another form. The implications of these findings are that the data can be used to predict the dimensions of future systems. This will allow us to estimate the complexity of developing and maintaining those systems.

It is interesting to note that the exponents for Ant and Tomcat rarely fall within the 95% confidence intervals of the JDK. It is believed that these exponents are due to deeper properties of the collections. The conclusion is that whilst there are common properties between these systems, each individual system has its own unique characteristics.

Bieman and Murdock have already shown that there is a large body of freely accessible source code available on the Web (Bieman and Murdock 2001). In terms of future work, it would be interesting to construct a Java code repository for all the available code on the Web, taking advantage of the graph structure both to show trails and generate Jar files for programmer's use. It would also be interesting to verify results of the power-law study using such a crawl. Assuming that these results hold, a number of techniques can be brought to bear to explain the phenomena.

In order to explain the power law in World Wide Web graphs, new models for its growth and evolution have emerged. The key to these models is a process known as *preferential attachment* (Albert, Barabási, and Jeong 2000) in which pages which have a high indegree are more likely to be referred to by new links. This can be explained by considering a page with higher indegree as being more popular more important and better connected. It is thus more likely to be visited by a user who may then also choose to link to that page. Research is ongoing to find methods to improve the model – for example, by combining preferential and non-preferential attachment (Levene, Fenner, Loizou, and Wheeldon 2002). Other future work will investigate the accuracy with which these models can predict the structure of program code.

Future work on refactoring will focus on two areas. Firstly, expanding the scope of what a key class is. The Potential Gain metric will be used to extract details relating to method parameters, method return types and interfaces. A second area of future research will investigate the potential for the key classes identified to be refactored. The research thus represents a first step in establishing the features of classes most eligible for refactoring.

Finally, it is hoped that the AutoCode and AutoDoc systems can be extended to allow personalized results so that programmers working on a particular field have query results tailored to their needs.



## Chapter 9

# Future Work and Concluding Remarks

The trail is the thing, not the end of the trail.  
Travel too fast and you miss all you are traveling for.

L'Amour 1984

To travel hopefully is a better thing than to arrive.

Stevenson 1894

The most important characteristic of a very complex system is the user's inability to learn its structure as a whole.

Reiser 2001

## 9.1 Summary of the Thesis

The main body of this thesis has been presented in two parts, covering two major contributions – the implementation of a navigation system for discovering Memex-like trails in directed graphs and the application of this technology to the fields of web navigation, database search and program comprehension.

An existing algorithm, the Best Trail has been refined and new algorithms have been presented for removing redundant information from trails; for computing the potential gain metric used to select starting points that allow for greater navigation opportunities; for generating web-cases – collections of files for enabling trail-based search; for generating short but meaningful extracts from web pages; for merging multiple webcases to enable scalable deployment and for generating graphs from hierarchically categorized document collections. These contributions are summarized in figure 9.1.

Section	Contribution
3.7	A new implementation of the Best Trail Algorithm, with improvements to the node selection functions.
3.6	A new algorithm for filtering redundant information from trails.
3.8	Computational complexity of the Best Trail Algorithm.
4.3	A new algorithm for computing the Potential Gain metric for determining starting points for future navigation.
4.6	Proof that the Potential Gain can aid in selecting starting points for automated navigation.
5.7	A new algorithm for computing Web page summaries.
6.3	The first application to provide completely pre-emptive computation of memex-like trails across web sites.
7.2	The first application of memex-like trails to unstructured and semi-structured search of relational data.
7.8	A new algorithm to compute graphs from hierarchically classified document collections.
8.4	The first application of memex-like trails in aiding program comprehension.
8.5	Identification of power-laws in Java source code.

Figure 9.1: Contributions of the thesis.

However, the work in this field is far from complete. The following sections will outline further opportunities for research and development to improve the trail finding mechanisms and apply them to new areas.

**Section 9.2** discusses how the system could provide personalized trails for individual users.

**Section 9.3** discusses how the system could be extended to provide trail-construction facilities as a meta-search engine interfacing with traditional search engines.

**Section 9.4** describes the software navigation problem and outlines ideas for relieving the problem using trail finding concepts.

**Section 9.5** gives suggestions for using the trail finding system to generate navigable paths through virtual environments.

**Section 9.6** describes general properties of graphs and scoring functions. Applications in which such properties are observed are likely candidates for automated trails discovery.

**Section 9.7** gives some final thoughts on the value of the thesis and the opportunities available for future research.

## 9.2 Personalization

Personalization schemes have been used extensively on the Web. They have been used to recommend goods and services on e-commerce sites, to suggest interesting stories for online newspapers and to suggest pertinent articles and discussion forums (Pretschner and Gauch 1999). Personalized search systems have been developed which filter results according to user profiles and personalized link-suggestion systems such as Personal WebWatcher have been developed to suggest the most interesting links for a user from a given page (Mladenic 1999). Just as the Webwatcher forms a trail over time, so does Personal WebWatcher. However, the problem remains that the link-at-a-time approach does not allow the user to see the context initially. Just as the *navigation engine* described in this thesis computes complete sequences in advance, so must a personalized trail engine.

Personalization systems may be developed using many techniques. Profiles may be stored as networks, keyword lists and decision trees. They may be built in advance from prior knowledge or evolve slowly over time. They may require explicit programming such as user selections or the results of a questionnaire or rely on implicit knowledge derived from bookmarks or past browsing or query history.

The *navigation engine* may incorporate personalization in several ways:

1. Redefine the node score,  $\mu(n)$ , to return the score of a node,  $n$ , with respect to a query and a user, as suggested in chapter 3.
2. Redefine the trail scoring functions,  $\rho(t)$ , to return the score of a trail as a function of the page scores and the user's profile.
3. Compute a larger number of trails and filter the resulting set based upon user profiles.

There are many challenges to be faced in the development of such a system, but the rewards will hopefully be a better search and navigation experience for all users.

### 9.3 Meta-Search

Scaling to the Web is difficult. An alternative strategy is to provide meta-search facilities across existing search engines. A search engine takes an input query and returns a set of ranked pages. In order to provide meta-search abilities, a graph must be constructed based upon the search engines output. The Best Trail algorithm can then be applied to this graph as before. Such a system could be implemented as an **ActiveWebcase** extension – taking input from many sources and building graphs for each query. Several issues immediately present themselves:

**Speed** Jakob Nielsen has stated that “Every Web usability study . . . has shown the same thing: users beg us to speed up page downloads”. In order to achieve quick response times, the meta-search system must not request multiple pages in sequence. To do so would result in the users perceived response time being greater than the sum of all operation times. Operations must be parallelizable.

**Legality** Google’s search services are made available for “personal, non-commercial use” The terms of service explicitly state that “permission to meta-search Google for a research project . . . will not be granted” (Google 2002b). Similarly, to meta-search AllTheWeb.com developers must “enter into a commercial agreement with FAST or sign up for a relevant Developers Program”. The situation has changed slightly with the introduction of Google’s SOAP API, but this remains an issue.

**Obtaining data** Conventional meta-search engines have worked by using *screen scrapers* which parse the HTML results from each engine. Writing such scrapers is very time-consuming although there has been some research in automating the process (Ashish and Knoblock 1997; Chidlovskii, Borghoff, and Chevalier 1997; Kushmerick 1997). Recently, efforts have been made to standardize the communication between such services using XML and SOAP. Google now provide such a service, which can be queried using almost any programming language on all common architectures, but with highly restricted access. In particular, there is a limit of 1000 queries a day, which provides sufficient capacity for a meta-search system to be developed, but not made public. More seriously, at the time of writing, only ten pages can be returned in the result set for a single query. This is totally inadequate for forming a graph.

**Summaries** To store all a search engine’s pages in advance would imply an ability to build a search engine equal in scale to Google. Downloading each page would cause a massive increase in bandwidth usage and user’s query response time. Therefore, any meta-search system must rely on summary data from the search engines to provide data for any re-ranking or for displaying to the user. The unsolved problem is that these summaries are not consistent (see chapter 5, section 5.7). This is a general meta-search issue, not restricted to a trail-based system.

**The Graph** A graph must be constructed on which the Best Trail algorithm can be run. However, the basic results of a search engine provide no information about the inlink or outlink structure of any of the pages. However it may be possible to construct a graph by adding soft-links based upon certain heuristics:

**DMOZ categories** Chapter 7, section 7.8 detailed how links could be constructed in such classification hierarchies. Unfortunately Google is the only major search engine to provide this information, and it does so only in limited circumstances. Restricting searches to the DMOZ pages is pointless – the current navigation engine architecture can already scale to indexing all pages referenced by DMOZ.

**Domains** Pages from the same domain could be linked together under some constraints, but this would have limited impact on results spread across many domains. What could be done is to use the first set of results to establish the important domains, then use subsequent queries for the best results on each domain. The problem with this approach is speed.

**Similarity** IR metrics could be used to detect similar pages which could be connected by soft links. Sufficient keyword information is probably not present though as the system would be forced to rely on summaries.

If these problems can be overcome, then the best trail algorithm can be used as before. To implement a meta-search system the following would then need to be accomplished:

**Parsers** must be written for extracting information from the raw data. Either HTML parsers for screen scrapers, or XML and SOAP based parsers could be used. Probably a combination of the two approaches will be required.

**ActiveWebcase** must be extended so that the data for the TrailAlgorithm comes from the parsed data.

**JSPs** must be written to glue the components together and provide a usable interface.

It would also be possible to evaluate the graph-building techniques by comparing the results of a crawled graph with those of the heuristic approach. The challenges involved in trail-based meta-search are endless.

## 9.4 The Software Navigation Problem

This work has primarily concentrated on the navigation problem as experienced in hypertext environments. A similar problem affects traditional software. Alan Cooper describes the phenomenon as “uninformed consent”. In his example, Sony Trans Com’s P@ssport, the user was “required to make a choice, the scope and consequences of which are not known” (Cooper 1999). This is a form of the Navigation Problem commonly found in all types of software.

Consider for example, a typical PC with Microsoft Windows and Office installed, with which the user wishes to write a letter. Before the user can start to accomplish his goal, he must overcome a navigation problem. The correct course of action may be to click on Start, then Programs, then Microsoft Office, then Microsoft Word. This represents four decisions which the user is expected to make, despite none of the labels offering any metadata and none of them mentioning the words “write” or “letter”. This might seem picky, but the problem gets worse when the task of changing Word’s behaviour is considered. Should the user click on “File”, “Tools”, or something else? In practice, it may be necessary to examine all menus, and try many panels and dialogue boxes before the desired option is found. For example, to change the background/foreground colours so they are easier on the eye, the user must follow the path **Tools** → **Options** → **General** → **Blue background, white text**. This problem is not specific to Microsoft software – it is a feature common to all graphical (and many non-graphical) interfaces. If the system could let the user tell the computer what he wanted to accomplish, it might be possible to suggest paths in the same manner that the site navigation engine did for hypertext.

It is possible that the next generation of command line interface will encompass keyword search features. Hans Reiser’s future vision for the Linux file-system includes a syntax for combining filename manipulation with search facilities (Reiser 2001). Reiser’s system includes two main elements. Groupings, such as  $[x\ y\ x]$ , act like search queries for isolated documents or collections of documents. Orderings, such as  $x/y/z$  can act as conventional file system paths, or can replace CORBA name paths and other hierarchical structures. For example the name  $[my\ secrets]/[love\ letter\ susan]$  would describe a document associated with the words “love”, “letter” and “susan” within a directory associated with the words “my” and “secrets”. It is possible to extend this with trail and path finding at the file system level. For example  $x/[y\ z]*$  might be a construct to find trails matching the disjunctive query of the keywords  $y$  and  $z$  from within the element identified by the name space  $x$ .

It has already been demonstrated how information from relational databases and web sources can be combined. Integrating information from the user’s operating environment and local and network filesystems seems an obvious and challenging evolution of these ideas. Figure 9.2 shows how the NavSearch interface could be used at the operating system level.

All this work is part of ongoing research into unifying the interfaces of the web, hypertext and desktop applications. There are endless possibilities for this area of research. For example, providing consistent user interfaces across different media, providing applications which communicate via abstract interface layers to provide HTML and X-Windows presentations, or incorporating Nelson’s transclusive links at the application level so that an element of one application (say Word) can be embedded into another (e.g. PhotoShop) with the same update properties as were envisaged with Xanadu.

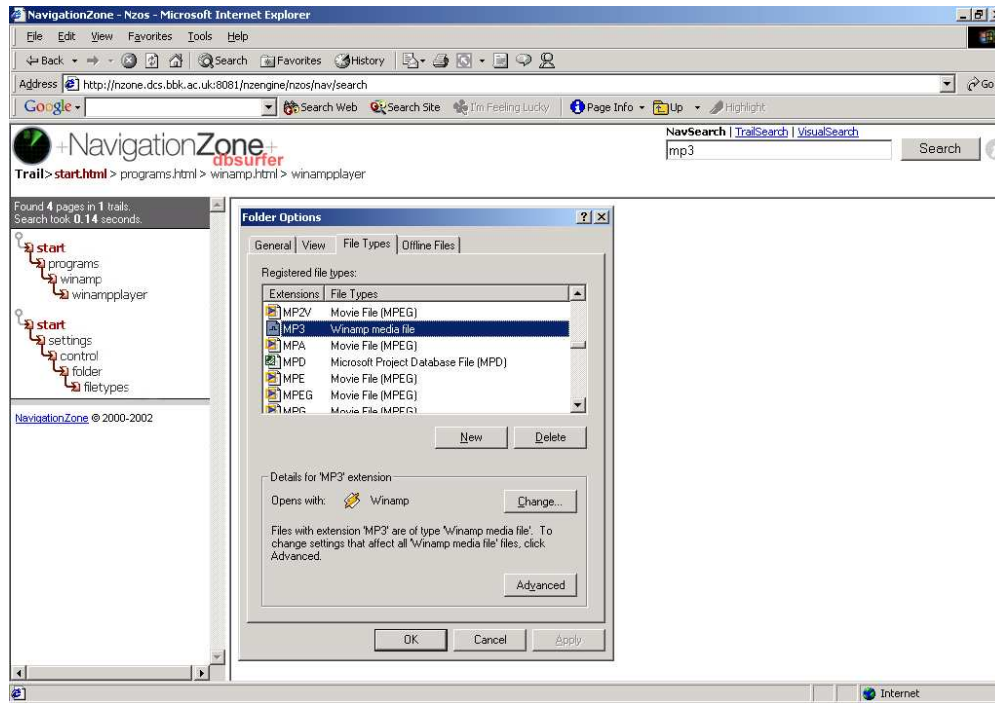


Figure 9.2: Mock-up of how the trail finding interface could be applied to the graph of potential user interactions at the operating system level.



## 9.5 Navigation in Virtual Environments

Virtual environments or 3D worlds are an area of software development with many inherent navigation issues. The problems faced are very similar to those discussed in chapter 2. Chittaro, Ranon, and Leronutti 2003 states that “Many Web3D worlds, . . . typically leave the user alone and partially or totally unassisted in navigating the environment”. They continue, stating that the usability program faced in leaving the user unassisted can range “from navigation issues (e.g. wayfinding) to difficulties in figuring out which operations can be performed on the objects in the world”.

VRML Tour Creator is a system which generates tours along a specified ordered set of points in a Virtual Reality Modeling Language (VRML) world (Chittaro, Ranon, and Leronutti 2003). The points are specified manually, as with a traditional hypertext guided tour. VRML Tour Creator then finds a path between these points, avoiding other objects in the world. A virtual tour-guide then moves along this path highlighting points of interest.

The program requires the following inputs:

1. The VRML file of the world for which the tour will be developed.
2. The Humanoid-Animation (H-Anim) model of the virtual tour-guide and its dimensions.
3. The desired maximum walking speed.
4. The ordered list of  $k$  objects/places to be presented.

The last of these could be constructed as the output of the Best Trail algorithm given a suitable network model and scoring function. For example, in the museum tour, nodes could represent rooms where links would occur whenever the rooms were adjacent. Each room would be annotated with meta-data describing the exhibits within it, the time period, origin, etc. The Best Trail Algorithm could then construct a tour through various rooms of the museum in response to the users query. As the user reached each room, a tour could be constructed describing the major exhibits in the room. The VRML Tour guide would then produce the animation required in response to this. Feedback would enable the tours to be altered in response to a better understanding of the user’s interests and needs.

## 9.6 Graph Characteristics

Several scenarios have been described in this thesis using links in Web sites, databases and source code. In each of these scenarios, one or more graphs are constructed, the vertices of which represent candidates for inclusion on trails. In the examples shown, these vertices have represented Web pages, database rows and Java classes.

Although any binary relation can be used to form a graph, not all graphs are suitable candidates for a trail finding approach. The graphs created for the suggested applications share several common properties. Some of these properties would be desirable for any trail-based system. Others are specific to the design of the navigation engine used here.

1. For each application concerned, a trail or path is a reasonable answer to an information request.
2. A scoring function is available which maps the vertices of the graph to set of real numbers. In the examples implemented, this has always been an information retrieval function, based upon Salton's normalized *tf.idf* scores.
3. The trails are scored according to their content, not their length. Future systems would also be required to work within this framework. For example, if a system was developed for computing paths on a map of London, then finding the shortest path from Leicester Square to St. Pauls would be better achieved by an algorithm such as Dijkstra's. However, if this hypothetical system was asked to suggest a tour around London that gave focus to Victorian architecture, then the Best Trail approach would be more reasonable.
4. The average indegree and outdegrees are sufficiently large enough that the trails can be meaningful and so that an exhaustive search is not a viable solution.
5. The graphs are all sparse. The Best Trail algorithm scales badly on fully-connected graphs. In the examples shown the indegrees and outdegrees are often organized according to a power law or similar distribution.
6. The graphs are neither regular nor flat. Many graph algorithms exist for special cases undirected graphs in which vertices are arranged in a plane. Most of these algorithms are not content driven, but would still provide a better starting point for investigation in these cases.
7. The graphs are often cyclic, but not always. For example, the inheritance and interface graphs used by Autocode (Section 8.4) are acyclic. If the graphs are known to be acyclic and contain no nodes with duplicate content, then the scoring functions should be altered to remove the redundancy checking. This will speed up the computation of the trails.

## 9.7 Final Remarks

The concept of trails has appeared consistently throughout this thesis. A trail-based metric (the potential gain) and a trail-finding algorithm (the Best Trail) have both been presented.

Such trails can be found everywhere. Trails can be formed by chains of logical constraints in an RDBMS, by paths of coupling relationships in computer programs, as paths through virtual and real-world environments and as sequences of moves in games and simulations. The work in this thesis has shown how common techniques can be applied to all of these areas.

In all the cases examined, the system provides significant benefits for users. Trails provide contextual cues by showing relevant pages which link to, and are linked from, other pages on the trail. This contextual information helps disambiguate meanings and gives the user a better understanding of the likely content of the trail's pages. The trails thus help the user to solve the *resource discovery problem*. This problem was defined in section 2.4 as that of finding a resource which answers a given question or which provides information about a given subject.

The contextual information also alleviates the *navigation problem*, defined in section 2.8 as the problem of preventing people from getting “lost in hyperspace”. This is achieved by showing the user where he has been and where he is relative other pages on the trails.

The use of the Best Trail algorithm is key to helping solve the navigation problem – by semi-automating the navigation process. The algorithm can follow links and separate relevant information from peripheral documents. Because the navigation process is now semi-automated, the user is saved from having to continually make navigation decisions without appropriate clues to the relative worth of links. The user has a simple option of following the path provided or using the path to make more informed navigation decisions.

The end result is an interface that (for Web sites at least) allows users to access information more quickly, more accurately and with a higher degree of confidence.

Many more improvements and applications are possible and some of these will hopefully appear in the years to come. This work marks only a starting point for many possible developments, but sadly, the end for this thesis.

I regret to announce that . . . this is the END.

I am going.

I am leaving NOW.

GOOD-BYE!

Bilbo Baggins in Lord of the Rings (Tolkien 1954)

# Appendix A

## List of Abbreviations

- AAAI American Association for Artificial Intelligence. Homepage: [www.aaai.org](http://www.aaai.org)
- ACM Association of Computing Machinery. Homepage: [www.acm.org](http://www.acm.org)
- ADT Abstract Data Type (Howe 1993).
- AMA American Medical Association.
- APCM Adaptive Probabilistic Concept Modelling (Autonomy 2003).
- API Application Programming Interface.
- ARC Augmentation Research Centre (Engelbart 1962; Engelbart and English 1968).
- ARPA Advanced Research Projects Agency. See DARPA. Homepage: [www.darpa.mil](http://www.darpa.mil)
- ASCII American Standard Code for Information Interchange. A.k.a. plain text.
- BFS Breadth First Search (Aho, Hopcroft, and Ullman 1983).
- CACM Communications of the ACM.
- CDM Comparative Development Methodologies.
- CERN Conseil Européen pour la Recherche Nucleaire. Now the European Organization for Nuclear Research. Homepage: [www.cern.ch](http://www.cern.ch)
- CMS Content Management System or Content Management Server.
- CMU Carnegie Mellon University. Homepage: [www.cmu.edu](http://www.cmu.edu)
- CMYK Cyan Magenta Yellow Key (Foley, van Dam, Feiner, and Hughes 1990).
- CNN Cable News Network. Homepage: [www.cnn.com](http://www.cnn.com)
- CORBA Common Object Request Broker Architecture. FAQ: [www.omg.org/gettingstarted/corbafaq.htm](http://www.omg.org/gettingstarted/corbafaq.htm)
- CSIRO Commonwealth Scientific and Industrial Research Organisation. Homepage: [www.csiro.au](http://www.csiro.au)
- DAG Directed Acyclic Graph (Aho, Hopcroft, and Ullman 1983).
- DARPA Defense Advanced Research Projects Agency. See ARPA.
- DBLP Digital Bibliography and Library Project. Homepage: [dblp.uni-trier.de](http://dblp.uni-trier.de)
- DBMS DataBase Management System. See RDBMS.
- DDL Data Definition Language. Part of SQL.

- DFS Depth First Search (Aho, Hopcroft, and Ullman 1983).
- DML Data Manipulation Language. Part of SQL.
- DMOZ Directory MOZilla. See ODP. Homepage: [www.dmoz.org](http://www.dmoz.org)
- DTI Department of Trade and Industry. Homepage: [www.dti.gov.uk](http://www.dti.gov.uk)
- FAQ Frequently Asked Questions.
- FCE Faculty of Continuing Education.
- FTP File Transfer Protocol.
- GA Genetic Algorithm.
- GUI Graphical User Interface.
- HCP Hamiltonian Cycle Problem. See: [www.ing.unlp.edu.ar/cetad/mos/Hamilton.html](http://www.ing.unlp.edu.ar/cetad/mos/Hamilton.html)
- HITS Hyperlink-Induced Topic Search. See Kleinberg 1998 for details.
- HPA Hypertext Probabilistic Automata (Borges 2000).
- HPG Hypertext Probabilistic Grammar (Borges 2000).
- HPP Hamiltonian Path Problem. See HCP.
- HSV Hue Separation Value (Foley, van Dam, Feiner, and Hughes 1990).
- HTML Hypertext Markup Language.
- HTTP Hypertext Transfer Protocol (Berners-Lee 1996).
- ICDT International Conference on Database Theory.
- ICSM International Conference on Software Maintenance. Homepage: [www.cs.vu.nl/icsm2003](http://www.cs.vu.nl/icsm2003)
- IPJS Information Processing Society of Japan. Homepage: [www.ipsj.or.jp](http://www.ipsj.or.jp)
- IWPC International Workshop on Program Comprehension. Homepage: [www.iwpc2003.uvic.ca](http://www.iwpc2003.uvic.ca)
- IDF Inverse Document Frequency (Baeza-Yates and Ribeiro-Neto 1999).
- IDL Interface Definition Language. Tutorial: [java.sun.com/docs/books/tutorial/idl](http://java.sun.com/docs/books/tutorial/idl)
- IE Internet Explorer. Homepage: [www.microsoft.com/windows/ie](http://www.microsoft.com/windows/ie)
- IIS Internet Information Services. Homepage: [www.microsoft.com/iis](http://www.microsoft.com/iis)
- IMF International Monetary Fund. Homepage: [www.imf.org](http://www.imf.org)
- IT Information Technology.
- JDK Java Development Kit. Homepage: [java.sun.com](http://java.sun.com)
- JPEG Joint Photographic Experts Group. Usually refers to the image format developed by the group rather than the group itself (Foley, van Dam, Feiner, and Hughes 1990).
- JSP Java Server Pages. Homepage: [java.sun.com/products/jsp](http://java.sun.com/products/jsp)
- JVM Java Virtual Machine.
- KMS Knowledge Management System.
- LNCS Lecture Notes in Computer Science. Homepage: [www.springer.de/comp/lncs](http://www.springer.de/comp/lncs)
- MBA Masters in Business Administration.
- MIME Multipurpose Internet Mail Extensions (Freed and Borenstein 1996).

- NLP Natural Language Processing.
- NLS oN-Line System (Engelbart 1962; Engelbart and English 1968).
- NIST National Institute of Standards and Technology. Homepage: [www.nist.gov](http://www.nist.gov)
- ODP Open Directory Project. See DMOZ. Homepage: [www.dmoz.org](http://www.dmoz.org)
- OMG Object Modelling Group. Homepage: [www.omg.org](http://www.omg.org)
- OOP Object Oriented Programming.
- P2P Peer to Peer.
- PARC Palo Alto Research Center. Owned by Xerox and home to the WIMP user interface standard and Ethernet networking. Homepage: [www.parc.xerox.com](http://www.parc.xerox.com)
- PDF Portable Document Format. Homepage: [www.adobe.com/products/acrobat/](http://www.adobe.com/products/acrobat/)
- PPR Probabilistic Phrase Reranking (Radev, Fan, Qi, Wu, and Grewal 2002).
- RDBMS Relational DataBase Management System.
- REP Robots Exclusion Policy. Homepage: [www.robotstxt.org/wc/robots.html](http://www.robotstxt.org/wc/robots.html)
- RFC Request for Comments. Homepage: [www.ietf.org/rfc.html](http://www.ietf.org/rfc.html)
- RGB Red Green Blue (Foley, van Dam, Feiner, and Hughes 1990).
- RPC Remote Procedure Calls.
- RPM Redhat Package Manager. Homepage: [www.rpm.org](http://www.rpm.org)
- RTF Rich Text Format.
- SALSA Stochastic Approach for Link-Structure Analysis (Lempel and Moran 2000).
- SAX Simple API for XML (Harold and Means 2001; Brownell 2002).
- SBA Small Business Administration. Homepage: [www.sba.gov](http://www.sba.gov)
- SCAM Source Code Analysis and Manipulation (workshop). Homepage: [www.brunel.ac.uk/csst-mmh2/scam2003](http://www.brunel.ac.uk/csst-mmh2/scam2003)
- SCC Strongly Connected Component. A set of the nodes in a graph where any node is reachable from any other (Aho, Hopcroft, and Ullman 1983).
- SCSIS School of Computer Science and Information Systems (at Birkbeck). Homepage: [www.dcs.bbk.ac.uk](http://www.dcs.bbk.ac.uk)
- SGML Standard Generalized Markup Language. Overview: [www.w3.org/MarkUp/SGML](http://www.w3.org/MarkUp/SGML)
- SIAM Society for Industrial and Applied Mathematics. Homepage: [www.siam.org](http://www.siam.org)
- SIGIR Special Interest Group on Information Retrieval. Homepage: [www.acm.org/sigir/](http://www.acm.org/sigir/)
- SIGMOD Special Interest Group on Management of Data. Homepage: [www.acm.org/sigmod](http://www.acm.org/sigmod)
- SOAP Simple Object Access Protocol. Homepage: [www.w3.org/TR/SOAP](http://www.w3.org/TR/SOAP)
- SPS SharePoint Server. Homepage: [www.microsoft.com/sharepoint](http://www.microsoft.com/sharepoint)
- SQL Structured Query Language.
- SRC Systems Research Center. Homepage: [research.compaq.com/SRC/home.html](http://research.compaq.com/SRC/home.html)
- SRI Stanford Research Institute.
- STC Suffix Tree Clustering.
- TCP/IP Transfer Control Protocol/Internet Protocol (Stevens and Wright 2001).

- TF Term Frequency (Baeza-Yates and Ribeiro-Neto 1999).
- TF.IDF Term Frequency Inverse Document Frequency. The product of TF and IDF (Baeza-Yates and Ribeiro-Neto 1999).
- TKC Tightly Knit Community (Effect) (Lempel and Moran 2000).
- TLD Top Level Domain. e.g. the `.com` in `java.sun.com`.
- TREC Text REtrieval Conference. Homepage: [trec.nist.gov](http://trec.nist.gov)
- TSP Travelling Salesman Problem.
- TV TeleVision.
- UCL University College London. Homepage: [www.ucl.ac.uk](http://www.ucl.ac.uk)
- UCS Universal Character Set.
- UIUC University of Illinois at Urbana-Champaign. Homepage: [www.uiuc.edu](http://www.uiuc.edu)
- UI User Interface.
- ULU University of London Union
- UR Universal Relation (Ullman 1989; Levene 1992).
- URL Uniform Resource Locator. Describes the location of web pages (Berners-Lee 1994).
- UTF UCS Translation Format(s).
- VLC Very Large Collection. A TREC track replaced by the Web track.
- VLDB Very Large DataBases. Homepage: [www.vldb.informatik.hu-berlin.de](http://www.vldb.informatik.hu-berlin.de)
- VRML Virtual Reality Modeling Language.
- VSM Vector Space Model (Baeza-Yates and Ribeiro-Neto 1999).
- W3C World Wide Web Consortium. Homepage: [www.w3.org](http://www.w3.org)
- WCC Weakly Connected Component (Aho, Hopcroft, and Ullman 1983).
- WCRE Working Conference on Reverse Engineering. Homepage: [www.cs.ualberta.ca/wcre2003/](http://www.cs.ualberta.ca/wcre2003/)
- WIMP Windows Icons Menus Pointer.
- WYSIWYG What You See Is What You Get.
- XML eXtensible Markup Language (Harold and Means 2001). Homepage: [www.w3c.org/XML](http://www.w3c.org/XML)
- XSL eXtensible Stylesheet Language (Harold and Means 2001).
- XSLT eXtensible Stylesheet Language Transformations (Harold and Means 2001).

## Appendix B

# List of Mathematical Symbols

- $\wp$  Power set.
- $r^2$  Pearson's product moment correlation coefficient.
- $\rho$  Spearman's non-linear correlation measure.
- $M'$  The transpose of the matrix,  $M$ .
- $\emptyset$  The empty set.
- $\{x\}$  The set containing the value  $x$ .
- $\{x|f(x)\}$  The set containing all possible values for which  $f(x)$  is true.
- $|s|$  The number of elements in the set  $s$ .
- $\cup$  The union of two sets.
- $x \in S$  True if the element  $x$  is in the set  $S$ .
- $A : B \rightarrow C$  A function,  $A$ , taking an argument of type  $B$  and returning a value of type  $C$ .
- $\top, \perp$  True and false.
- $\wedge$  Logical "and".
- $\neg$  Logical negation.
- $\exists$  Logical existence.
- $O(n)$  Time or space in the order of  $n$ .
- $\sum$  Summation.
- $\leftarrow$  Assignment.



## Appendix C

# Linear Correlation between Ranking Metrics

	PR	In	Out	Pg	Gr	Hub	Auth	Foley	Log(PR)	Log(In)	Log(Out)	Log(Pg)	Log(Gr)	Log(Hub)	Log(Auth)	Log(Foley)
PR	1.00	0.34	0.08	0.12	0.32	0.03	0.34	0.11	-0.93	0.00	0.01	-0.05	-0.03	0.01	-0.00	0.00
In		1.00	0.22	0.35	0.93	0.07	0.99	0.30	0.04	0.00	0.13	0.16	0.29	0.13	0.01	0.00
Out			1.00	0.71	0.23	0.26	0.22	0.49	0.00	0.00	0.31	0.39	0.23	0.31	0.02	0.00
Pg				1.00	0.41	0.77	0.36	0.66	0.00	0.01	0.73	0.85	0.60	0.74	0.04	0.01
Gr					1.00	0.23	0.93	0.54	0.03	0.00	0.27	0.30	0.52	0.27	0.02	0.00
Hub						1.00	0.08	0.56	-0.01	0.01	0.89	0.95	0.70	0.90	0.05	0.01
Auth							1.00	0.30	0.04	0.00	0.12	0.16	0.28	0.12	0.01	0.00
Foley								1.00	0.00	0.01	0.58	0.63	0.75	0.59	0.03	0.01
Log(PR)									1.00	0.00	0.03	0.11	0.13	0.03	0.00	0.00
Log(In)										1.00	0.01	0.01	0.02	0.01	0.24	1.00
Log(Out)											1.00	0.95	0.72	0.99	0.05	0.01
Log(Pg)												1.00	0.75	0.96	0.05	0.01
Log(Gr)													1.00	0.73	0.05	0.02
Log(Hub)														1.00	0.05	0.01
Log(Auth)															1.00	0.24
Log(Foley)																1.00

Figure C.1: Correlation between Web metrics on the Sleepycat webcase using Pearson's product moment correlation coefficient

	PR	In	Out	Pg	Gr	Hub	Auth	Foley	Log(PR)	Log(In)	Log(Out)	Log(Pg)	Log(Gr)	Log(Hub)	Log(Auth)	Log(Foley)
PR	1.00	0.08	0.01	0.00	0.01	0.00	0.01	0.01	-0.99	0.00	-0.01	-0.10	-0.11	-0.01	0.00	0.00
In		1.00	0.09	0.21	0.51	0.20	0.47	0.33	0.00	0.02	0.19	0.26	0.59	0.19	0.02	0.01
Out			1.00	0.28	0.07	0.16	0.07	0.86	-0.00	-0.03	0.27	0.44	0.15	0.27	-0.03	0.00
Pg				1.00	0.42	0.95	0.44	0.40	-0.00	-0.01	0.16	0.50	0.39	0.16	-0.01	0.00
Gr					1.00	0.46	0.99	0.41	-0.00	0.00	0.05	0.21	0.52	0.05	0.00	0.00
Hub						1.00	0.47	0.33	-0.00	0.00	0.12	0.45	0.40	0.12	0.00	0.00
Auth							1.00	0.40	-0.00	0.00	0.04	0.20	0.48	0.04	0.00	0.00
Foley								1.00	-0.00	0.00	0.19	0.36	0.36	0.19	0.00	0.00
Log(PR)									1.00	0.00	0.02	0.11	0.15	0.02	0.00	0.00
Log(In)										1.00	-0.06	-0.05	0.06	-0.06	1.00	0.59
Log(Out)											1.00	0.83	0.42	1.00	-0.06	-0.01
Log(Pg)												1.00	0.56	0.83	-0.05	0.01
Log(Gr)													1.00	0.42	0.06	0.03
Log(Hub)														1.00	-0.06	-0.01
Log(Auth)															1.00	0.59
Log(Foley)																1.00

Figure C.2: Correlation between Web metrics on the SCSIS webcase using Pearson's product moment correlation coefficient

	PR	In	Out	Pg	Gr	Hub	Auth	Foley	Log(PR)	Log(In)	Log(Out)	Log(Pg)	Log(Gr)	Log(Hub)	Log(Auth)	Log(Foley)
PR	1.00	0.03	0.00	0.00	0.00	0.00	0.01	0.00	-1.00	0.00	-0.00	-0.02	-0.02	-0.00	-0.00	0.00
In		1.00	0.05	0.03	0.20	0.01	0.55	0.16	0.00	0.00	0.04	0.06	0.21	0.04	0.01	0.00
Out			1.00	0.11	0.05	0.11	0.04	0.49	-0.00	-0.01	0.25	0.41	0.19	0.25	0.01	-0.00
Pg				1.00	0.97	0.05	0.01	0.11	-0.00	0.00	0.05	0.17	0.17	0.05	0.00	0.00
Gr					1.00	0.03	0.18	0.08	0.00	0.00	0.03	0.13	0.19	0.03	0.01	0.00
Hub						1.00	0.04	0.39	-0.00	0.01	0.19	0.48	0.48	0.20	0.02	0.00
Auth							1.00	0.09	0.00	0.00	0.01	0.02	0.09	0.01	0.00	0.00
Foley								1.00	0.00	0.00	0.14	0.33	0.33	0.14	0.02	0.00
Log(PR)									1.00	0.00	0.02	0.02	0.00	0.00	0.00	0.00
Log(In)										1.00	-0.10	-0.03	0.06	-0.04	0.28	0.66
Log(Out)											1.00	0.79	0.29	0.96	-0.05	-0.04
Log(Pg)												1.00	0.53	0.79	0.02	-0.00
Log(Gr)													1.00	0.31	0.11	0.04
Log(Hub)														1.00	0.03	0.00
Log(Auth)															1.00	0.19
Log(Foley)																1.00

Figure C.3: Correlation between Web metrics on the UCL webcase using Pearson's product moment correlation coefficient

	PR	In	Out	Pg	Gr	Hub	Auth	Foley	Log(PR)	Log(In)	Log(Out)	Log(Pg)	Log(Gr)	Log(Hub)	Log(Auth)	Log(Foley)
PR	1.00	0.02	0.00	0.00	0.00	0.00	0.00	0.01	-1.00	-0.00	-0.00	-0.03	-0.03	-0.00	-0.00	-0.00
In		1.00	0.10	0.05	0.23	0.01	0.56	0.40	0.00	0.00	0.06	0.11	0.21	0.06	0.01	0.00
Out			1.00	0.38	0.03	0.51	0.01	0.79	-0.00	0.00	0.19	0.36	0.21	0.19	0.02	0.00
Pg				1.00	0.78	0.31	0.01	0.34	0.00	0.00	0.05	0.15	0.11	0.06	0.01	0.00
Gr					1.00	0.01	0.22	0.16	0.00	0.00	0.01	0.05	0.09	0.02	0.00	0.00
Hub						1.00	0.01	0.42	-0.00	0.00	0.02	0.05	0.04	0.02	0.00	0.00
Auth							1.00	0.40	0.00	0.00	0.01	0.02	0.04	0.01	0.00	0.00
Foley								1.00	0.00	0.00	0.10	0.26	0.27	0.11	0.02	0.00
Log(PR)									1.00	0.00	0.01	0.03	0.04	0.01	0.00	0.00
Log(In)										1.00	-0.03	0.00	0.03	-0.01	0.14	0.81
Log(Out)											1.00	0.73	0.29	0.95	0.04	0.00
Log(Pg)												1.00	0.51	0.75	0.07	0.01
Log(Gr)													1.00	0.31	0.13	0.02
Log(Hub)														1.00	0.07	0.01
Log(Auth)															1.00	0.11
Log(Foley)																1.00

Figure C.4: Correlation between Web metrics on the UCL-CS webcase using Pearson's product moment correlation coefficient

	PR	In	Out	Pg	Gr	Hub	Auth	Foley	Log(PR)	Log(In)	Log(Out)	Log(Pg)	Log(Gr)	Log(Hub)	Log(Auth)	Log(Foley)
PR	1.00	0.20	0.00	-0.00	0.14	-0.00	0.01	0.14	-0.98	0.00	0.00	-0.03	-0.01	0.00	-0.00	0.00
In		1.00	0.00	-0.00	0.72	-0.00	0.18	0.77	0.02	0.00	0.00	0.01	0.16	0.00	0.00	0.00
Out			1.00	0.93	0.01	0.12	-0.00	0.04	-0.00	0.01	0.01	0.93	0.04	0.06	0.01	0.01
Pg				1.00	-0.00	0.00	-0.00	0.02	-0.00	0.00	0.00	0.87	-0.01	0.02	0.01	0.00
Gr					1.00	-0.00	0.05	0.67	0.01	0.00	0.00	0.02	0.31	0.01	0.00	0.00
Hub						1.00	-0.00	0.01	-0.00	0.00	0.00	0.07	-0.02	0.01	0.00	0.00
Auth							1.00	0.01	0.00	0.00	0.00	-0.00	0.02	0.00	0.00	0.00
Foley								1.00	0.01	0.00	0.00	0.06	0.32	0.01	0.00	0.00
Log(PR)									1.00	0.00	0.00	0.03	0.03	0.00	0.00	0.00
Log(In)										1.00	1.00	0.02	0.01	0.10	0.27	1.00
Log(Out)											1.00	0.02	0.01	0.10	0.27	1.00
Log(Pg)												1.00	0.11	0.15	0.02	0.02
Log(Gr)													1.00	0.04	0.01	0.01
Log(Hub)														1.00	0.04	0.10
Log(Auth)															1.00	0.27
Log(Foley)																1.00

Figure C.5: Correlation between Web metrics on the Intel webcase using Pearson's product moment correlation coefficient

	PR	In	Out	Pg	Gr	Hub	Auth	Foley	Log(PR)	Log(In)	Log(Out)	Log(Pg)	Log(Gr)	Log(Hub)	Log(Auth)	Log(Foley)
PR	1.00	0.07	0.00	0.00	0.00	0.00	0.00	0.01	-1.00	0.00	-0.00	-0.02	-0.02	-0.00	-0.00	0.00
In		1.00	0.01	0.01	0.15	-0.00	0.03	0.20	0.01	0.00	0.03	0.04	0.17	0.03	0.01	0.00
Out			1.00	0.11	0.01	0.04	-0.00	0.35	-0.00	-0.01	0.11	0.21	0.07	0.12	0.00	0.00
Pg				1.00	0.03	0.00	-0.00	0.06	-0.00	0.00	0.03	0.13	0.05	0.03	0.00	0.00
Gr					1.00	-0.00	0.00	0.38	0.00	0.00	0.01	0.01	0.09	0.01	0.00	0.00
Hub						1.00	-0.00	0.01	-0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00
Auth							1.00	0.00	-0.00	0.00	-0.00	0.02	-0.00	0.00	0.00	0.00
Foley								1.00	0.00	0.00	0.09	0.17	0.10	0.01	0.00	0.00
Log(PR)									1.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00
Log(In)										1.00	-0.11	-0.04	0.07	-0.07	0.40	0.68
Log(Out)											1.00	0.89	0.44	0.96	-0.08	-0.05
Log(Pg)												1.00	0.54	0.91	0.02	-0.00
Log(Gr)													1.00	0.47	0.10	0.05
Log(Hub)														1.00	0.02	-0.02
Log(Auth)															1.00	0.28
Log(Foley)																1.00

Figure C.6: Correlation between Web metrics on the Birkbeck webcase using Pearson's product moment correlation coefficient

	PR	In	Out	Pg	Gr	Hub	Auth	Foley	Log(PR)	Log(In)	Log(Out)	Log(Pg)	Log(Gr)	Log(Hub)	Log(Auth)	Log(Foley)
PR	1.00	0.12	0.01	0.00	0.08	-0.00	0.00	0.06	-0.99	0.00	-0.00	-0.06	-0.05	-0.00	-0.00	0.00
In		1.00	0.13	0.09	0.68	-0.00	0.08	0.49	0.01	0.01	0.09	0.12	0.35	0.10	0.01	0.00
Out			1.00	0.35	0.23	0.28	-0.00	0.49	-0.00	-0.06	0.58	0.74	0.42	0.58	0.00	-0.00
Pg				1.00	0.74	0.02	-0.00	0.34	-0.00	-0.00	0.21	0.51	0.53	0.21	0.01	0.00
Gr					1.00	-0.00	0.02	0.51	0.01	0.01	0.14	0.36	0.55	0.14	0.01	0.00
Hub						1.00	-0.00	0.06	-0.00	0.00	0.01	0.03	0.00	0.01	0.00	0.00
Auth							1.00	0.01	-0.00	0.00	-0.00	0.04	-0.00	0.00	0.00	0.00
Foley								1.00	0.00	0.01	0.38	0.57	0.66	0.39	0.03	0.00
Log(PR)									1.00	0.00	0.01	0.07	0.08	0.01	0.00	0.00
Log(In)										1.00	-0.08	-0.07	0.04	-0.08	0.37	0.29
Log(Out)											1.00	0.86	0.41	0.99	-0.01	-0.01
Log(Pg)												1.00	0.64	0.87	0.01	-0.00
Log(Gr)													1.00	0.42	0.06	0.01
Log(Hub)														1.00	0.03	-0.00
Log(Auth)															1.00	0.11
Log(Foley)																1.00

Figure C.7: Correlation between Web metrics on the DTI webcase using Pearson's product moment correlation coefficient



	In	Out	Pg	Gr	Hub	Auth	Foley	Log(In)	Log(Out)	Log(Pg)	Log(Gr)	Log(Hub)	Log(Auth)	Log(Foley)
In	1.00	0.10	0.05	0.26	0.01	0.57	0.46	0.00	0.09	0.11	0.25	0.09	0.00	0.00
Out		1.00	0.42	0.04	0.53	0.01	0.77	0.00	0.24	0.34	0.21	0.24	0.01	0.00
Pg			1.00	0.76	0.32	0.01	0.36	0.00	0.11	0.18	0.13	0.11	0.00	0.00
Gr				1.00	0.01	0.23	0.20	0.00	0.03	0.06	0.12	0.03	0.00	0.00
Hub					1.00	0.01	0.36	0.00	0.03	0.07	0.04	0.03	0.00	0.00
Auth						1.00	0.45	0.00	0.01	0.02	0.05	0.01	0.00	0.00
Foley							1.00	0.00	0.24	0.34	0.37	0.24	0.01	0.00
Log(In)								1.00	0.00	0.00	0.00	0.00	0.07	1.00
Log(Out)									1.00	0.93	0.40	1.00	0.02	0.00
Log(Pg)										1.00	0.49	0.93	0.02	0.00
Log(Gr)											1.00	0.41	0.04	0.00
Log(Hub)												1.00	0.03	0.00
Log(Auth)													1.00	0.07
Log(Foley)														1.00

Figure C.8: Correlation between Web metrics on the JDK 1.4 webcase using Pearson's product moment correlation coefficient

## Appendix D

# Non-Linear Correlation between Ranking Metrics

	PR	In	Out	Pg	Gr	Hub	Auth	Foley
PR	1.00	0.67	0.49	0.48	0.69	0.47	0.34	0.65
In		1.00	0.60	0.60	0.85	0.59	0.54	0.78
Out			1.00	0.95	0.59	0.92	0.21	0.65
Pg				1.00	0.59	0.96	0.22	0.66
Gr					1.00	0.59	0.53	0.89
Hub						1.00	0.21	0.64
Auth							1.00	0.45
Foley								1.00

Figure D.1: Correlation between Web metrics on the Sleepycat webcase using Kendall's Tau Statistics

	PR	In	Out	Pg	Gr	Hub	Auth	Foley
PR	1.00	0.84	0.65	0.64	0.87	0.63	0.47	0.84
In		1.00	0.65	0.65	0.94	0.65	0.68	0.89
Out			1.00	0.98	0.68	0.97	0.36	0.74
Pg				1.00	0.68	0.99	0.39	0.74
Gr					1.00	0.67	0.65	0.97
Hub						1.00	0.39	0.73
Auth							1.00	0.56
Foley								1.00

Figure D.2: Correlation between Web metrics on the Sleepycat webcase using Spearman's Rho

	PR	In	Out	Pg	Gr	Hub	Auth	Foley
PR	1.00	0.36	0.26	0.26	0.52	0.24	-0.18	0.44
In		1.00	0.64	0.64	0.44	0.62	0.13	0.53
Out			1.00	0.98	0.22	0.96	0.10	0.46
Pg				1.00	0.23	0.96	0.10	0.47
Gr					1.00	0.21	-0.04	0.70
Hub						1.00	0.12	0.44
Auth							1.00	-0.01
Foley								1.00

Figure D.3: Correlation between Web metrics on the SCSIS webcase using Kendall's Tau Statistics

	PR	In	Out	Pg	Gr	Hub	Auth	Foley
PR	1.00	0.58	0.25	0.26	0.68	0.24	-0.29	0.68
In		1.00	0.51	0.51	0.63	0.50	-0.02	0.58
Out			1.00	1.00	0.21	0.99	0.00	0.40
Pg				1.00	0.22	0.99	0.00	0.40
Gr					1.00	0.21	-0.09	0.88
Hub						1.00	0.02	0.39
Auth							1.00	-0.11
Foley								1.00

Figure D.4: Correlation between Web metrics on the SCSIS webcase using Spearman's Rho

	PR	In	Out	Pg	Gr	Hub	Auth	Foley
PR	1.00	-0.41	-0.82	-0.82	-0.33	-0.82	-0.18	-0.32
In		1.00	0.46	0.46	0.71	0.46	0.46	0.62
Out			1.00	0.99	0.33	0.99	0.13	0.44
Pg				1.00	0.33	0.99	0.13	0.44
Gr					1.00	0.33	0.58	0.69
Hub						1.00	0.13	0.44
Auth							1.00	0.41
Foley								1.00

Figure D.5: Correlation between Web metrics on the JDK 1.4 webcase using Kendall's Tau Statistics

	PR	In	Out	Pg	Gr	Hub	Auth	Foley
PR	1.00	0.22	0.55	0.65	0.21	0.65	0.10	0.20
In		1.00	0.38	0.37	0.86	0.37	0.64	0.77
Out			1.00	0.87	0.26	0.87	0.06	0.43
Pg				1.00	0.28	0.99	0.06	0.44
Gr					1.00	0.28	0.76	0.85
Hub						1.00	0.06	0.44
Auth							1.00	0.58
Foley								1.00

Figure D.6: Correlation between Web metrics on the JDK 1.4 webcase using Spearman's Rho

	PR	In	Out	Pg	Gr	Hub	Auth	Foley
PR	1.00	0.36	0.17	0.18	0.33	0.17	-0.01	0.33
In		1.00	0.42	0.42	0.76	0.40	0.32	0.64
Out			1.00	0.93	0.25	0.84	0.27	0.39
Pg				1.00	0.25	0.89	0.23	0.40
Gr					1.00	0.23	0.40	0.88
Hub						1.00	0.30	0.38
Auth							1.00	0.36
Foley								1.00

Figure D.7: Correlation between Web metrics on the UCL webcase using Spearman's Rho

	PR	In	Out	Pg	Gr	Hub	Auth	Foley
PR	1.00	0.32	0.18	0.18	0.25	0.17	-0.27	0.21
In		1.00	0.36	0.36	0.79	0.36	0.46	0.63
Out			1.00	1.00	0.27	0.98	0.18	0.34
Pg				1.00	0.27	0.98	0.19	0.34
Gr					1.00	0.28	0.58	0.89
Hub						1.00	0.21	0.34
Auth							1.00	0.55
Foley								1.00

Figure D.8: Correlation between Web metrics on the UCL-CS webcase using Spearman's Rho

	PR	In	Out	Pg	Gr	Hub	Auth	Foley
PR	1.00	0.61	0.03	0.03	0.43	0.09	0.42	0.06
In		1.00	0.09	0.09	0.79	0.03	0.70	0.24
Out			1.00	1.00	0.15	0.41	0.10	0.62
Pg				1.00	0.15	0.42	0.10	0.62
Gr					1.00	0.07	0.60	0.52
Hub						1.00	0.23	0.31
Auth							1.00	0.24
Foley								1.00

Figure D.9: Correlation between Web metrics on the Intel webcase using Spearman's Rho

	PR	In	Out	Pg	Gr	Hub	Auth	Foley
PR	1.00	0.41	0.22	0.22	0.52	0.21	-0.25	0.67
In		1.00	0.74	0.75	0.35	0.74	0.25	0.49
Out			1.00	1.00	0.05	0.96	0.14	0.34
Pg				1.00	0.06	0.96	0.13	0.34
Gr					1.00	0.07	0.02	0.80
Hub						1.00	0.19	0.34
Auth							1.00	-0.02
Foley								1.00

Figure D.10: Correlation between Web metrics on the Birkbeck webcase using Spearman's Rho

	PR	In	Out	Pg	Gr	Hub	Auth	Foley
PR	1.00	0.56	0.39	0.40	0.39	0.38	0.21	0.36
In		1.00	0.41	0.41	0.82	0.39	0.32	0.68
Out			1.00	1.00	0.39	0.98	0.25	0.52
Pg				1.00	0.40	0.98	0.24	0.52
Gr					1.00	0.36	0.22	0.89
Hub						1.00	0.28	0.50
Auth							1.00	0.18
Foley								1.00

Figure D.11: Correlation between Web metrics on the DTI webcase using Spearman's Rho



# Bibliography

- Abiteboul, S., P. Buneman, and D. Suciu (2000). *Data on the Web: From Relations to Semistructured Data and XML*. San Francisco, Ca.: Morgan-Kaufmann.
- Acksyn, R., D. McCracken, and E. Yoder (1988, July). Kms : A distributed hypermedia system for managing knowledge in organizations. *Communications of the ACM* 31(7), 820–835.
- Adamic, L. A. (2000). Zipf, power-laws, and pareto - a ranking tutorial. Technical report, Internet Ecologies Area, Xerox Palo Alto Research Center, 3333 Coyote Hill Rd., Palo Alto, CA 94304.
- Adamic, L. A. (2002). *Network Dynamics: The World Wide Web*. Ph. D. thesis, Stanford.
- Agrawal, R. and R. Srikant (2002). Searching with numbers. In *Proceedings of International World Wide Web Conference*, pp. 420–431.
- Agrawal, S., S. Chaudhuri, and G. Das (2002). Dbxplorer: A system for keyword-based search over relational databases. In *Proceedings of IEEE International Conference on Data Engineering*, pp. 5–16.
- Aho, A. V., J. E. Hopcroft, and J. D. Ullman (1983, January). *Data Structures and Algorithms*. Addison-Wesley Pub Co.
- Albert, R., A.-L. Barabási, and H. Jeong (2000). Scale-free characteristics of random networks: the topology of the world-wide web. *Physica A* 281, 69–77.
- Albert, R., H. Jeong, and A.-L. Barabási (2000, July). Error and attack tolerance of complex networks. *Nature* 406, 378–382.
- Alvestrand, H. (1995, March). Tags for the identification of languages. RFC# 1766.
- Anderson, C. R. and E. Horvitz (2002). Web montage: A dynamic personalized start page. In *Proceedings of International World Wide Web Conference*, Honolulu, HI.
- Anh, V. N. and A. Moffat (2002). Impact transformation: effective and efficient web retrieval. In *Proceedings of the 25th annual ACM/SIGIR conference on Research and Development in Information Retrieval*, Tampere, Finland, pp. 3 – 10. ACM Press.
- Apache Software Foundation (2003). Apache ant.
- Ashish, N. and C. A. Knoblock (1997). Semi-automatic wrapper generation for internet information sources. In *Conference on Cooperative Information*

- Systems*, pp. 160–169.
- Autonomy (2003). Autonomy technology white paper.
- Baeza-Yates, R. and B. Ribeiro-Neto (1999). *Modern Information Retrieval*. Reading, Ma.: ACM Press and Addison Wesley.
- Baeza-Yates, R., B. Ribeiro-Neto, and G. Navarro (1999). Indexing and searching. In R. Baeza-Yates and B. Ribeiro-Neto (Eds.), *Modern Information Retrieval*, pp. 191–228. Reading, Ma.: ACM Press and Addison Wesley.
- Bailey, P., N. Craswell, and D. Hawking (2001). Engineering a multi-purpose test collection for web retrieval experiments. To appear.
- Bailliez, S., N. K. Barozzi, J. Bergeron, S. Bodewig, P. Chanezon, J. D. Davidson, T. Dimock, P. Donald, D. Gillard, E. Hatcher, D. Holt, B. Kelly, A. J. Kuiper, C. MacNeill, S. Mazzocchi, E. Meade, S. Ruby, N. Seessle, J. S. Stevens, M. Umasankar, R. Vaughn, D. Walend, P. Wells, and C. Strong (2002). Apache ant 1.5.1 manual.
- Bar-Yossef, Z. and S. Rajagopalan (2002). Template detection via data mining and its applications. In *Proceedings of International World Wide Web Conference*, Honolulu, HI, pp. 580–591.
- Bergman, M. K. (2000, July). The deep web: Surfacing hidden value. White paper, Bright Planet.
- Berners-Lee, T. (1989, March). Information management: A proposal. Technical report, CERN.
- Berners-Lee, T. (1994). Uniform resource locators. Technical report, CERN. RFC# 1738.
- Berners-Lee, T. (1996). Rfc-1945 hypertext transfer protocol - http/1.0. RFC# 1738.
- Berners-Lee, T. (1999). *Weaving the Web: The original design and Ultimate Destiny of the World Wide Web*. London: Orion Books.
- Bernstein, M. (1990, November). An apprentice that discovers hypertext links. In A. Rizk, N. A. Streitz, and J. André (Eds.), *Hypertext : Concepts, Systems and Applications : Proceedings of the First European Conference on Hypertext*, The Cambridge Series on Electronic Publishing. Cambridge University Press.
- Bharat, K., A. Broder, M. Henzinger, P. Kumar, and S. Venkatasubramanian (1998). The connectivity server: fast access to linkage information on the web. In *Proceedings of 7th International World Wide Web Conference*, pp. 14–18.
- Bieman, J. and V. Murdock (2001, November). Finding code on the world wide web: a preliminary investigation. In *Proceedings of the First International Workshop on Source Code Analysis and Manipulation (SCAM)*, Florence, Italy.
- Boldi, P. and S. Vigna (2003). The webgraph framework i: Compression techniques. Technical Report 293-03, Dipartimento di Scienze dell’Informazione, Università di Milano.
- Borges, J. (2000). *A Data Mining Model to Capture User Web Navigation Patterns*. Ph. D. thesis, University College London.

- Botafogo, R., E. Rivlin, and B. Shneiderman (1992). Structural analysis of hypertexts: Identifying hierarchies and useful metrics. *ACM Transactions on Information Systems* 10, 142–180.
- Briand, L., J. Daly, and J. Wust (1999). A unified framework for coupling measurement in object-oriented systems. *IEEE Transactions on Software Engineering* 25(1), 91–121.
- Briand, L., P. Devanbu, and W. Melo (1997). An investigation into coupling measures for C++. In *Proceedings of the 19th International Conference on Software Engineering (ICSE'97)*, Boston, USA, pp. 412–421.
- Brin, S. and L. Page (1998). The anatomy of a large-scale hypertextual web search engine. In *Proceedings of International World Wide Web Conference*, Brisbane, pp. 107–117.
- Broder, A. (1997). On the resemblance and containment of documents. In *Proceedings of Compression and Complexity of Sequences (SEQUENCES'97)*, pp. 21–29.
- Broder, A., S. Glassman, and M. Manasse (1997). Clustering the web. Technical report, Compaq SRC.
- Broder, A., R. Kumar, F. Maghoul, P. Raghavan, A. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener (2000, May). Graph structure in the web. In *Proceedings of the 9th World Wide Web Conference*, Amsterdam, pp. 309–320.
- Broder, A. Z. (2000). Identifying and filtering near-duplicate documents. In *Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching*, pp. 1–10.
- Brownell, D. (2002, January). *SAX2*. 101 Morris Street, Sebastapol, CA 95472: O'Reilly & Associates Inc.
- Burke, C. B. (1994, June). *Information and Secrecy : Vannevar Bush, Ultra, and the Other Memex*. Scarecrow Press.
- Bush, V. (1945). As we may think. *Atlantic Monthly* 76, 101–108.
- Cailliau, R. (1995). A little history of the world wide web.
- Carroll, L. (1865). *Alice's Adventures in Wonderland*. Candlewick Press. Gutenberg Etext #929.
- Cavnar, W. B. and J. M. Trenkle (1994). N-gram-based text categorization. In *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*.
- Chakrabarti, S., S. Srivastava, M. Subramanyam, and M. Tiwari (2000). Using memex to archive and mine community web browsing experience. In *Proceedings of International World Wide Web Conference*, pp. 669–684.
- Chen, M., M. Hearst, J. Hong, and J. Lin (1999). Cha-cha: A system for organizing intranet search results. In *USENIX Symposium on Internet Technologies and Systems*.
- Chidamber, S. R. and C. F. Kemerer (1991). Towards a metrics suite for object-oriented design. In *OOPSLA '91*, Phoenix, Arizona, pp. 197–211.
- Chidlovskii, B., U. Borghoff, and P. Chevalier (1997). Towards sophisticated wrapping of web-based information repositories.

- Chittaro, L., R. Ranon, and L. Leronutti (2003). Guiding visitors of web3d worlds through automatically generated tours. In *Web3D proceedings*.
- Cho, J., H. Garcia-Molina, and L. Page (1998). Efficient crawling through URL ordering. In *Proceedings of International World Wide Web Conference*, Brisbane, pp. 161–172.
- Cleverdon, C. (1997). The cranfield tests on index language devices. In K. Sparck Jones and P. Willett (Eds.), *Readings in Information Retrieval*, pp. 47–59. San Francisco, Ca.: Morgan-Kaufmann.
- Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM* 13(6), 377–387.
- Coleridge, S. T. (1816). Kubla khan.
- Conklin, J. (1987). Hypertext: An introduction and survey. *IEEE Computer* 20, 17–41.
- Cooper, A. (1999). *The Inmates are Running the Asylum*. Sams.
- Craswell, N. and D. Hawking (2002). Overview of the trec-2002 web track. In E. M. Voorhees and L. P. Buckland (Eds.), *NIST Special Publication: SP 500-251, The Eleventh Text Retrieval Conference (TREC 2002)*. Gaithersburg, Maryland: Department of Commerce, National Institute of Standards and Technology.
- Crestani, F., M. Lalmas, C. J. V. Rijsbergen, and I. Campbell (1998, December). “is this document relevant? ...probably”: a survey of probabilistic models in information retrieval. *ACM Computing Surveys (CSUR)* 30, 528–552.
- Cutler, M., H. Deng, S. Maniccam, and W. Meng (1999). A new study on using html structures to improve retrieval. In *The Eleventh IEEE International Conference on Tools with Artificial Intelligence*, pp. 406–409.
- Cutler, M., Y. Shih, and W. Meng (1997). Using the structure of html documents to improve retrieval. In *Usenix Symposium on Internet Technologies and Systems (USITS 97)*.
- Damashek, M. (1995). Gauging similarity via n-grams: Language-independent sorting, categorization and retrieval of text. *Science* 267, 843–848.
- Davison, B. D. (2000, July). Recognizing nepotistic links on the web. In *Proceedings of the AAAI-2000 Workshop on Artificial Intelligence for Web Search*, Austin, TX, pp. 23–28. AAAI Press.
- Dorigo, M., V. Maniezzo, and A. Colorni (1996). The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics* 26(1), 29–41.
- Dougherty, D. and A. Robbins (1997, March). *Sed & Awk* (2nd ed.). 101 Morris Street, Sebastapol, CA 95472: O’Reilly & Associates Inc.
- Drexler, K. E. (1987). Hypertext publishing and the evolution of knowledge. *Social Intelligence* 1(2), 87–120. Last updated in September 1996.
- Drineas, P., A. Frieze, R. Kannan, S. Vempala, and V. Vinay (1999, January). Clustering in large graphs and matrices. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, Baltimore, Maryland, pp. 291–299.

- Drori, O. (2000). Using text elements by context to display search results. In *Information Doors (a workshop proceedings held in conjunction with the ACM Hypertext 2000 and ACM Digital Libraries 2000 conferences)*, San Antonio, Texas, USA., pp. 17–22.
- Edmunson, H. (1969). New methods in automatic indexing. In I. Mani and M. T. Maybury (Eds.), *Advances in Automatic Text Summarization*, pp. 23–42. Cambridge, MA 02142: MIT Press.
- Eisenberg, A. and J. Melton (2002). Sql/xml is making good progress. *SIGMOD Record* 31(2), 101–108.
- Engelbart, D. C. (1962). Augmenting human intellect : A conceptual framework. Technical report, Stanford Research Institute, Menlo Park, California 94025, USA.
- Engelbart, D. C. and W. K. English (1968, December). A research center for augmenting human intellect. In *Proceedings of the 1968 Fall Joint Computer Conference*, San Francisco, CA, pp. 395–410.
- Everitt, B. (1998). *The Cambridge Dictionary of Statistics*. The Edinburgh Building, Cambridge CB2 2RU, UK: Cambridge University Press.
- Fagan, J. L. (1987). Automatic phrase indexing for document retrieval. In C. Yu and C. van Rijsbergen (Eds.), *Proceedings of the 10th Annual ACM/SIGIR Conference on Research and Development in Information Retrieval*, pp. 91–101.
- Fagin, R., A. R. Karlin, J. Kleinberg, P. Raghavan, S. Rajogopalan, R. Rubinfeld, M. Sudan, and A. Tomkins (2000). Random walks with back buttons. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*.
- Fagin, R., R. Kumar, K. S. McCurley, J. Novak, D. Sivakumar, J. A. Tomlin, and D. P. Williamson (2003). Searching the workplace web. In *Proceedings of the World Wide Web Conference*, Budapest, Hungary.
- Fairhead, H. (1990, December). Screen hypes. *Computer Shopper*, 135–141.
- Flake, G., C. L. Giles, and S. Lawrence (2000). Efficient identification of web communities. In *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining*.
- Flake, G. W., S. Lawrence, C. L. Giles, and F. M. Coetzee (2002, March). Self organization of the web and identification of communities. *IEEE Computer* 35(3), 66–71.
- Flanagan, D. (1997). *Java in a Nutshell* (2nd ed.). 101 Morris Street, Sebastapol, CA 95472: O'Reilly & Associates Inc.
- Foley, J. D., A. van Dam, S. K. Feiner, and J. F. Hughes (1990). *Computer Graphics, Principles and Practice* (2nd ed.). Reading, Mass., U.S.A.: Addison-Wesley.
- Fowler, M., K. Beck, J. Brant, W. Opdyke, and D. Roberts (1999, June). *Refactoring: Improving the Design of Existing Code* (1st ed.). Addison-Wesley Pub Co.
- Fowler, M., D. Rice, M. Foemmel, E. Hieatt, R. Mee, and R. Stafford (2002, November). *Patterns of Enterprise Application Architecture* (1st ed.). Addison Wesley Professional.

- Freed, N. and N. Borenstein (1996, November). Multipurpose internet mail extensions (mime) part two: Media types. RFC# 2046.
- Friedl, J. E. F. (2002, July). *Mastering Regular Expressions* (2nd ed.). 101 Morris Street, Sebastopol, CA 95472: O'Reilly & Associates Inc.
- Friendly, L. (1995, June). The design of distributed hyperlinked programming documentation. In *Proceedings of The International Workshop on Hypermedia Design in Montpellier*, France. Sun Microsystems, Inc.
- Frost, R. (1875). The road not taken.
- Furuta, R., F. M. Shipman III, C. C. Marshall, D. Brenner, and H. Hsieh (1997, April). Hypertext paths and the world-wide web: Experiences with walden's paths. In *Proceedings of the Eighth ACM Conference on Hypertext*, Southampton, U.K., pp. 167–176.
- Gamma, E., R. Helm, R. Johnson, and J. Vlissides (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. One Jacob Way, Reading, Massachusetts 01867: Addison Wesley Longman, Inc.
- Gansner, E. R., E. Koutsofios, S. C. North, and K.-P. Vo (1993). A technique for drawing directed graphs. *IEEE Trans. Software Engineering* 19 19(3), 214–230.
- Geisler, G. (2000). Enriched links: A framework for improving web navigation using pop-up views. Technical report, Interaction Design Laboratory, School of Information and Library Science, University of North Carolina, U.S.A.
- Goldman, R., N. Shivajumar, S. Venkatasubramanian, and H. Garcia-Molina (1998). Proximity search in databases. In *Proceedings of the 24th Intl. Conf. on Very Large Databases (VLDB)*, New York, U.S.A.
- Goldman, R. and J. Widom (2000). Wsq/dsq: A practical approach for combined querying of database and the web. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 285–296.
- Google (2002a). Google toolbar.
- Google (2002b). Terms of service.
- Gordon, M. and P. Pathak (1999, March). Finding information on the world wide web: The retrieval effectiveness of search engines. *Information Processing and Management* 2(35), 141–180.
- Gori, M., M. Maggini, and E. Martinelli (1999). Nautilus - navigate autonomously and target interesting links for users. Technical report, Dipartimento di Ingegneria dell'Informazione, Universit di Siena. Technical Report RT-DII-20/99.
- Greenspun, P. (1994). We have chosen shame and will get war. Submitted to 2nd International World Wide Web Conference (Rejected).
- Grossman, D. A. and O. Frieder (1998). *Information Retrieval: Algorithms and Heuristics*. Boston: Kluwer Academic Publishers.
- Guillaume, J.-L., M. Latapy, and L. Viennot (2002). Efficient and simple encodings for the web graph. In *Poster Proceedings of the International World Wide Web Conference*.
- Guinan, C. and A. F. Smeaton (1992). Information retrieval from hypertext using dynamically planned guided tours. In *Proceedings of the ACM European*

*Conference on Hypertext.*

- Gurry, M. and P. Corrigan (1996). *Oracle Performance Tuning*. 101 Morris Street, Sebastapol, CA 95472: O'Reilly & Associates Inc.
- Halasz, F. G. (1988). Notecards: A multimedia idea processing environment. In S. Ambron and K. Hoop (Eds.), *Interactive Multimedia*. Microsoft Press.
- Hamilton, J. R. and T. K. Nayak (2001, December). Microsoft sql server full-text search. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering's Special Issue on Text and Databases* 24(4), 7–10.
- Hammond, N. and L. Allinson (1988, June). Travels around a learning support environment: Rambling, orienteering or touring. In *Chi'88 Conference Proceedings: Human Factors in Computing Systems*, pp. 269–273. Association for Computing Machinery.
- Hammond, N. and L. Allinson (1989a, September). Extending hypertext for learning: An investigation of access and guidance tools. In *Proceedings of the BCS HCI conference*, Nottingham, pp. 293–304. Cambridge University Press.
- Hammond, N. and L. Allinson (1989b). Extending hypertext for learning: An investigation of access and guidance tools. In V. Sutcliffe and L. Macaulay (Eds.), *People and Computers*. Cambridge: Cambridge University Press.
- Harman, D., E. Fox, R. Baeza-Yates, and W. Lee (1992). Inverted files. In R. Baeza-Yates and W. Frakes (Eds.), *Information Retrieval: Data Structures & Algorithms*, pp. 28–43. Upper Saddle River, NJ: Prentice Hall.
- Harold, E. R. and W. S. Means (2001). *XML in a Nutshell*. 101 Morris Street, Sebastapol, CA 95472: O'Reilly & Associates Inc.
- Harrison, R., S. J. Counsell, and R. Nithi (1998). Coupling metrics for OO design. In *IEEE International Symposium on Software Metrics*, Bethesda, Maryland, US, pp. 150–157.
- Haveliwala, T. H. (1999). Efficient computation of pagerank. Technical report, Stanford University.
- Haveliwala, T. H. (2002). Topic-sensitive pagerank. In *Proceedings of International World Wide Web Conference*, Hawaii.
- Hawking, D., N. Craswell, P. Bailey, and K. Griffiths (2001). Measuring search engine quality. *Information Retrieval* 4(1), 33–59.
- Hearst, M. (1995, May). Tilebars: Visualization of term distribution information in full text information access. In *Proceedings of the Conference on Human Factors in Computing Systems*, Denver, Colorado, USA.
- Hearst, M., J. Pedersen, and D. Karger (1995, November). Scatter/gather as a tool for the analysis of retrieval results. In *Working Notes of the AAAI Fall Symposium on AI Applications in Knowledge Navigation*, Cambridge, MA, U.S.A.
- Hearst, M. and P. Pedersen (1996). Reexamining the cluster hypothesis: Scatter/gather on retrieval results. In *Proceedings of 19th Annual ACM/SIGIR Conference on Research and Development in Information Retrieval*, Zurich.
- Hearst, M. A. (1997, March). Scientific american - interfaces for searching the web. *Scientific American*, 68–72.

- Hearst, M. A. and C. Karadi (1997, July). Cat-a-cone : An interactive interface for specifying searches and viewing retrieval results using a large category hierarchy. In *Proceedings of 20th Annual ACM/SIGIR Conference on Research and Development in Information Retrieval*, Philadelphia, PA, U.S.A.
- Heather, M. A. and B. N. Rossiter (1990). Database support for very large hypertexts: Data organization, navigation and trails. *Electronic Publishing - ODD 3 : 3*, 141–154.
- Henzinger, M., A. Heydon, M. Mitzenmacher, and M. Najork (1999). Measuring index quality using random walks on the web. In *Proceedings of International World Wide Web Conference*, Montreal, pp. 1291–1303.
- Herbert, F. (1965). *Dune*. Ace Books.
- Heydon, A. and M. Najork (1999a). Mercator: A scalable, extensible web crawler. *World Wide Web 2*, 219–229.
- Heydon, A. and M. Najork (1999b). Performance limitations of the java core libraries. In *Proceedings of the ACM Java Grande Conference*, pp. 35–41.
- Hirai, J., S. Raghavan, A. Paepcke, and H. Garcia-Molina (2000, May). Webbase : A repository of web pages. In *Proceedings of the 9th International World Wide Web Conference*, Amsterdam, pp. 277–293.
- Hopcroft, J. and J. Ullman (1979). *Introduction to Automata Theory, Languages and Computation*. Reading, Ma.: Addison-Wesley.
- Howe, D. (1993). The free on-line dictionary of computing.
- Hristidis, V. and Y. Papakonstantinou (2002). Discover: Keyword search in relational databases. In *Proceedings of the 28th VLDB Conference*, Hong Kong.
- Huberman, B., P. Pirollo, J. Pitkow, and R. Lukose (1998). Strong regularities in world wide web surfing. *Science 280*, 95–97.
- Hulgeri, A., G. Bhaltoia, C. Nakhe, S. Chakrabarti, and S. Sudarshan (2001). Keyword search in databases. *Bulletin of the Technical Committee on Data Engineering. Special Issue on Imprecise Queries 24 No. 3*, 22–32.
- Jakobsson, M. (2002, June). Fractal hash sequence representation and traversal. In *Proceedings of IEEE International Symposium on Information Theory*, Palais de Beaulieu, Lausanne, Switzerland.
- Jansen, B. J., A. Spink, and T. Saracevic (1998). Failure analysis in query construction: Data and analysis from a large sample of web queries. In *Digital Libraries 98*, Pittsburgh, PA, USA.
- Jeh, G. and J. Widom (2001). Simrank: A measure of structural-context similarity. Technical report, Stanford InfoLab.
- Joachims, T., D. Freitag, and T. Mitchell (1997). WebWatcher: A tour guide for the World Wide Web. In *Proceedings of International Joint Conference on Artificial Intelligence*, Nagoya, Japan, pp. 770–775.
- Joachims, T., T. Mitchell, and D. Freitag (1995). Webwatcher: A learning apprentice for the world wide web. In *AAAI Spring Symposium on Information Gathering*.
- Johnson, R. and B. Foote (1988). Designing reusable classes. *Journal of Object-Oriented Programming 1*(2), 22–35.



- Johnson, R. and W. Opdyke (1993a). Refactoring and aggregation. In *Object Technologies for Advanced Software*, Number 742 in Lecture Notes in Computer Science, pp. 264–278. Springer Verlag.
- Johnson, R. E. and W. F. Opdyke (1993b). Creating abstract superclasses by refactoring. In *Proceedings of the ACM Computer Science Conference, CSC'93*.
- Jones, K. S., S. Walker, and S. E. Robertson (2000). A probabilistic model of information retrieval: development and comparative experiments - part 2. *Information Processing and Management* 36(6), 809–840.
- Junod, D. N. (1992). Amigaguide : A hypertext documentation system.
- Kahn, P. and K. Lenk (2001). *Mapping Web Sites*. Rue du Bugnon 7, CH-1299 Crans-Pres-Celigny, Switzerland: Rotovision SA.
- Kamvar, S. D., T. H. Haveliwala, and G. H. Golub (2003, April). Adaptive methods for the computation of pagerank. Technical report, Stanford University.
- Kamvar, S. D., T. H. Haveliwala, C. D. Manning, and G. H. Golub (2003a, March). Exploiting the block structure of the web for computing pagerank. Technical report, Stanford University.
- Kamvar, S. D., T. H. Haveliwala, C. D. Manning, and G. H. Golub (2003b). Extrapolation methods for accelerating pagerank computations. In *Proceedings of the World Wide Web Conference*, Budapest.
- Kawai, H., S. Akamine, K. Kida, K. Matsuda, and T. Fukushima (2002). Development and evaluation of the withair mobile search engine. In *Poster Proceedings of International World Wide Web Conference*, Honolulu, HI. Internet Systems Research Laboratories, NEC Corp.
- Kim, S. and B.-T. Zhang (2000, August). Web-document retrieval by genetic learning of importance factors for html tags. In A.-H. Tan and P. S. Yu (Eds.), *PRICAI 2000 Workshop on Text and Web Mining*, Melbourne, pp. 13–23.
- Kleinberg, J., D. Gibson, and P. Raghavan (1998). Inferring web communities from link topology. In *UK Conference on Hypertext*, pp. 225–234.
- Kleinberg, J. M. (1998). Authoritative sources in a hyperlinked environment. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, pp. 668–677.
- Koch, G. and K. Loney (2002). *Oracle 9i : The Complete Reference*. Osborne McGraw-Hill.
- Koster, M. (1994, June). Robot exclusion.
- Kushmerick, N. (1997). *Wrapper Induction for Information Extraction*. Ph. D. thesis, University of Washington.
- Kwok, C., O. Etzioni, and D. S. Weld (2001, May). Scaling question answering to the web. In *Proceedings of the 10th International World Wide Web Conference*, Hong Kong.
- L'Amour, L. (1984, June). *Ride the Dark Trail* (Reissue ed.). Bantam Books.
- Lamping, J., R. Roa, and P. Pirolli (1995). A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *CHI '95*, ACM,

- New York (1995) 401408.
- Larry Wall, Tom Christiansen, J. O. (2000, July). *Programming Perl* (3rd ed.). 101 Morris Street, Sebastapol, CA 95472: O'Reilly & Associates Inc.
- Lawrence, S., K. Bollacker, and C. L. Giles (1999, November). Indexing and retrieval of scientific literature. In *Eighth International Conference on Information and Knowledge Management, CIKM 99*, Kansas City, Missouri, pp. 139–146.
- Lawrence, S. and C. L. Giles (1998). Searching the world wide web. *Science* 280, 98–100.
- Lawrence, S. and C. L. Giles (1999). Accessibility of information on the web. *Nature* 400, 107–109.
- Leighton, V. and J. Srivastava (1999). First 20 precision among world wide web search services. *Journal of the American Society for Information Science* 10(50), 870–881.
- Lempel, R. and S. Moran (2000). The stochastic approach for link-structure analysis (SALSA) and the TKC effect. In *Proceedings of the 9th International World Wide Web Conference*, pp. 387–402.
- Levene, M. (1992). *The Nested Universal Relation Model*, Volume 595 of *Lecture Notes in Computer Science*. Berlin: Springer-Verlag.
- Levene, M., J. Borges, and G. Loizou (2001). Zipf's law for web surfers. *Knowledge and Information Systems* 3, 120–129.
- Levene, M., T. I. Fenner, G. Loizou, and R. Wheeldon (2002). A stochastic model for the evolution of the web. *Computer Networks and ISDN Systems* 39, 277–287.
- Levene, M. and G. Loizou (1999). Navigation in hypertext is easy only sometimes. *SIAM Journal on Computing* 29, 728–760.
- Levene, M. and G. Loizou (2002). Kemeny's constant and the random surfer. *American Mathematical Monthly* 109, 741–745.
- Levene, M. and R. Wheeldon (2001). A Web site navigation engine. In *Poster Proceedings of International World Wide Web Conference*, Hong Kong.
- Levene, M. and R. Wheeldon (2003). Navigating the world-wide-web. In M. Levene and A. Poulouvasilis (Eds.), *Web Dynamics*. Springer-Verlag.
- Levene, M. and N. Zin (2001, June). A navigation engine for assessing the quality of a trail between linked pages.
- Ley, M. (2002). Digital library and bibliography project.
- Li, W.-S., K. S. Candan, Q. Vu, and D. Agrawal (2001). Retrieving and organizing web pages by "information unit". In *Proceedings of International World Wide Web Conference*, Hong Kong, pp. 230–244.
- Lim, L., M. Wang, S. Padmanabhan, J. S. Vitter, and R. Agarwal (2003). Dynamic maintenance of web indexes using landmarks. In *Proceedings of the World Wide Web Conference*, Budapest, Hungary.
- Luhn, H. (1958). The automatic creation of literature abstracts. In I. Mani and M. T. Maybury (Eds.), *Advances in Automatic Text Summarization*, pp. 15–21. Cambridge, MA 02142: MIT Press.

- Mase, H. (1998). Experiments on automatic web page categorization for ir system. Technical report, Stanford University.
- Masier, A. and D. Simmen (2001, December). Db2 optimization in support of full text search. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering's Special Issue on Text and Databases* 24(4), 3–6.
- Mat-Hassan, M. and M. Levene (2001). Can navigational assistance improve search experience: A user study. *First Monday* 6(9).
- Mauldin, M. (1997). Lycos: Design choices in an internet search service. *IEEE Expert Online* 1(12).
- Mayfield, J. and P. McNamee (1997). N-gram vs. words as indexing terms. In *TREC-6 Conference Notebook Papers*.
- Mayfield, J. and P. McNamee (1999). Indexing using both n-grams and words. In *The Seventh Text REtrieval Conference (TREC-7). NIST Special Publication 500-242*, pp. 419–423, 1999.
- Mayfield, J. and P. McNamee (2003, July). Single n-gram stemming. In *Proceedings of the 26th Annual ACM/SIGIR Conference on Research and Development in Information Retrieval*, Toronto, Ontario, pp. 415–416.
- Meyrowitz, N. K. (1986). Intermedia: The architecture and construction of an object-oriented hypermedia system and application framework. In *Proceedings of the ACM Conference on Object Oriented Programming Systems Languages and Applications*, pp. 186–201. ACM Press.
- Microsoft Corporation (2001, May). Introducing microsoft sharepoint portal server 2001.
- Miller, E., D. Shen, J. Liu, and C. Nicholas (2000, July). Performance and scalability of a large-scale n-gram based information retrieval system. *Journal of Digital Information* 1.
- Mizuuchi, Y. and K. Tajima (1999). Finding context paths for web pages. In *Proceedings of the ACM Hypertext Conference*.
- Mladenec, D. (1998, October). *Machine Learning on non-homogeneous, distributed text data*. Ph. D. thesis, University of Ljubljana, Slovenia.
- Mladenec, D. (1999, July). Machine learning used by personal webwatcher. In *Proceedings of ACAI-99 Workshop on Machine Learning and Intelligent Agents*, Chania, Crete.
- Moreau, L. and W. Hall (1998). On the expressiveness of links in hypertext systems. *The Computer Journal* 41(7), 459–473.
- Muchowski, J. and A. Smith (1994). *Indy Workstation Owner's Guide*. Silicon Graphics, Inc.
- Mukherjea, S. and J. Foley (1995). Showing the context of nodes in the world-wide web. In *Proceedings of ACM Conference on Human Factors in Computing Systems*, Denver, Colorado.
- Mukherjea, S. and Y. Hara (1997). Focus + context views of world-wide web nodes. In *Proceedings of UK Conference on Hypertext*, pp. 187–196.
- Myers, A. C. (1990). Gdiff man page.
- Najjar, R., S. Counsell, G. Loizou, and K. Mannock (2003, March). The role of constructors in the context of refactoring object-oriented systems. In

- Proceedings of the 7th European Conference on Software Maintenance and Reengineering*, Benevento, Italy, pp. 111–120.
- Najork, M. and A. Heydon (2001). High performance web crawling. In J. Abello, P. M. Pardalos, and M. G. Resende (Eds.), *Handbook of Massive Data Sets*. Kluwer Academic Publishers, Inc.
- Najork, M. and J. L. Wiener (2001). Breadth-first search crawling yields high quality pages. In *Proceedings of the International World-Wide-Web Conference*, pp. 114–118.
- Nelson, T. H. (1965). A file structure for the complex, the changing and the indeterminate. In *Proceedings of the 1965 20th national conference*, Cleveland, Ohio, United States, pp. 84–100.
- Nelson, T. H. (1993). *Literary Machines* (93.1 ed.). Mindful Press.
- Nelson, T. H. (1995). *Lemonade*.
- Nelson, T. H. (1999, December). Xanalogical structure, needed now more than ever : Parallel documents, deep links to content, deep versioning and deep re-use. *ACM Computing Surveys (CSUR)* 31.
- Nielsen, J. (1989, November). The matters that really matter for hypertext usability. In *Hypertext '89 Proceedings*.
- Nielsen, J. (1997). Search and you may find. useit.com alertbox.
- Nielsen, J. (2000). *Designing Web Usability: The Practice of Simplicity*. Indianapolis, Indiana: New Riders Publishing.
- Nielsen, J. (2001, May). Search: Visible and simple. Useit.com Alertbox.
- Nielsen, J. (2002, January). Site map usability. useit.com Alertbox.
- Nyce, J. and P. Kahn (1991). *From Memex to Hypertext: Vannevar Bush and the Mind's Machine*. San Diego, Ca.: Academic Press.
- O'Donoghue, D., A. Leddy, J. Power, and J. Waldron (2002, June). Bi-gram analysis of java bytecode sequences. In *Principles and Practice of Programming in Java*, Trinity College Dublin.
- Opdyke, W. (1992). *Refactoring object-oriented frameworks*. Ph. D. thesis, University of Illinois, Urbana-Champaign, IL, USA.
- Oram, A. and S. Talbott (1993). *Managing Project with Make*. O'Reilly.
- Over, P. and J. Yen (2003, May). An introduction to duc 2003: Intrinsic evaluation of generic news text summarization systems. In *Proceedings of DUC 2003: Workshop on Text Summarization*, Edmonton, Canada.
- Page, L. (1998, January). Method for node ranking in a linked database.
- Page, L., S. Brin, R. Motwani, and T. Winograd (1998). The pagerank citation ranking: Bringing order to the web. Working paper, Department of Computer Science, Stanford University.
- Pandurangan, G., P. Raghavan, and E. Upfal (2002). Using pagerank to characterize web structure. In *Proceedings of the 8th Annual International Computing and Combinatorics Conference (COCOON)*.
- Pinkerton, B. (1994). Finding what people want: Experiences with the webcrawler. In *Proceedings of the Second International World Wide Web Conference*, Chicago.

- Pinkerton, B. (2002). *WebCrawler: Finding What People Want*. Ph. D. thesis, University of Washington.
- Pollock, A. and A. Hockley (1997, March). What's wrong with internet searching. In *Designing for the Web : Empirical Studies*, Microsoft Corporate Headquarters, Redmond, Washington. Human Factors Unit, BT Laboratories, Ipswich, UK.
- Porter, M. F. (1980). An algorithm for suffix stripping. *Program* 14(3), 130–137.
- Potantin, A. (2002). The fox - a tool for object graph analysis. Technical report, Victoria University of Wellington. BSc. Honours report.
- Potantin, A., J. Noble, M. R. Freat, and R. Biddle (2003). Scale-free geometry in object-oriented programs. Submitted to CACM.
- Pratt, W., M. Hearst, and L. M. Fagan (1999, July). A knowledge-based approach to organizing retrieved documents. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, Orlando, Florida, U.S.A.
- Pretschner, A. and S. Gauch (1999, December). Personalization on the web. Technical report, Information and Telecommunication Technology Center (ITTC), The University of Kansas, Lawrence, KS.
- Radev, D., W. Fan, H. Qi, H. Wu, and A. Grewal (2002). Probabilistic question answering on the web. In *Proceedings of the International World Wide Web Conference*.
- Raghavan, S. and H. Garcia-Molina (2001, September). Crawling the hidden web. In *Proceedings of the 27th Intl. Conf. on Very Large Databases (VLDB)*, pp. 129–138.
- Raghavan, P. (2001, December). Structured and unstructured search in enterprises. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering's Special Issue on Text and Databases* 24(4), 15–18.
- Randall, K. H., R. Stata, R. Wickremesinghe, and J. L. Wiener (2002, April). The link database: Fast access to graphs of the web. In *Proceedings of the Data Compression Conference*, Snao Bird, Utah.
- Raymond, E. S. (1998, March). The cathedral and the bazaar. *First Monday* 3(3).
- Raymond, E. S. (2001, January). *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary* (Revised ed.). 101 Morris Street, Sebastapol, CA 95472: O'Reilly & Associates.
- R.Durbin and D. Willshaw (1987, April). An analogue approach to the travelling salesman problem using an elastic net method. *Nature* 326(16), 689.
- Reiser, H. (2001). The naming system venture. White Paper.
- Richardson, M. and P. Domingos (2002). The intelligent surfer: Probabilistic combination of link and content information in pagerank. *Advances in Neural Information Processing Systems* (14). To Appear.
- Risvik, K. M. and R. Michelsen (2002, June). Search engines and web dynamics. *Computer Networks* 39(3), 289–302.
- Robertson, S. E. and S. Walker (1999). Okapi/keenbow at trec-8. In *Proceedings of the Eighth Text REtrieval Conference*, pp. 151–162.

- Sagiv, Y. (1983). A characterization of globally consistent databases and their access paths. *ACM Transactions on Database Systems* 8, 266–286.
- Salton, G. and C. Buckley (1998). Term weighting approaches in automatic text retrieval. *Information Processing and Management* 24, 513–523.
- Sarda, N. L. and A. Jain (2001). Mragyati : A system for keyword-based searching in databases. *Computing Research Repository cs.DB/0110052*.
- Shakespeare, W. (1599). As you like it.
- Shaw, S. (1998). A generic robot architecture. Technical report, University College London. c316 BSc Project Report.
- Shiozawa, H., H. Nishiyama, and Y. Matsushita (2001). The natto view: An architecture for interactive information visualization. *IPSJ JOURNAL* 38(11).
- Shirazi, J. (2000). *Java Performance Tuning*. O'Reilly.
- Shivakumar, N. and H. Garcia-Molina (1999). Finding near-replicas of documents on the web. In *WEBDB: International Workshop on the World Wide Web and Databases, WebDB*. LNCS.
- Sillitoe, T., B. N. Rossiter, and M. A. Heather (1990). Trail management in hypertext: Database support for navigation through textual complex objects. In A. Brown and P. Hitchcock (Eds.), *Proceedings of the 8th British National Conference on Databases*, pp. 224–242.
- Silverstein, C., M. Henzinger, H. Marais, and M. Moricz (1999). Analysis of a very large altavista query log. *SIGIR Forum* 33(1), 6–12.
- Singhal, A. (2001). Modern information retrieval: A brief overview. *IEEE Data Engineering Bulletin* 4(24), 35–43.
- Singhal, A., C. Buckley, and M. Mitra (1996). Pivoted document length normalization. In *Research and Development in Information Retrieval*, pp. 21–29.
- Skene, J. (2001). How to implement a component. Technical report, Navigation Zone, Ltd.
- Sleepycat Software (2001). *Berkeley DB*. New Riders Publishing.
- Smith, M. K., D. McGuinness, R. Volz, and C. Welty (2002, November). Web ontology language (owl) guide. Technical report, W3C.
- Spertus, E. and L. A. Stein (2000). Squeal : A structure query language for the web. In *Proceedings of the 9th International World Wide Web Conference*, pp. 95–103.
- Spink, A., J. Bateman, and B. J. Jansen (1998). Searching heterogeneous collections on the web: behaviour of excite users. *Information Research: An Electronic Journal* 2(5).
- Spink, A., B. J. Jansen, D. Wolfram, and T. Saracevic (2002, March). From e-sex to e-commerce: Web search changes. *IEEE Computer* 3(35).
- Stevens, R. and G. R. Wright (2001, November). *TCP/IP Illustrated*. Addison Wesley Professional.
- Stevenson, R. L. (1850–1894). Virginibus puerisque and other papers : El dorado. *Project Gutenberg Etext #386*.
- Stotts, P. and R. Furata (1989). Petri-net based hypertext: Structure with browsing semantics. *ACM Transactions on Information Systems* 7, 3–29.

- Stotts, P. D., R. Furuta, and J. C. Ruiz (1992). Hyperdocuments as automata: Trace-based browsing property verification. In *Proceedings of European Conference on Hypertext*, pp. 272–281.
- Sullivan, D. (2001, December). Search engine sizes. Technical report.
- Sun Microsystems Inc. (2001). Javadoc tool home page. <http://java.sun.com/j2se/javadoc/>.
- Swinehart, D., P. T. ZellWeger, R. Beach, and R. Hagmann (1986, October). A structural view of the Cedar programming environment. *ACM Transactions on Programming Languages and Systems* 8(4), 419–490.
- Thompson, T. and N. Baran (1988, November). The next computer. *Byte*.
- Tokuda, L. and D. Batory (2001). Evolving object-oriented designs with refactorings. *Automated Software Engineering* 8, 89–120.
- Tolkien, J. (1954). *The Lord of the Rings*. HarperCollins.
- Tomasi, M. D. and B. Mehlenbacher (1999). Re-engineering online documentation: Designing examples-based online support systems. *Technical Communication*, 46 1(46), 55–66.
- Tombros, A. (1997). Reflecting user information needs through query biased summaries. Master’s thesis, Department of Computing Science, University of Glasgow.
- Tombros, A. and M. Sanderson (1998, August). The advantages of query-biased summaries in information retrieval. In *Proceedings of the 21st Annual ACM SIGIR Conference on Research and Development in Information Retrieval*, Melbourne, Australia, pp. 2–10. ACM Press.
- Trigg, R. H. (1988, October). Guided tours and tabletops : Tools for communicating in a hypertext environment. *ACM Transactions of Office Information Systems* 6(4), 398–414.
- Trigg, R. H. and M. Weiser (1986, January). Textnet: A network-based approach to text handling. *ACM Transactions on Office Information Systems* 4(1), 1–23.
- Ullman, J. (1989). *Principles of Database and Knowledge-Base Systems*, Volume 2. Rockville, Md.: Computer Science Press.
- Valverde, S., R. Ferrer-Cancho, and R. V. Sole (2002, April). Scale-free networks from optimal design. *Condensed Matter Archive cond-mat/0204344*. Submitted to Europhysics Letters.
- Van Rijsbergen, C. J. (1979). *Information Retrieval, 2nd edition*. Dept. of Computer Science, University of Glasgow.
- Voorhees, E. M. (1999, December). Natural language processing and information retrieval. In J. G. Carbonell, J. Siekmann, and M. T. Pazienza (Eds.), *Information Extraction: Towards Scalable, Adaptable Systems (Lecture Notes in Artificial Intelligence)*, pp. 32–48. Springer Verlag.
- Wachowski, A. and L. Wachowski (1999).
- Walker, J. H. (1987, November). Document examiner: Delivery interface for hypertext documents. In *Hypertext’87 Proceedings*, Chapel Hill, North Carolina, USA, pp. 307–323. ACM.

- Weinreich, H. and W. Lamersdorf (2000). Concepts for improved visualization of web link attributes. In *Proceedings of International the 9th World Wide Web Conference*, pp. 403–418.
- Wexelbat, A. and P. Maes (1999). Footprints: History-rich tools for information foraging. In *Proceedings of CHI'99*, Pittsburgh, PA, U.S.A.
- Wheeldon, R. (1999). Report on the hparobot. Technical report, University College London. c316 BSc Project Report.
- Wheeldon, R. and S. Counsell (2003a, June). Making refactoring decisions in large-scale java systems: an empirical stance. *Computing Research Repository cs.SE/0306098*.
- Wheeldon, R. and S. Counsell (2003b, September). Power law distributions in class relationships. In *Proceedings of 3rd International Workshop on Source Code Analysis and Manipulation (SCAM)*, Amsterdam, pp. 45–54. IEEE Computer Society.
- Wheeldon, R., S. Counsell, and K. Keenoy (2003, September). Autocode: Using memex-like trails to improve program comprehension. In A. van Deursen, C. Knight, J. I. Maletic, and M.-A. Storey (Eds.), *Proceedings of 2nd Annual Designfest on Visualizing Software for Understanding and Analysis (VIS-SOFT)*, Amsterdam, pp. 48–49.
- Wheeldon, R. and M. Levene (2003, November). The best trail algorithm for adaptive navigation in the world-wide-web. In *Proceedings of 1st Latin American Web Congress*, Santiago, Chile.
- Wheeldon, R., M. Levene, and K. Keenoy (2003, July). Search and navigation in relational databases. *Computing Research Repository cs.DB/0307073*.
- Wheeldon, R., M. Levene, and N. Zin (2002). Autodoc: A search and navigation tool for web-based program documentation. In *Poster Proceedings of International World Wide Web Conference*, Honolulu, HI.
- Wolf, G. (1995, June). The curse of xanadu. *Wired*.
- Wolfram, D., A. Spink, B. J. Jansen, and T. Saracevic (2001, October). Vox populi: The public searching of the web. *Journal of the American Society for Information Science and Technology* 12(52), 1073–1074.
- Wolpert, D. H. and W. G. Macready (1995). No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe, NM.
- Wolpert, D. H. and W. G. Macready (1997, April). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation* 1(1), 67–82.
- Wong, W. Y. P. and D. L. Lee (1993). Implementations of partial document ranking using inverted files. *Information Processing and Management* 29(5), 647–669.
- Wood, L. (1998). Spacesearch.
- Yahoo! (2001). The history of yahoo! - how it all started...
- Yang, C. C. and F. L. Wang (2003). Fractal summarization for mobile devices to access large documents on the web. In *Proceedings of the World Wide Web Conference*, Budapest, Hungary.



- Zachary, G. P. (1999). *Endless Frontier: Vannevar Bush, Engineer of the American Century*. Cambridge, Massachusetts: MIT Press.
- Zamir, O. and O. Etzioni (1999). Grouper: a dynamic clustering interface to Web search results. *Computer Networks and ISDN Systems* 31, 1361–1374.
- Zawinski, J. (2000). Java sucks.
- Zellweger, P. T. (1989, November). Scripted documents: A hypermedia path mechanism. In *Hypertext '89 Proceedings*, 3333 Coyote Hill Rd., Palo Alto, CA. 94304.
- Zheng, Z. (2002). Developing a web-based question answering system. In *Proceedings of the International World Wide Web Conference*, Honolulu, HI.
- Zin, N. and M. Levene (1999). Constructing web views from automated navigation sessions. In *ACM Digital Library Workshop on Organizing Web Space 1999 (WOWS)*, pp. 54–58.

# Index

- n*-grams, 36, 161
- ActiveWebcase, 120, 121
- Aggregation, 241
- Algorithm
  - Best Trail, 75
  - Page Filter, 133
  - Text Summarization, 136
- AllTheWeb, 137
- Ant colony, 73, 96, 124
- Ask Jeeves, 137
- Augmentation Research Centre, 25
- Authorities, 44
- Automata, 65
  
- Bag-of-Words, 36
- BANKS, 224, 232
- Berners-Lee, Tim, 18, 29
- Birkbeck, 90, 107, 108
- Bow-Tie Model, 30
- Bush, Vannevar, 18, 23
  
- CiteSeer, 224
- Conklin, 27, 30
- Coupling, 240, 252
- Crawling, 34, 116, 205, 217
  - Incremental, 146
  
- DBLP, 212, 219, 221, 224
- DbSurfer, 216, 235
- Document Examiner, 26
- DTI, 90, 107
  
- Engelbart, Douglas, 25, 62, 239
- Enquire, 29
  
- Gain Rank, 106
- Google, 45, 137, 268
  - Tool bar, 150
- GraphViz, 71, 154
- Guided Tour, 27, 50, 62
  
- HES, 26
- HITS, 44, 46, 100, 106
- HTML, 29
  - Weighting schemes for, 43
- HTTP, 29
- Hubs and Authorities, 44
  
- Information Retrieval, 36
- Inheritance, 241
- Intel, 90, 107
- Interface, 241
- Intermedia, 26
- IREngine, 120
  
- Java2HTML, 248
- Javadoc, 90, 109, 239, 242, 244, 248, 262
- JDK, 90, 107, 108
- Join Discovery Problem, 216
  
- Kleinberg, 44
  
- Lost in Hyperspace, 50
- Lycos, 137
  
- Memex, 18, 23
- Mercator, 116, 205
- Metasearch, 33
- Microsoft Office, 133, 215
  
- Navigation Problem, 18, 50
- NavSearch, 116, 150, 172, 244, 248
- Nelson, Ted, 23, 239
- Nodes, Landmark, 44
- NoteCards, 26
  
- PageRank, 34, 45, 100, 106
  - efficient computation of, 45
  - topic-specific, 45
- PageRanks, 112
- Parameter type, 241
- PDF, 133, 215

- Pinkerton, 33, 34, 46, 108
- Postscript, 133, 215
- Potential Gain, 98, 100, 102, 104, 106, 259
  - Query Specific, 112
  - Site based, 112
- Power law, 31, 32, 104, 252
- Precision, 39
- Preferential Attachment, 32
  
- Rank Sinks, 45
- Rapid Selector, 23
- Recall, 39
- Refactoring, 241, 259
- Relevance, 37
- Resource Discovery, 33
- Resource Discovery Problem, 18
- Return type, 241
  
- SALSA, 46
- SCC, 30
- SCSIS, 90, 107, 108
- Search Engines, 33
- Security, 235
- SharePoint, 213, 226
- Shockwave Flash, 133, 215
- Similarity, 37
- Site Maps, 51
- Sleepycat, 90, 107, 108
- Strongly Connected Component, 30
- Sum Distinct, 80, 81, 83, 90
  
- Tar, 134
- Teoma, 137
- tf.idf, 37
- Title
  - HTML tag, 43
  - Short, 143, 161
- TKC Effect, 46
- TrailAlgorithm, 120, 121
- TrailNode, 120
- Transclusion, 23
- Transcopyright, 23
- TREC, 40, 90, 104, 107, 108
  
- UCL, 90, 107, 108
- Udanax, 23
  
- Vector Space Model, 37
  
- Vertical Trailer, 213, 226, 235
  
- WebCrawler, 33, 46
- Weighted Sum, 80, 81, 83, 86, 90
- WT10g, 104
- WT10g, TREC, 40
  
- Xanadu, 23
  - Curse of, 23
  
- Zip, 134
- ZOG, 26