Name Matching in an X.500 White Pages Directory

Paul Barker
University College London

Thesis submitted for the degree of Doctor of Philosophy
February 1997
Abstract

The expansion of data communications networks, the Internet in particular, has encouraged the development of a number of information retrieval services. One of the key services is a white pages directory service: a user provides the name and organisation of the person they are looking for, and the service returns details such as telephone and facsimile numbers, electronic and paper mail addresses, and so on.

While many aspects of directory services have attracted a lot of research and design effort, there has been relatively little attention focused on the problem of how best to use the name matching facilities provided by the directory service systems to find the directory entries that users require.

There are essentially three components to the name matching problem. First, we need to know how users formulate their queries: for example, do users tend to use full name forms, abbreviations or sets of initials? How much of a problem is misspelled input? Second, we also need to know the sort of names that directory administrators store in the directory: the directory's distributed management means that name formats vary from organisation to organisation. Third, a directory service provides a set of facilities for matching user input to directory names: which are the most effective facilities; can we devise strategies that deliver the correct results, and do so without returning many spurious results?

The core work in this thesis is empirical. My experimentation is based on the use of X.500, the international standard for directory services. I have gathered data from the NameFLOW-Paradise directory, a well-established X.500 directory. I use query and directory name data taken from this service in a series of experiments to test various name matching strategies. The experiments are based on the facilities provided by X.500.

The similarity between the name matching facilities provided by X.500 and other directory services means that the findings should be broadly applicable to non-X.500 directory services. This is particularly true of the study of approximate matching. Several directory services, including X.500, allow for approximate matching but do not specify which algorithm should be used to implement this type of matching. In practice, Soundex has been widely used. However, as many directory users and administrators have been unsatisfied with Soundex, I have investigated whether any alternative algorithms have better matching characteristics in the white pages paradigm.
I would like to thank a number of friends and colleagues for the help and support they offered me while I was working on this thesis: I needed it!

In particular, I'd like to thank: Graham Knight, my supervisor, for gently keeping me on the rails; Professor Peter Kirstein for commenting on an earlier draft, and for loaning me the workstation essential to my work; Professor Jon Crowcroft for commenting on the penultimate draft; Rex Galbraith for advice on statistical methods; James Sil­lence for commenting on early drafts of several chapters; Colin Robbins, Alan Turland and Tim Howes for advice on Quipu; David Chadwick for advice on X.500's nooks and crannies; Geoff Dowling for helping initiate my research on approximate matching; Karen Kukich for providing me with several papers on approximate matching; my colleagues on the PARADISE project at ULCC for allowing me to tinker with the DE software used to provide the PARADISE DUA service.

I also need to thank my family. The arrival of my sons meant that I stayed in and worked during evenings that I might otherwise have spent in the pub. Last, and far from least, I have to thank Vicky for her proof-reading and comments, and also for her forbearance as I locked myself away in the garret with papers and computer.
CONTENTS

2.7 The Role of the DUA in Name Matching ................................................................. 53
  2.7.1 Precision and Recall .......................................................................................... 53
  2.7.2 DUA Support when Phrasing Queries ............................................................... 53
  2.7.3 Pre-processing Users' Input .............................................................................. 54
  2.7.4 Post-processing Results from the Directory ....................................................... 55
  2.7.5 How Hard Does the DUA Try to Match User Input to Entries? ..................... 55
2.8 The Directory Environment ...................................................................................... 55
  2.8.1 Response Times ................................................................................................. 56
  2.8.2 Administrator Limits on Searching and Listing ............................................... 59
2.9 Comparison with Other Directory Systems ............................................................ 60
  2.9.1 Querying Operations ......................................................................................... 60
  2.9.2 Types of Matching ............................................................................................ 61
  2.9.3 Attribute Types ............................................................................................... 61
  2.9.4 Object Classes .................................................................................................. 61
  2.9.5 Booleans ........................................................................................................... 61
  2.9.6 Other Controls .................................................................................................. 61
  2.9.7 Directory Environment Issues ........................................................................ 62
  2.9.8 Facilities Not Provided by These Systems ...................................................... 62
  2.9.9 Summary .......................................................................................................... 62
2.10 Conclusions ............................................................................................................ 62

3 User Input .................................................................................................................... 65
  3.1 Introduction ............................................................................................................ 65
  3.2 Influences on User Input ....................................................................................... 66
    3.2.1 A Review of Other Systems .............................................................................. 66
    3.2.2 Bibliographic Databases .................................................................................. 68
    3.2.3 Email names ................................................................................................... 69
    3.2.4 Free-form or Stringent Name Formats ............................................................ 70
  3.3 The Query Data ..................................................................................................... 70
  3.4 Country names ..................................................................................................... 72
    3.4.1 The Query Formats .......................................................................................... 74
    3.4.2 Other Features of the Queries .......................................................................... 76
  3.5 Organisation Names ............................................................................................. 78
    3.5.1 Other Features of Organisation Name Queries ............................................... 81
  3.6 Department names ............................................................................................... 86
  3.7 Person names ....................................................................................................... 91
  3.8 UPN input .......................................................................................................... 100
    3.8.1 PARADISE query data .................................................................................... 100
    3.8.2 ULCC query data ............................................................................................. 104
CONTENTS

6.2 Approximate Matching Problems and Techniques ............................................................. 204
  6.2.1 Types of Misspelling Error ........................................................................................ 204
  6.2.2 Other Issues in Misspelled Names .............................................................................. 207
  6.2.3 Correction techniques .................................................................................................. 208

6.3 Some Approximate Matching Experiments ........................................................................... 216
  6.3.1 The Experimental Methodology ................................................................................. 216
  6.3.2 Damerau ............................................................................................................ 218
  6.3.3 Soundex ............................................................................................................................. 219
  6.3.4 Metaphone ...................................................................................................................... 222
  6.3.5 Bickel ................................................................................................................................ 223
  6.3.6 Approximate Matching Algorithms and Damerau Misspellings ............................. 227
  6.3.7 The Speed of the Algorithms ..................................................................................... 229

6.4 Conclusions on Approximate Matching ................................................................................. 232

7 Conclusions .......................................................................................................................................... 235
  7.1 Outline .......................................................................................................................................... 235
  7.2 The Contribution Made by This Thesis .............................................................................. 235
    7.2.1 Summary of the Main Findings .................................................................................. 236
  7.3 Comparing Three Algorithms .................................................................................................. 237
    7.3.1 Query Resolution Within the DUA ........................................................................... 237
    7.3.2 Read then Search ............................................................................................................ 238
    7.3.3 UFN ................................................................................................................................... 238
    7.3.4 DE ....................................................................................................................................... 239
  7.4 Facilities That Would Help DUA Developers .................................................................... 240
    7.4.1 Features in the 1993 Standard .................................................................................. 240
    7.4.2 Other Changes ............................................................................................................... 241
  7.5 A Powerful Strategy ................................................................................................................... 241
  7.6 Possible Future Work ................................................................................................................ 242
    7.6.1 Consolidating the Findings of this Thesis ............................................................... 243
    7.6.2 Work That Awaits a Mature Heterogeneous Directory ..................................... 244
    7.6.3 Interesting New Work Areas ........................................................................................ 245
  7.7 The Relevance of the Contribution ......................................................................................... 246

A Attributes and Search Filters ......................................................................................................... 247
  A.1 Attribute Types .......................................................................................................................... 247
  A.2 Filter Items ................................................................................................................................... 247
  A.3 Boolean Expressions ................................................................................................................. 248
  A.4 Sequences of Filters ................................................................................................................... 248

B Country and Organisation Name Mapping Tables ......................................................................... 249
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.1 Country Name Queries</td>
<td>249</td>
</tr>
<tr>
<td>B.2 Organisation Name Queries</td>
<td>250</td>
</tr>
<tr>
<td>C Details of Matching Experiments</td>
<td>251</td>
</tr>
<tr>
<td>C.1 Hard-to-match Multiple Token Organisation Queries</td>
<td>251</td>
</tr>
<tr>
<td>C.2 Difficult Organisation Queries</td>
<td>252</td>
</tr>
<tr>
<td>C.3 Difficult Department Queries</td>
<td>256</td>
</tr>
<tr>
<td>C.4 Techniques for Matching Multiple Token Person Queries</td>
<td>260</td>
</tr>
<tr>
<td>C.5 Database Size and Entries Matched</td>
<td>262</td>
</tr>
<tr>
<td>D Approximate matching in Quipu</td>
<td>265</td>
</tr>
<tr>
<td>E Metaphone Codes in Full</td>
<td>267</td>
</tr>
<tr>
<td>F Verifying Approximate Matches</td>
<td>269</td>
</tr>
<tr>
<td>F.1 Message to Referees</td>
<td>269</td>
</tr>
<tr>
<td>F.2 The Referee-Determined Approximate Matches</td>
<td>270</td>
</tr>
<tr>
<td>G Response times</td>
<td>271</td>
</tr>
<tr>
<td>G.1 Introduction</td>
<td>271</td>
</tr>
<tr>
<td>G.2 Experimental Methodology</td>
<td>272</td>
</tr>
<tr>
<td>G.2.1 The Use of the dish DUA</td>
<td>272</td>
</tr>
<tr>
<td>G.2.2 Gathering the Response Time Data</td>
<td>273</td>
</tr>
<tr>
<td>G.2.3 Processing the Results Data</td>
<td>274</td>
</tr>
<tr>
<td>G.2.4 DUAs and DSA in the Tests</td>
<td>274</td>
</tr>
<tr>
<td>G.2.5 Distributed Operations</td>
<td>275</td>
</tr>
<tr>
<td>G.2.6 The Operations Tested</td>
<td>276</td>
</tr>
<tr>
<td>G.3 Starting the DUA and Connecting to the Directory</td>
<td>277</td>
</tr>
<tr>
<td>G.3.1 Introduction</td>
<td>277</td>
</tr>
<tr>
<td>G.3.2 Connection Tests</td>
<td>278</td>
</tr>
<tr>
<td>G.3.3 Analysis of the Speed of DUA Start-up</td>
<td>279</td>
</tr>
<tr>
<td>G.4 Some Basic Search Filters and Read Operations</td>
<td>281</td>
</tr>
<tr>
<td>G.5 Comparing the Speed of the First and Subsequent Operations on a Connection</td>
<td>282</td>
</tr>
<tr>
<td>G.6 Complex Search Filters and Sequences of Searches</td>
<td>283</td>
</tr>
<tr>
<td>G.7 Search Filters Including Object Classes</td>
<td>285</td>
</tr>
<tr>
<td>G.8 The Size of Result Sets</td>
<td>286</td>
</tr>
<tr>
<td>G.9 Referral vs Chaining</td>
<td>287</td>
</tr>
<tr>
<td>G.10 Results from Use of DE Service at ULCC</td>
<td>287</td>
</tr>
<tr>
<td>G.11 DAP vs LDAP experiment</td>
<td>289</td>
</tr>
<tr>
<td>G.12 Conclusions</td>
<td>291</td>
</tr>
</tbody>
</table>
List of Figures

2.1 Accessing the directory ................................................................. 36
2.2 A typical person entry ................................................................. 37
2.3 A fragment of the DIT ................................................................. 38

3.1 UFN examples shown to users ...................................................... 102

C.1 The relationship between database size and the proportion of unique surnames .. 263
C.2 The relationship between database size and the proportion of queries returning less
    than a screenful (24) of results ................................................. 264

G.1 DUA and DSA test configurations .............................................. 275
G.2 The components used in the DAP versus LDAP test ..................... 289
### List of Tables

2.1 Attribute types commonly used in search filters ....................................................... 47
2.2 Examples of form mis-matches ..................................................................................... 49
3.1 Electronic mail name formats ...................................................................................... 70
3.2 Basic characteristics of four sets of sample query data ................................................ 73
3.3 Summary of all country name input analysed by form ................................................. 74
3.4 Country name input excluding default values .............................................................. 75
3.5 Non-US and Non-UK country name input ................................................................. 75
3.6 Country names and dots in input .................................................................................. 76
3.7 Country name input and spelling mistakes ................................................................. 77
3.8 Percentage of country name queries that are RDNs with various sizes of name-to-
    RDN mapping tables ........................................................................................................ 78
3.9 Basic characteristics of four sets of UK organisation query data ................................. 79
3.10 Organisation name input analysed by form ............................................................... 80
3.11 Organisation name input excluding default values .................................................... 80
3.12 Organisation names including dots ............................................................................. 81
3.13 Organisation name spelling mistakes ......................................................................... 82
3.14 Organisation name spelling mistakes and length of input ........................................ 82
3.15 Character position of first incorrect character in misspelled organisation name input 83
3.16 Form of university queries with “university” or “uni” in the name ............................ 83
3.17 Queries containing stop list words and punctuation ................................................... 84
3.18 Percentage of single token queries organisation name queries that are DNS names . 85
3.19 Percentage of organisation queries that are RDNs ................................................... 85
3.20 Percentage of queries that are RDNs with various sizes of organisation name-to-
    RDN mapping tables ........................................................................................................ 86
3.21 Basic characteristics of four sets of UK organisation query data ................................. 86
3.22 Department name input analysed by form .................................................................. 87
3.23 Department name input for UCL users looking for entries within UCL ................. 88
3.24 Department name spelling mistakes ......................................................................... 88
3.25 Character position of first incorrect character in misspelled department name input 89
3.26 Department name queries containing any of the words: Department, School, or
    Institute ............................................................................................................................. 90
3.27 Queries containing stop list words and punctuation ................................................... 90
3.28 Proportion of queries where department name is same as RDN ............................... 91
3.29 Percentage of queries that are RDNs with various sizes of department name-to-
    RDN mapping tables ........................................................................................................ 91
3.30 Basic characteristics of six sets of person name query data ...................................... 92
LIST OF TABLES

3.31 Summary of person name query formats ................................................................. 93
3.32 Detailed summary of person name query formats .................................................... 95
3.33 Character position of first incorrect character in misspelled person name input ........ 97
3.34 Initials with and without dots .................................................................................. 98
3.35 Queries containing stop list words and punctuation .................................................. 98
3.36 The use of full and familiar name forms in person name queries ......................... 99
3.37 Results from first set of UFN data ........................................................................... 101
3.38 Results from second set of UFN data ..................................................................... 103
3.39 Contrasting UFN and simple mode input for country names .................................. 105

4.1 Summary of organisation name data ........................................................................ 117
4.2 Percentage of RDNs consisting of a single token .................................................... 118
4.3 Average length of organisation names ..................................................................... 119
4.4 Entries where an associatedDomain attribute contains additional name forms ....... 120
4.5 Summary of organisational unit name data .............................................................. 121
4.6 No. of organisations with varying percentages of departments with alternative names 122
4.7 Average length of organisational unit names ............................................................ 122
4.8 Different names for computer centres ...................................................................... 123
4.9 Matching common department names ..................................................................... 125
4.10 Summary of person entry data ................................................................................ 127
4.11 Summary of person name query formats ................................................................ 127
4.12 Person entries including alternative common name attributes ............................... 129
4.13 No. of organisations with varying percentages of persons with alternative names ... 130
4.14 Percentage with at least one common name conforming to “A...Smith” format ....... 130
4.15 Person entries including a userid attribute ............................................................ 131
4.16 Use of full and familiar name forms for person entries ......................................... 132

5.1 Country name matching using basic filters ............................................................... 140
5.2 Country name matching using more sophisticated search filters ............................ 143
5.3 The effect of trying to omit country name resolution by using the read and search strategy on the number of operations required to resolve a country name ......................... 144
5.4 Reasons for country name matching failures ............................................................ 146
5.5 Country name spelling mistakes and approximate matching .................................... 146
5.6 Breakdown of organisation name matching by filter ............................................... 148
5.7 The success of non-exact matching filters at matching short organisation name queries not matched using more precise matching methods .............................................. 150
5.8 More sophisticated organisation name matching filters ......................................... 153
5.9 The impact of read and search and mapping tables on the number of filters used matching organisation names ................................................................. 154
LIST OF TABLES

5.10 Reasons for organisation name matching failures .............................................................. 156
5.11 Misspelled organisation names and approximate matching .............................................. 157
5.12 Matching organisation names stripped of "University (of)" ................................................ 157
5.13 A comparison of the original DE algorithm with an algorithm including the most effective enhancements ................................................................................................................ 162
5.14 Breakdown of matching department names by filter ........................................................... 164
5.15 Type of department name input for short queries .............................................................. 165
5.16 Short department name queries not matched using exact matching .................................. 166
5.17 Breakdown of department name matching by filter ............................................................ 168
5.18 Reasons for failing to find the correct department entry ................................................... 169
5.19 Misspelled department names and approximate matching ................................................ 170
5.20 Matching hard-to-match department name queries using truncated input ....................... 172
5.21 A comparison of the original DE department matching algorithm with two algorithms including effective enhancements .......................................................... 174
5.22 Name formats used for person names in UCL database .................................................... 177
5.23 Percentage of person name queries by filter where at least one entry is matched ................ 179
5.24 Average number of results returned per filter type ........................................................... 180
5.25 Percentage of 'single token, no dept' queries producing a lot of matches ............................ 181
5.26 The number of matches achieved by approximate matching if it is used only after exact and substring matching have failed for single token, no department person name queries ................................................................. 181
5.27 Percentage of queries returning results that return the correct result ................................ 181
5.28 Person Name query length and the number of matches .................................................... 184
5.29 Person name matching with various filters ....................................................................... 186
5.30 Person name spelling mistakes and approximate matching ............................................... 187
5.31 Percentage accuracy for approximate and truncated string matching ............................... 188
5.32 Percentage of cnApproz queries returning results for which there is no directory entry ................................................................................................................................. 188
5.33 Average no. of results returned for any-substring and specific forename-surname filter .......................................................... 191
5.34 Percentage of single token queries returning results where correct result is lost due to restricting the size of the result set ........................................................................ 193
5.35 Improved Person Name Matching Filters ........................................................................ 195
5.36 Number of person name queries where department query correctly identifies correct department ................................................................................................................. 196
5.37 The precision of the department name matching techniques .......................................... 197
6.1 Soundex codes ....................................................................................................................... 209
6.2 Example Soundex keys ....................................................................................................... 209
### LIST OF TABLES

6.3 Some of the Metaphone codes ................................................. 212
6.4 Weights assigned to letters by Bickel matching algorithm ........ 215
6.5 Damerau matching: precision and recall .................................. 219
6.6 Soundex matching: recall and precision with various options .... 220
6.7 Metaphone matching: recall and precision with various options .. 222
6.8 Bickel matching: recall and precision with various options ....... 225
6.9 The weights used by Bickel and those calculated for the UCL database ........................................... 226
6.10 Effectiveness of approximate matching algorithms with common typing errors .. 228
6.11 Effectiveness of Bickel algorithm with ranked results ............ 229
6.12 Speed comparison of algorithms with minimal indexing ............ 230
6.13 Speed of Soundex and Metaphone with enhanced indexing ...... 231
6.14 Speed of various configurations of the Bickel algorithm .......... 231
7.1 Improvements to UFN matching of person names .................. 239
7.2 Improvements to DE matching of person names ..................... 240
A.1 Attribute types mentioned in this thesis ............................ 247
A.2 Examples of the various types of matching in filter items .......... 248
B.1 Country names that would be in a ten entry mapping tables for the UCL and UCL-CS query data .............................................. 249
B.2 Organisation names that would be in a ten entry mapping tables for the UCL and UCL-CS query data .............................................. 250
C.1 Techniques for hard-to-match multiple token organisation queries .............................................. 252
C.2 Methods for matching organisation entries with hard-to-find queries .............................................. 255
C.3 Methods for matching department entries with hard-to-find queries .............................................. 259
C.4 A comparison of several multiple token person name filters .......... 261
D.1 Examples of user input and directory name form mis-matches .......... 266
E.1 The Metaphone codes (A-K) as used in the Quipu system .......... 267
E.2 The Metaphone codes (K-Z) as used in the Quipu system .......... 268
G.1 Time taken in seconds to start dish and to do one/ten binds .......... 278
G.2 Analysing the components of DUA start-up and binding .......... 280
G.3 The processing time speed-up achieved by reducing the size of some ISODE configuration files ................................. 280
G.4 Comparing the speed (in seconds) of read operations and searches with basic filters 282
G.5 Comparing the speed of the first and subsequent operations on a connection .......... 283
G.6 Comparing the speed of different search strategies with multiple filter items ...... 284
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>G.7</td>
<td>Comparing the speed of search filters with and without object class filter items</td>
<td>286</td>
</tr>
<tr>
<td>G.8</td>
<td>The response time in seconds varying according to the number of entries returned</td>
<td>286</td>
</tr>
<tr>
<td>G.9</td>
<td>Chaining compared with Referral</td>
<td>287</td>
</tr>
<tr>
<td>G.10</td>
<td>Query resolution times, categorised by the number of search operations required by DE to resolve a department name query</td>
<td>288</td>
</tr>
<tr>
<td>G.11</td>
<td>Query resolution times, categorised by the number of search operations required by DE to resolve a person name query</td>
<td>289</td>
</tr>
<tr>
<td>G.12</td>
<td>The size of the DE binary built with DAP and LDAP libraries</td>
<td>290</td>
</tr>
<tr>
<td>G.13</td>
<td>A comparison of DAP and LDAP queries using DE</td>
<td>290</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 White Pages Directory Services

We do not have to go back many years for the term directory to have meant, for most people, a telephone directory printed on lightweight paper. The main part of the telephone directory was printed on white pages, and allowed users to look up telephone numbers and postal addresses given the name of the person they were looking for. The directory would be updated every one or two years.

In recent years, the growth of communications and the proliferation of services such as facsimile and electronic mail has meant that paper directories are increasingly inadequate. It is impractical to have paper directories for everything: telephone numbers, facsimile numbers, electronic mail addresses, and so on. Paper directories suffer from the problem of high production costs [Hun90] and being out of date as soon as they are printed. Gilbert shows in [Gil90] that paper directories can cost businesses money through increased operator costs. A further argument for moving from paper directories to electronic ones is that a service such as electronic mail is much improved if the service is supported by a computerised directory.

The growth of networks and computer communications provides a potential solution to these problems. In the last fifteen years, an enormous amount of effort has been invested in developing computerised directories for looking up information about people and organisations. The majority of this effort has gone into designing and developing white pages directories. There is no precise definition for the term electronic white pages directory. However, if we meld the definitions in [SP94] and [PA94], the essential characteristics of a white pages service are:

- the user of the service provides the name of the person for whom they require details, and the service returns information such as a telephone number and/or electronic mail address;
- the service may also require the user to enter a locality or organisation name to limit the scope of the search;
- users of the service should not have to get the details of their query exactly right: for example, the service should work even if the form of the name in the user's query is not exactly the same as the form in the directory; the service should also tolerate minor misspellings.

It should be clear that the term white pages is used for such a service because of its similarities with searching for information in the white pages of a telephone directory.

The problem I address in this thesis is how to make the best of the facilities provided by an electronic white pages directory service to match user queries to the required entries in the directory. If users' queries exactly match the name information in the directory, then the matching
problem is trivial. However, users' queries often differ in form and spelling from the names in directory entries. Directory services provide facilities such as substring matching and approximate (fuzzy) matching to help with such cases. The question that remains is how to use these facilities to correctly match as many queries as possible, with as few spurious matches as possible, as efficiently as possible.

The structure of this introductory chapter is as follows. Section 1.2 outlines in more detail the problem of matching user input and directory entries.

Section 1.3 looks at some directory technology issues. It considers briefly whether the white pages problem merits its own technical solutions or whether such services should be provided by the general information retrieval tools such as the World-Wide Web [BLCGP92]. It then looks at some directory service systems that have been influential during the last fifteen years.

Section 1.4 describes why an X.500 directory is a good choice for experiments with name matching, both for technical and pragmatic reasons.

Section 1.5 outlines the research methodology. It describes the gathering of query and directory data, and the experiments used to test various name matching strategies.

Section 1.6 tries to sharpen the focus by noting some additional directory service issues that are not covered in this thesis.

Section 1.7 outlines the contents of the other chapters in this thesis.

Finally, Section 1.8 makes a few comments on presentational style.

1.2 Matching User Queries and Directory Entries

In this thesis, I have split the problem of matching user names to directory entries into three areas of discussion. These are:

- understanding the format of user queries;
- knowing the format of name information in the directory;
- making the best use of the name matching facilities provided by a directory service to find entries in the directory.

We will discuss these issues in turn.

1.2.1 User Queries

A user has to enter one or more name components when trying to find an entry in the directory. A name component is typically one of the following: a person's name, the department where that person works, the organisation where he/she works, the country where he/she works. If a user is looking for a person within their own organisation, a user will generally only have to provide a single name component: the person's name. This name component may comprise one
or more *tokens*. For example, a user may enter "smith", "j smith", "john smith", "smith, j" or one of many other variant forms. If the person being sought works in a different organisation or a different country, the user will have to supply other name components as well.

A query such as:

\[ \text{p barker, cs, ucl, uk} \]

appears to be a reasonable query for the author's directory entry. However, is this query representative of what users really enter? For example, for person name input, how often do users enter an initial and surname, or do users prefer other forms such as surname only, a login name, forenames only, or forenames and surname? For organisation names, do users tend to enter full names, abbreviated names or sets of initials? Do users tend to enter or omit department names?

Aside from the form of name components, how often do users misspell their input and need some form of approximate matching support to find the required entry?

It is essential that we understand the forms of users' queries to have a realistic chance of optimising a name matching strategy. We examine user input in detail in Chapter 3.

### 1.2.2 The Format of Name Information in the Directory

We also need to know what name formats are used in a directory. In a centralised directory service, this is a simple issue as the service administrators can exercise control over the format of the data. However, for reasons that we will touch upon in Section 1.3 most current directory services are distributed directories, with each organisation administering the data for their own organisation. This distribution of administrative authority makes it hard to impose any rigidity of name format. We will see in Chapter 4 that formats of data in the directory vary as much as users' query formats.

Another important factor is whether directory administrators include alternative name forms to facilitate name matching. It should be evident that if the directory includes many variant name forms, then the matching problem is simplified, as a user's input is more likely to match one of several name formats than any one single format. While it is obviously desirable that directory administrators should include variant formats, do they do this in practice?

A further factor is the provision of other attributes that may be used for matching. A directory entry for a person can include many attributes, such as telephone and facsimile numbers, electronic and paper mail addresses, room numbers, areas of technical expertise, and so on. In principle, almost any attribute can be used in trying to match user input to a directory entry. For example, a person name look-up might be based on a person's surname, their full name or their computer login name. An organisation look-up might be on the full organisation name or it might be based on the organisation's DNS name [Moc87a] [Moc87b]. We may have to develop name matching strategies based on several different attributes.
1.2.3 Name Matching Algorithms

Once we have a good understanding of user queries and directory naming, we are then ready to try to devise algorithms to match the input to the required entry.

While the matching facilities provided by directory services vary in detail, most services provide some form of exact matching, substring matching and approximate matching. It is easy to see the potential usefulness of these types of matching.

- **Exact** matching, by definition, only matches entries with exactly the same name information, and is thus a precise type of matching.
- **Substring** matching is useful for matching abbreviated queries, such as “Cam” or “Cambridge” for “Cambridge University”, or “Paul” for “Paul Barker”.
- **Approximate** matching might be useful for matching misspelled input. It could also potentially be used for mismatches of form: for example, it might match input of “University of Cambridge” against an entry name of “Cambridge University”. However, the looseness of approximate matching means that it tends to return more entries than other types of matching.

The solution to the matching problem is knowing how to make best use of these facilities. We have identified uses for several types of matching, but how do we best use the facilities in combination? A key factor is how closely user input matches the name values in the directory: if, for example, there is poor correspondence, the role for exact matching will be small.

Another factor is providing what the user requires from the service. User A might require a service that finds the required entry at least 80% of the time, but does so using as few querying operations as possible. Alternatively, user B might require a service that finds the required entry at least 98% of the time, but where restricting the number of querying operations on the directory is not a priority. In contrast the most important issue for user C might be to restrict the size of result sets.

It is thus unlikely that we will be able to identify a name matching strategy that is unequivocally the best – different requirements require different trade-offs between finding as many correct results as possible, finding as few incorrect results as possible, and look-up efficiency in terms of speed and economy of use of directory operations.

1.3 White Pages Directory Services and Protocols

In this section, we first consider whether directory services require special tools or whether they are best provided by general information retrieval tools. We then look at research and developments in directory services over the last fifteen years. Finally, we note that although there is a large collection of literature on directory services, the issue of name matching has received little attention. The work in this thesis is intended as a contribution in this area.
1.3. SPECIFIC OR GENERALISED TOOLS FOR WHITE PAGES

In recent years, we have seen developments in two types of tools for doing white pages look-ups. First, a number of tools have been developed specifically to tackle the white pages problem. These include Whois [HW82], CCSO [WG93] [HDP96] and X.500 [ISO88b].

As well as these tools which specifically provide white pages services, some general purpose information retrieval tools have been developed. One use of these tools has been to access white pages information. Prominent examples of these tools are Gopher [MaC92] and the World-Wide Web [BLCGP92].

One survey, jointly authored by the creators of some of the most influential and widely used information retrieval tools, compared nine “resource discovery approaches” [SEKN92]. Some of these tools were information type-specific, for example Archie [DE92] for finding files, while others were general purpose such as the World-Wide Web. A useful contribution made in the paper was identifying both the common ground and the differences between the various tools and the information retrieval domains. The authors predicted that experience with these tools would lead to cross-fertilisation of ideas, and that systems would tend to converge over time. This assessment begs the question of whether we will need specific information retrieval tools to provide white pages directory services. However, the authors of the paper note that there are significant differences between the various information retrieval domains, limiting the effectiveness of a totally generalised approach.

The evidence of recent years suggests that the authors’ assessment was about right. We have seen the World-Wide Web become pervasive, and made an effective information resource by powerful indexing tools such as AltaVista [AV96]. However, we have also seen the development of information-specific tools such as WhoWhere [WW96], which serves electronic mail addresses. The author of this thesis believes that general purpose and information-specific retrieval tools are likely to persist for some time to come.

In the next section, we will review developments of electronic directory services, and identify how this thesis builds on our knowledge of white pages services.

1.3.2 IMPORTANT RESEARCH AND IMPLEMENTATIONS

A number of electronic directory services and protocols have been specified over the last fifteen years that tackle the problem of finding white pages information such as telephone numbers and electronic mail addresses for individuals and organisations.

The approach to providing a white pages directory service on communications networks has changed over the years. Older systems tended to be centralised databases. The Whois service, first specified in 1982 [HW82], provided a directory based on a single server for all ARPANET and MILNET users. The CSNET name server [SLN82] also used a centralised database: the focus of CSNET was on simplicity and getting a system in place to facilitate communication between researchers. A factor common to these two systems is that they were both designed as
practical solutions to directory service problems – the scale of the network at that time meant that centralised directories were feasible.

However, at the same time as these systems were being deployed, other researchers were working on distributed directory systems. Two influential systems were produced by groups of researchers at Xerox: Oppen and Dalal’s Clearinghouse [OD81], and Birrell et al’s Grapevine [BLNS82]. Several reasons have been put forward why a global directory should be a distributed, rather than a centralised, directory. A distributed system is not restricted by the computing power of, or network connectivity to, a single server. A distributed system typically means having servers nearer to users, and thus more efficient access. Multiple servers may add some resilience to failure in a system, and they help to spread the querying load. Security is best served by local administrators physically controlling their own sensitive data, rather than trusting the security mechanisms of a directory run by another organisation. Distributed data management divides the task amongst numerous organisational administrators, and gives organisations autonomy over how they structure their directory databases and how they name entries within them.

The majority of directory service developments in recent years have been for distributed directory services. Some notable examples include ECMA’s TR32 [ECM87] and Profile [Pet88]. These were followed in 1988 by X.500 [ISO88b], an ISO/CCITT international standard for a distributed directory, the use of which is the subject of much of this thesis. X.500 has spawned imitators that share many of X.500’s characteristics, but which use different protocols: these include Novell’s Netware Directory Service (NDS), Microsoft Exchange [Mic95], DIXIE[HSB91] and the Internet-specified Lightweight Directory Access Protocol (LDAP) [YHK95]. The old Whois service has recently been revamped and updated as Whois++ [DSFW95] [WFS96], adding distributed directory functionality to what was a centralised directory. An alternative enhancement of the Whois service has been proposed as Referral Whois (RWhois) [WK94]: RWhois servers can refer clients to the appropriate server. The Nomenclator project [Ord95] is producing a distributed directory by combining the 300+ centralised CCSO servers [HDP96] into a distributed system.

An alternative approach to building a distributed directory has been proposed by other researchers. This approach is to build a unified service on top of various underlying directory services. Droms proposed a Directory Access Service [Dro90], and described how this would work over underlying services such as Finger [Zim91], Whois [FHS85] and Profile [Pet88]. Schwartz and Tsirigotis designed Netfind [ST91] along similar lines, extending the coverage to include CCSO and X.500 servers.

It should be apparent to the reader that there is a substantial literature on directory service systems. The papers describe a wide range of issues including naming structures, the information model, schemas, the query and update operations, the protocol that implements those operations, the distributed systems architecture, replication facilities, security features including authentication and access control, system administration, and many other issues. However, relatively little has been written on how to use these services’ name matching facilities.
1.4. THE USE OF X.500 AS A BASIS FOR STUDY AND EXPERIMENTATION

1.3.3 Name matching

While there is an extensive literature on directory services in general, we noted that little has been written on how to use these services' name matching facilities to provide the best service to the user. Some approaches have been proposed – for example, by Afifi and Huitema [AH92], Kille [Kil95b], and Findlay et al [FMN90a] – but these papers are mostly descriptive and do not analyse in detail the validity or limitations of the suggested approaches. The core work of this thesis is an attempt to fill this gap. We examine how to make effective use of a directory service's name matching facilities to find the required directory entries given user input that may not always correspond closely with the naming information in the directory.

1.4 The Use of X.500 as a Basis for Study and Experimentation

The core work in this thesis is empirical. We study how to build a name matching strategy by using data from a substantial deployed white pages directory service. We are interested in two types of data.

- We want to know about the uniformity or heterogeneity of data formats used in a distributed directory administered by hundreds of data providers.
- We also want information on how users formulate their queries.

We then use this query and directory data in a set of experiments to refine a name matching strategy.

I have used an X.500-based service to provide data for analysis and experimentation. In this section, I will first argue that X.500 is a good choice for experimentation. Second, I will argue that the majority of the analysis applies to directory services in general.

An important practical factor in choosing X.500 for experiments is that a substantial X.500 directory has been deployed. In May 1995, DANTE (the organisation responsible for coordinating the international X.500 directory) estimated that there were 750 publicly accessible X.500 servers in the NameFLOW-PARADISE directory [Ber95] with details of over 4500 organisations. A report from the previous year estimated that the directory contained well over one million entries [Goo94]. Furthermore, my own involvement with the PARADISE project [Goo91] meant that I had access to a large volume of query data.

Pragmatic considerations aside, there are several other good reasons to believe that X.500 is a good choice of directory service for an analysis of name matching strategies. First, X.500 is an international standard, which confers some advantages. Its status as a standard means that it is likely to be adopted by large users such as governments and the military, and strongly

1The NameFLOW-PARADISE directory is a publicly accessible X.500 directory service primarily for academic and research organisations.
CHAPTER 1. INTRODUCTION

considered by other groups of users. Of course, this does not mean that it will be universally adopted. At the time of writing, it is too early to say whether X.500 will eventually become the directory service. On a positive note, vendors are taking X.500 seriously and there are many implementations available[GS94]. Against this, adoption of X.500 has been relatively slow on the Internet, and growth in the last few years [Goo94] has been slower than the growth of the Internet[ISO96].

However, the momentum in the Internet lost by X.500 has largely been gained by LDAP [YHK95], which is derived from X.500. Over 40 companies, including most of the major software vendors, announced in April 1996 that they were using or planning to use LDAP, with industry analysts saying that as LDAP has so much support, it was unlikely that anyone will be able to introduce a competing standard [IEE96]. One reason for LDAP's appeal is that it has adopted so much of X.500, including naming and the information model. Because of the similarity, gatewaying between LDAP and X.500 systems is straightforward. Such gateways have existed for several years [HS96]: using LDAP means gaining access to the X.500 information base as well.

Of particular relevance to this thesis, LDAP has adopted X.500's search operation almost as is, which means that the detailed work on name matching algorithms described in chapter 5 is directly applicable to an LDAP directory system.

I have used the public domain Quipu implementation [Kil88] of X.500 for my experimental work. However, since the name matching capabilities of X.500 are specified in the standard, the choice of implementation is irrelevant from the standpoint of name matching. (Of course, some implementations may be better than others in terms of performance and management facilities.)

There is one exception to this uniformity of name matching facilities among X.500 implementations. The X.500 standard does not define an approximate matching algorithm: the choice of algorithm is left to the implementor. In practice, the majority of X.500 systems (including Quipu), and non-X.500 systems too, use the Soundex algorithm [Knu73] for approximate matching. We examine Soundex and some alternatives in Chapter 6.

I also believe that the majority of the analysis can be applied to directory services based on other protocols. The core facilities of current directory services such as X.500, LDAP, Whois++ and CCSO are very similar. For example, they all offer search operations with some form of exact, substring and approximate matching. (I compare the name matching facilities provided by these four systems in Chapter 2.)

There are, of course, some differences between the various services. X.500 provides read and list operations which are not offered by the other three services mentioned. However, the majority of the analysis should apply almost unchanged to most directory services.

Finally we have to consider whether the X.500 query and directory data analysed in this thesis is typical for directory services in general. Once again, I believe that the data is free from X.500 bias. The query data analysed is taken from usage of the DE user interface [Bar91], written by the author as part of the PARADISE project. DE deliberately provides a very general abstraction of
1.5. THE RESEARCH METHODOLOGY

a directory service. An identical abstraction could be used for querying other types of directory such as CCSO or Whois servers. Therefore, I believe the query data is representative of queries for any type of white pages directory service. Second, there is nothing intrinsically X.500-ish about the data in the directory. In fact, the format of the data varies widely from server to server. I believe that the over-riding influence on the precise format of an individual site's data is the format of the data in that site's primary data source: e.g., if the site's personnel data includes forenames rather than initials, then that site's directory names include forenames.

In summary, the NameFLOW-Paradise X.500 system provides a test bed of considerable size and heterogeneity. The large number of participating organisations ensures a variety of data. The service is mature and has a substantial user base: several thousand queries have been analysed from four different user communities. Furthermore, the problem of name matching is largely technology-independent, with several widely used technologies offering broadly similar functionality.

We can now define the thesis as follows:

Name matching techniques in a distributed white pages directory can be improved by analysing statistics of usage of live directory services. The thesis is demonstrated by using empirical methods in a series of experiments.

1.5 The Research Methodology

The research in this thesis is based on taking a large sample of user queries from a live directory service. These queries are then replayed in a test environment against a copy of the data held in that directory.

We use the queries in a large number of experiments. First, we demonstrate the effectiveness of the X.500's basic querying facilities at matching raw user input with the target directory entries. We then examine some more sophisticated name matching strategies, based on using combinations of the basic name matching facilities. We also experiment with some simple transformations of user input as we try to match some of the more difficult queries.

1.5.1 User Input

The analysis of user input is based on several thousand queries of the NameFLOW-Paradise directory service. These queries were made using the author's DE user interface [Bar91], a widely deployed user interface developed during the PARADISE project. The query data was gathered from four different instances of DE: two run from within UCL and two run by the University of London Computer Centre (ULCC). All the queries were real service usage, made by users who (I presume) were unaware that their queries would subsequently be analysed.

We examine the main forms of user input. We look for potentially hard-to-match queries, which, for example, have unusual formats or contain punctuation characters.
I have published some preliminary findings on this topic in [Bar95a]. However, that work was based on analysing a single source of queries. The work in this thesis is a far more extensive treatment of the topic. This work should be highly relevant to other directory services.

1.5.2 Directory Data

The analysis of directory names is based on data in the NameFLOW-Paradise directory. The scale of the directory means that a full analysis is impossible. Instead, most of the study is based on sampling data in five countries which have non-trivial amounts of directory entries. These samples illustrate well the diversity of naming in the directory.

As with user input, we examine the main trends in how entries are named in the directory. We examine how generous data administrators have been in providing additional values in the directory for matching: for example, including both “UCL” and “University College London”. We also look for name formats that are likely to make matching more difficult. This analysis of directory data should largely hold for other directory services.

1.5.3 Name Matching Experiments

The core work of the thesis is a set of experiments to test various name matching algorithms. This work is based on matching subsets of the query data and directory data described above. The various sets of directory data were loaded into a single test DSA running on the author’s workstation.

I conducted four sets of matching experiments, one for each of the name components: country names, organisation names, department names, and person names. An analysis of the query and directory data shows that the techniques required to match country names, for example, are quite different to those required to match person names.

Each set of matching experiments initially focuses on the effectiveness of the basic matching facilities, such as exact and substring matching, at finding the required entries. We then examine some more sophisticated approaches, based on some which have been implemented or described in the literature. These are:

- the name matching aspects of the UFN algorithm, defined in RFC 1781 [Kil95b];
- the approach used by the DE user interface [Bar91], which uses a sequence of searches with increasingly loose matching, the sequence continuing until a match is found or all filters have been tried;
- the approach described by Afifi and Huitema in [AH92], which tries to be as economical as possible in its use of directory operations.

We examine several aspects of the effectiveness of these approaches: how good they are at finding the correct result; how often they find the correct result and no other results; the average
1.5. THE RESEARCH METHODOLOGY

size of the result set; the average number of directory operations required to resolve the queries. For each type of input, we find that there is a residue of hard-to-match queries that are not matched using any of these techniques. We explore enhancements to the algorithms that find the correct entries for the majority of these harder queries.

1.5.4 Approximate Matching

One thing that emerged from the research on user input and data formats was a clear need for some use of approximate matching. However, anecdotal evidence from both users and directory experts suggested that users often found the results of Quipu's Soundex approximate matching to be unpredictable, reducing their confidence in the directory service. One group of user interface developers even stopped using approximate matching as they felt the results it gave were usually spurious [Mah95]. Another group of implementors replaced Soundex with Metaphone [Phi90], which they claim gives more satisfactory results.

We conduct further matching experiments to compare the Soundex algorithm with Metaphone and two other alternatives. The experiments with approximate matching algorithms are based on queries of the UCL database. In particular a set of misspelled person name queries — genuine misspellings by directory users — is used to assess the matching capabilities of the various algorithms.

1.5.5 Response Times

The speed with which a service responds to queries is a very important factor. A name matching strategy that out-performs other strategies in terms of the proportion of queries successfully answered and its accuracy of matching may be of little practical use if it takes ten times as long to deliver its results. It is clearly useful to have some feel for whether some name matching strategies are faster or slower than others.

We conduct some experiments using DSAs in the NameFLOW-Paradise directory to compare some name matching strategies. We gather the timing information in two ways. First, we use a minimalist directory user interface\(^2\) in a special test harness. The tests are constructed so that the response times are representative of what an ordinary user of the directory would normally achieve. Secondly, we make detailed timings of directory operations using a specially modified version of the DE user interface.

These response time measurements are of necessity made in a directory environment which is dominated by a single implementation of X.500, the afore-mentioned Quipu. As it is unrealistic to assume that other implementations will offer the same performance characteristics, we cannot use the findings to generalise about other implementations. Furthermore, the experiments are also largely made from workstations within my own department, which may not be representative of querying environments in general.

\(^2\)The *dish* user interface which comes as part of the Quipu distribution.
CHAPTER 1. INTRODUCTION

Because of these limitations, the material on response times is not presented in the main text, but as a substantial appendix. Despite the reservations I have stated about the generality of this work, the study produced some interesting findings, and illustrates some pitfalls that those administering directory services would do well to avoid.

1.6 Some Issues that are Not Discussed

Although I have set out carefully the issues that are covered in this thesis, I hope to further tighten the focus by noting some issues that are not discussed.

The thesis concentrates on name matching, but this is just one part of the problem of finding the required entry. A significant problem in a distributed directory is knowing which part of the name-space to search, or being able to identify the appropriate server to search. This problem can arise, for example, when a directory user wants to find the entry for Bloggs, whom he/she met at a conference, but the user cannot remember where Bloggs works. This is a significant problem and is the focus of much current research. I have described one approach to the problem in [Bar95c], and Chadwick sets out an alternative in [Cha96]. Attempts are being made in the Internet community to design a standardised solution [Wei96].

The thesis looks at the problem of name matching for four categories of name information: countries, organisations, departments and people. This reflects the predominant naming hierarchy in the NameFLOW-Paradise directory. We should bear in mind that a full-scale directory might further sub-divide the name-space by introducing locality entries for counties, states or even cities.

The thesis makes no attempt to assess the suitability of general-purpose tools such as the World-Wide Web for directory services. While this is an interesting topic, it is someone else's thesis!

1.7 Structure of This Thesis

This section outlines the structure of the rest of this thesis. As the thesis divides into chapter-based topics, I have adopted an approach advocated by Aaron Sloman [Slo92] and split the literature review across the chapters. This way, the literature on, for example, approximate matching is considered immediately before my own work on the problem.

Chapter 2 contains a general introduction to the topic of matching user input to directory names. The chapter introduces enough of X.500 to make the subsequent discussion intelligible to X.500 novices. It then examines the querying facilities in detail, and shows how they can be used to solve a number of name matching problems. The chapter also considers some ways in which a user interface can build upon the services offered by X.500. It examines the impact, if any, of the distributed directory environment on querying algorithms. Finally, it compares X.500 with three other directory service systems and considers the applicability of this work, based on an X.500 system, to non-X.500 systems.
1.8. A FEW POINTS ON STYLE

Chapter 3 discusses the format of user queries. In particular it concentrates on which formats are popular, the correspondence between user input and the names of directory entries, any formats that cause matching problems, and the prevalence of spelling mistakes.

Chapter 4 analyses the format of directory entry names and any alternative name attributes that directory entries include. It focuses on several aspects: whether directory administrators have tended to use long names as recommended in the naming guidelines; how often administrators have included alternative name values (as this simplifies the task of matching); any problems with naming that tend to make matching more difficult.

Chapter 5 is the core of the thesis. The chapter describes some experiments using various X.500 querying facilities which show the characteristics of the various types of matching. It discusses issues such as: the ability of a search strategy to find the correct result but not to find lots of spurious results; the ability of a querying strategy to find results when the user's query does not closely resemble the name values in the directory; how good the approximate matching algorithm is at finding entries when the input is misspelled. It describes some techniques for matching entries that require the user interface to transform the input before trying to query the directory.

The work in Chapter 6 grew out of the experiments undertaken as part of the work for Chapter 5. This showed that while the Soundex-based approximate matching algorithm (used in Quipu, and in many other directory service implementations) is quite effective at matching misspelled input, it tends to return very large result sets. The chapter compares Soundex with three alternative approximate matching algorithms, and assesses their ability to match misspelled input.

Chapter 7 draws together the various strands of the thesis. It summarises the main findings and shows how the work has led to considerable improvements to both the DE and UFN algorithms. It also looks to the future, both in terms of what additional features it would be nice to have in a directory service, and how the work in this thesis could be developed further.

There are also a number of appendices. A few are particularly worthy of note. Appendix C includes details of some of the matching experiments: this material is likely to be of interest to user interface developers who are trying to fine-tune their name matching algorithms. Appendix D describes the details of the Quipu's approximate matching, including its use of the Soundex algorithm. Appendix G describes the experiments on response times. Some of the material in this appendix should help directory service administrators avoid some of the pitfalls that result in poor response times.

1.8 A Few Points on Style

There is potential in the text of this thesis for confusion between user queries and names in the directory. To avoid this problem, I use distinct formats. Queries are presented in little-endian
format: i.e., person name first, country name last. Directory names are presented in big-endian format, and are enclosed in braces – { } – as they are in the standard.

The directory matches names case-independently. I have mostly used lower case for user input and capitalised words in directory names where capitalisation looks natural. I occasionally break these rules if I think that the rules confuse the issue being discussed.

Throughout the thesis I use the terms "organisational unit" and "department" almost interchangeably. Organisational unit is an X.500 term: I tend to use department to avoid the clumsiness of the longer term.
Chapter 2

Matching Algorithm Issues

2.1 Introduction

In this chapter we examine the querying tools provided by X.500, and examine how they can best be used to support querying of the directory.

I assume that the reader has little or no knowledge of X.500 and I start by introducing those aspects of X.500 which the reader needs to understand in order to follow the discussions and arguments in this thesis. The basic description is of the 1988 version of X.500, although facilities introduced in the 1993 standard [ISO93] are noted where applicable. The primary reason for focusing on the old standard is that the experimental work described in later chapters was done in 1994 and 1995 when few, if any, X.500(1993) systems were deployed. In mid 1996, widespread deployment of X.500(1993) systems still seems unlikely in the short term.

In Section 2.3 I introduce two querying abstractions that can be built using X.500 services. We will see that while the browsing abstraction is often popular, particularly with novice users, the best approach is for users to query the directory by entering name input. X.500 offers a range of facilities for matching the name input against entries in the directory. In section 2.4 we consider the role of X.500's read operation. However, we see that while read operations may have a role to play in a querying algorithm, we will often need to use search operations to resolve queries. In Sections 2.5 and 2.6, we illustrate the use of the search facilities by considering how the directory could match a number of example queries.

In Section 2.7 we see that a directory user interface can add extra facilities to those provided by X.500, by doing things such as pre-processing users' queries into formats most likely to match directory entries.

In Section 2.8 we consider the directory as a heterogeneous environment and look at some of issues that could affect our choice of querying strategy. For example, strategy A might get the same results twice as fast as strategy B. Some strategies that work well in high bandwidth environments may be unsuitable for low bandwidth access.

Although the focus of the work in this thesis is on name matching using X.500 systems, we see in Section 2.9 that several popular directory systems have many similarities, suggesting that the findings of this thesis should be generally applicable to other directory systems.

In Section 2.10 we conclude this chapter and look ahead to the work in other chapters.

2.2 Outline of X.500

The purpose of this section is to introduce briefly the aspects of X.500 with which the reader will need to be familiar so that he/she can follow the discussion in the remainder of this thesis.
Figure 2.1 illustrates how a user accesses information in the directory. He/she uses a piece of software called a Directory User Agent (DUA). A DUA includes a set of functions to retrieve information from (and possibly to modify information in) the directory: the DUA communicates these requests to the directory in a standardised protocol called Directory Access Protocol (DAP). A DUA may also include functions to help a user form his/her queries, to present results in a user-friendly format, to interpret diagnostic messages, and so on. DUAs vary considerably in the facilities they provide to help the user access the directory. Some specialist DUAs may allow users to use all the features provided by X.500; other DUAs may use a small subset of X.500's operations, but add a lot of functionality by, for example, pre-processing queries or post-processing results.

Although X.500 is designed as a distributed directory, a DUA can in general treat the directory as a black box, and access all information via a single access point, knowing nothing about the physical distribution of information amongst directory servers.

The information in the directory is held in entries, where each entry describes someone or something in the world. Each entry consists of a set of attributes, where an attribute consists of an attribute type and one or more attribute values. The directory is object-oriented and each entry includes an objectClass attribute. The value(s) of this attribute define what sort of object the entry is describing, and which attributes that entry can, or even must, have. For example, an entry describing a person has an object class with a value of person (and possibly some other object classes as well). A person entry should be named by a value of the commonName attribute, and the entry must also include a surname attribute. In addition, a person entry can include many other attributes, including computer login userid, telephone and facsimile numbers, postal and email addresses, and so on. The X.500 standard defines a core set of object classes and attributes. This set is enhanced by definitions in RFC 1274 [BK91]. Communities, or even individuals, can define further object classes and attributes if required. Figure 2.2 illustrates a typical person entry. The entry's format is that used in the Quipu system's database files [KR91].

Several points about the entry are worth emphasising. The commonName (or cn) attribute in this case is multi-valued; the ampersand character is used as the separator between values.
Directory administrators can include several name forms such as: a version including forenames; a version including initials; or nicknames. A person entry must also include a "surname" (or sn) attribute, and may also include a computer login uid attribute (if the schema described in RFC1274 is supported). These three attribute types are important in the context of querying algorithms as we will see later that directory user agents commonly compare user queries with the values of these attributes to find the required entry. The objectClass attribute is also important when querying the directory as it allows a user agent to indicate that it is only interested in certain types of entry, such as person or organizationalUnit. For example, a query might have the semantics: find the entry for the person with the login name of "paulb".

Another crucial point stems from the fact that the directory is a distributed database, managed by many people. Apart from a few core attributes, there are no guarantees about which attributes an entry contains. For example, some organisations may provide person entries with rich attribute sets, while others may have person entries lacking even basic communications information such as telephone numbers and email addresses. Even more important from the point of view of finding entries, there is no guaranteed consistency of name formats between organisational databases: some databases may provide person entries with initials and surnames only, while others may provide full forename information.

The collection of entries in the directory is referred to as the Directory Information Base (DIB). However, as the entries are arranged hierarchically, we almost always refer to the information being held in the Directory Information Tree (DIT). A typical fragment of the DIT is shown in Figure 2.3.

Figure 2.3 illustrates several important concepts. First, entries for large objects, such as countries and organisations, are held towards the root of the DIT. Entries for smaller objects, such as organisational units (departments) and people, are held towards the leaves of the DIT.
The second important concept is the way that entries are named. One of the attribute type-value pairs in the entry is used to name the entry: this is the entry's Relative Distinguished Name (RDN). The choice of RDN attribute type is not arbitrary. The standard suggests which attributes should be used for naming entries: details are given in Section 2.5.3. From Figure 2.3, we see that the RDN for the author's entry is:

\texttt{cn=Paul Barker}

This name must be chosen so that no two sibling entries (i.e., entries with the same parent node) have the same name. This requirement for uniqueness means that longer name forms such as "John Peter Smith" are often preferred over forms such as "J Smith". If we concatenate the RDNs of all the entries from the root of the DIT to a given entry, we get the Distinguished Name (DN) for that entry. The DN for the author's entry is:

\texttt{"c=GB, o=University College London, ou=Computer Science, cn=Paul Barker"}

A DN is unique within the DIT. DNs are very important within X.500 as they are used as parameters of X.500 operations to identify which entry is to be read or modified, or which part of the directory is to be listed or searched.

It is important to note that while the four-level DIT hierarchy depicted in Figure 2.3 is typical of much of the directory, many other hierarchical structures are possible. There are two main variants. First, some parts of the DIT have one or more layers of locality entries between the country and organization entries. These are used, for example, to represent states in the US and Australia. A second variant is for organisations to have multiple layers of organisational unit entries. This allows organisations greater flexibility to represent their organisational structure within the
directory. Another variant often adopted by small organisations is to omit organisational unit entries.

A third important concept illustrated by Figure 2.3 is that the directory database is distributed amongst a collection of servers called Directory System Agents (DSAs). Each DSA holds the master data for part of the DIT; a DSA may also hold copies of other entries. In principle, although this would be a farcical arrangement, the DIT could be held one entry per DSA. In practice, a typical DSA run by an organisation in the current NameFLOW-Paradise directory holds three types of information.

- It holds the department and person entries (and possibly other types too) for that organisation.
- It must also hold knowledge of how to answer queries requesting information held in other servers. The amount of knowledge held can vary from knowing just one other DSA to try, to knowing about all other DSAs in the DIT.
- It will probably hold shadow copies of information for some/all country and organisation entries. It will do this as it speeds up query resolution dramatically.

The distribution of the DIT, both in terms of master and shadow entries, amongst DSAs has a direct bearing on the response times of the directory. There is no guarantee that shadow copies of information will be up-to-date as the directory allows for temporary inconsistency between master and shadow entries. However, information in a white pages directory tends to change quite slowly. Therefore, most white pages DUAs are happy to use shadowed information during the query resolution process as this speeds up querying; users and/or their DUAs have the option of insisting that only master entries are used in query resolution if the user/DUA has to be sure that the information is fully up-to-date.

However, the distribution of information amongst servers has no bearing on querying algorithms: querying is governed by the naming hierarchy of the DIT rather than the physical distribution of the information.

We have already noted that X.500 provides three operations for retrieving information from the directory: read, list and search. These operations vary considerably in their querying power.

- The read operation retrieves attributes from a named entry in the directory.
- The list operation returns the names of a set of sibling entries: i.e. all entries share the same parent node.
- The search operation retrieves attributes from entries that match certain search criteria, e.g:

  find all the person entries in the subtree with the parent node "{c=GB, o=University College London}" with a surname of "barker" and the word "research" in their job title.
CHAPTER 2. MATCHING ALGORITHM ISSUES

We have now introduced the essential components of the directory – the data and its structure, and the tools to query it. In the next section we examine two approaches to querying the directory.

2.3 Querying Abstractions

X.500's data structure and its querying operations allow two approaches to finding information in the directory: a DUA can either browse the directory, or enter name input to query the directory. Geller and Lesk describe these approaches as choices and commands respectively [GL83].

Browsing: In the browsing abstraction, a user is presented with a series of menus offering choices: the user makes a selection. This process of selecting from menus of choices continues until the required information is found. Note that, in the context of a directory service, a user is not required to enter any name information when browsing; the user merely has to select names from choices offered by the system.

Name input: In the name input abstraction, a user is asked to enter name information and the DUA attempts to find an entry/entries corresponding to the user's input. In general it is not necessary for the user's input to exactly match the name in the directory: directory systems offer a variety of types of looser matching including substring, regular expression and approximate matching.

In fact these two querying techniques can be used in tandem: for example, a DUA might allow a user to browse through lists of country or organisation entries, but insist on the user entering a name for the required person entry.

2.3.1 Browsing

A number of DUAs use the browsing, or choice driven, querying abstraction. With this style of querying, the DUA "lists" some entries in the directory and presents the results to the user. The user selects one of the results, and the selected entry can be used as the base object for further operations. This style of querying can be repeated until the leaves of the DIT are listed; an entry can then be selected and its attributes returned to the user.

It is important to note that the listing of entries is not always implemented using an X.500 list operation. One reason for this is that the X.500 list operation lists all entries (subject to size limits) immediately beneath the base object, whereas DUAs are often interested only in one or two types of entry. For example, a white pages DUA may be interested only in person and department entries within an organisation, whereas the organisational DIT may contain entries for computer hardware, network services, mail distribution lists, and even an entry for the DSA itself. The frequently adopted solution is to use a search operation to emulate a list operation, using a search filter that specifies that all entries of certain object classes should be matched. However, there is an important difference between the list operation and its emulation through
2.3. QUERYING ABSTRACTIONS

the search operation. The list operation returns a set of RDNs, whereas a search operation returns a set of DNs. If there are network bandwidth constraints, these larger results may be a problem. Bandwidth issues are explored further in Section 2.8.

A number of X.500 DUAs offer the list style of querying, including: POD [FMN90b], maX.500, go500gw, web500gw [HS96] and the author's DE [Bar91]. My subjective impression is that this querying style is popular with users. It apes the menu-driven approach to finding information or facilities that helped to popularise Macintosh computers. An advantage of the approach is that users are shown what is available in the directory, which obviates the need for guessing how to phrase a query. Geller and Lesk have found that this approach is favoured by users who are unfamiliar with a service or its data [GL83].

However, there are a number of problems with the approach that suggest that listing may be of limited use in querying the directory.

- Listing all entries at each level in the DIT hierarchy is not feasible if the DIT has a flat structure. While one could ask a user to select from a list of all the countries in the world, selecting a person entry from a list of 10,000 entries is impractical.

- Large result sets may be slow to transfer over the network. The 1993 standard’s paged results facility eases this problem: see Section 2.6 for more details.

- DSAs usually impose administrative limits that restrict the number of entries that may be returned to a user. This is often done to prevent trawling of information; it is sometimes done to restrict resource consumption. Whatever the reason, such restrictions inhibit listing.

- Quipu DSAs can enable a special set of list and search access controls (LACLs and SACLs) that specifically try and prevent trawling of data; these controls are specified in [HKH91]. These controls restrict the types of query that can be asked and/or the number of results that can be returned for any given set of data.

Some of these points are developed further in Section 2.8. Despite the fact that browsing is popular with users, implementors, understanding the limitations described above, have sometimes been reluctant to provide this style of querying. For example, Mahl indicated in [Mah91] that he would only implement browsing if he received a lot of negative feedback about this input driven approach!

Afifi and Huitema do not make use of list, as they say that list operations will often be denied in order to prevent commercially sensitive information, such as organisational structures, being obtainable by unauthorised users [AH92].

Finally, we should note that other directory service protocols such as LDAP, CCSO, Whois++ and SOLO all omit explicit list operations, although, as we noted earlier, listing functionality can be emulated by search operations.
2.3.2 Matching Name Input

With the name input abstraction, a user enters a query which consists of one or more name parts. These name parts are analogous to the indvidual lines of an address on a paper mail envelope, or the individual components of an RFC 822 email address. A user hopes that these name components collectively are sufficient to identify the required entry. In X.500 jargon, a user's guess at a directory name is termed a *purposed name*. The process of finding the required entry involves trying to match a user's input with attribute values in entries in the directory. The matching criteria are determined by the directory user or his/her DUA. Note that the input can be matched against any attribute, not just naming attributes. Furthermore, there is no need for a user's input to exactly match the values in the directory, but the query resolution process will generally be simpler the more nearly the query name components match the target entry's DN.

The name components of a query can be entered as several separate strings, or all in one go as a single string. The following are typical queries for the author's entry in a variety of formats: the user's input is in bold font.

**DE:** DE prompts for input component by component; the input is explicitly typed.

Enter person's name: barker
Enter department name: cs
Enter organisation name: ucl
Enter country name: uk

**UFN:** The UFN specification [Kil95a] is freer in format and does not explicitly categorise, or type, the name components. The name components should be in little-endian order, and are usually comma-separated. The flexible format means that a user is free to choose how many name components to include in his/her input. The UFN algorithm [Kil95b] defines a strategy for guessing the types of the name components, based on the number of components entered by the user.

Enter UFN: barker, university college, gb

**Email name** Afifi and Huitema use the RFC 822 format [Cro82] for illustrating their example queries. Again there is no explicit typing of name components.

Enter query: barker@cs.ucl.uk

A user will not always have to supply all the name components, as a DUA will often supply default values when a user omits a value. This usually happens for queries which are in some sense *local* to the directory user: e.g, the entry sought is for someone who works in the same department or same organisation as the person making the query.

The directory has two operations which can be used to match user input to directory names and values: *read* and *search*. We will examine how these operations work, and their relative merits, in the following sections.
2.4 X.500's Read Operation

Read is the simplest querying operation: the user provides the DN of the requested entry and the directory returns that entry, or selected attributes from that entry. A read operation can only be successful if the user-supplied DN is exactly correct.

However, a read operation can still return useful information even if the DN is not fully correct. In such cases, the read operation fails with a \textit{nameError}. This indicates how many RDN components of the user's purported name match names in the directory. Some examples will help to explain this.

Let us assume that a user formulates the following query, in UFN format, when trying to find the author's entry. We will assume that the names follow the standard DIT hierarchy of people within departments, within organisations, within countries.

\begin{verbatim}
barker, cs, university college london, gb
\end{verbatim}

An attempt to read this entry fails; the purported name is wrong. The directory then works out how much of the user's query is correct. This procedure starts by comparing components at the root of the DIT, and continues until a component cannot be matched. In this example, "gb" matches a country entry name, "university college london" matches an organisation name within "GB", but no department entries beneath \{c=GB, o=University College London\} have the name "cs". The read operation returns the DN \{c=GB, o=University College London\} in the matched component of the \textit{nameError} structure. The DUA must now use some alternative strategy for resolving the department name, or ask the user to provide alternative input.

Let us now consider a query of:

\begin{verbatim}
paul barker, computer science, university college london, uk
\end{verbatim}

In this case, the country name of "UK" does not match the directory name for the country entry and so the matched parameter is null. Note that due to the directory's top-down name resolution procedures, no indication can be given that, in this case, all the other name components are correct.

The limitations of the read operation should now be apparent. Fortunately, the search operation helps us to match query name components that do not exactly match the directory RDNs. However, before we go on to examine the capabilities of the search operation, it is worth briefly considering whether it is worth using read operations at all. The usefulness of read as a querying tool depends on two main issues:

- How closely does user input correlate with directory names? If the correlation is poor, most read operations will fail. Maybe it would be better to use a list or search strategy from the outset?

- There is a general assumption that read operations will be quicker than search operations, although maybe not much quicker with good database indexing. If reads really are quicker
than searches, there is potential for speeding up querying algorithms by using reads instead of searches.

Afifi and Huitema [AH92] have advocated a strategy that makes considerable use of read operations. This strategy has been refined by Woermann and Pacchioni as the AFRO algorithm [WP94]. The strategy is to use a read operation initially. If the purported DN in the read operation is not entirely correct, the read operation fails with a `nameError`. The strategy is then to examine the matched component of the `nameError` structure to determine which components of the purported name, if any, are correct. The matched part of the name is then used as the base object for a search operation to try to resolve the name component that caused the read to fail. This strategy of first trying to read an entry but falling back on search operations to resolve non-RDN components is used until the query is fully resolved.

The Afifi and Huitema work is interesting as no other DUA designers have made as much use of read operations. However, does the evidence support this approach? Aspects that need examining include:

- The Afifi and Huitema examples show names that are quite similar to their corresponding directory names. How true is this in general? This thesis contains a substantial analysis of this in Chapters 3, 4 and 5.

- Do other implementations share the response time characteristic cited by Afifi and Huitema of there being a two-to-one speed advantage for read over search? Figures for the Digital implementation show search and read speeds being almost identical for most types of search [Eme95]. Alternatively, the figures for the EAN system show that successful reads are substantially faster than searches, but that unsuccessful reads are slower [NBGS92].

- Is it reasonable to categorise all search operations together in this way? We might find that some searches are comparable to reads in performance, while other searches with complex Boolean filters are much slower. The EAN results show that the response time of their system varied widely for different search operations.

- Speed differentials that hold for local DSA access may not be true for a remote DSA, as remote operations entail additional communications overheads.

### 2.5 X.500's Search Operation

For the reasons outlined in Sections 2.3.1 and 2.4, the majority of querying algorithms make extensive use of `search` operations. In this section we explore the problem of matching user input to directory names, and introduce the facilities provided by the search operation.

A search operation has three main parameters:

- The base object specifies a DN, which identifies which subtree in the DIT is to be searched.
2.5. X.500's SEARCH OPERATION

- The *subset* argument of the search parameters allows a DUA to specify whether it wishes to search: the base object only; one level below the base object; the whole subtree beneath the base object.

- The *search filter* specifies the criteria that an entry must satisfy to be matched in a search.

A search operation has several other parameters. Varying these and the search filter gives scope for very many different mappings of user queries onto X.500 search operations. The core work of this thesis is to understand how to do this mapping as well as possible.

### 2.5.1 Search Filters

The principal component of a search operation is the search *filter*, as this selects which entries should be matched. In its simplest form a search filter consists of a single *filter item*. A search filter item takes the form:

\[
\langle \text{attributeType} \rangle \langle \text{matchingType} \rangle \langle \text{presentedValue} \rangle
\]

For example, a filter item to match all entries with a surname of "barker" would take the form:

\[\text{sn=barker}\]

More complex filters can be constructed by using Boolean combinations of filter items, using the AND, OR and NOT operators. We will see cases where more complex filters are required in the next section.

We will often need to describe search filters throughout this thesis, and a full description of how search filters are represented is given in Appendix A.2.

### 2.5.2 More Sophisticated Filters

The need for more sophisticated search filters should become clear by considering some example queries, and how they can be matched with the entry in Figure 2.2. We will see in Chapter 3 that the following examples are all realistic queries:

**paul barker** This is easy to resolve: the user input exactly matches the entry's RDN.

**p f barker** This input matches one of the commonName attribute values.

**p barker** The entry does not include the commonName form variant with a single initial.

**p. f. barker** The entry does not include any versions of commonName with dots after the initials.

**mr p barker** The entry does not include any commonName variants including personal titles.
The entry includes commonName variants with forenames only, and initials only, but not with this mixture.

This input can be exactly matched against the surname attribute.

This input can be exactly matched against the userid attribute.

There are no surname-first commonName variants in the directory.

This input would exactly match the user's entry if the name was correctly spelled.

The situation is complicated further as there is as much variety in the name forms used in directory data as there is in user input. Several points emerge from these examples and the earlier discussion.

- User input is often different in form to the values in the directory. Exact matching will often not find the required entry.

- We are not restricted to matching user input against the naming attribute: for example, for person entries it may be useful to match against surname and userid attributes, as well as or instead of commonName.

- With user interfaces such as DE and those accepting the UFN format, it may not be obvious whether, for example, a person name query is: a surname; a forename; a userid.

Although my examples illustrate the problem of matching person name input with name forms in the directory, there are analogous matching problems for country, organisation and department name input.

It should now be evident that a search strategy based on a single filter item using exact matching will fail to find many entries. X.500 provides a number of facilities that allow us to build a more sophisticated matching strategy.

### 2.5.3 Attribute Types in Search Filters

We have already seen several examples where the attribute type used in a search filter item is not a naming attribute. In fact, while we can in theory use almost any attribute for matching, certain attribute types are widely used in practice. These are summarised in Table 2.1.

When there is a choice, how do we decide which attribute type to use in a search filter? In some cases, there is an obvious choice. Two letter country names are usually the ISO 3166 country codes [ISO88a] used for naming country entries, and so the DUA should use the countryName attribute in the search filter. Longer input should be matched against the friendlyCountryName attribute. However, this is not foolproof: "UK" is not an ISO country code. The lesson from this example is that a name matching algorithm must be flexible and be prepared to try more than
2.5. X.500’s SEARCH OPERATION

<table>
<thead>
<tr>
<th>Object type</th>
<th>Long attribute name</th>
<th>Short att. name</th>
<th>Naming attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Country</td>
<td>countryName</td>
<td>c</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>friendlyCountryName</td>
<td>co</td>
<td>No</td>
</tr>
<tr>
<td>Organisation</td>
<td>organizationName</td>
<td>o</td>
<td>Yes</td>
</tr>
<tr>
<td>Organisational Unit</td>
<td>organizationalUnitName</td>
<td>ou</td>
<td>Yes</td>
</tr>
<tr>
<td>Person</td>
<td>commonName</td>
<td>cn</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>surname</td>
<td>sn</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>userid</td>
<td>uid</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 2.1: Attribute types commonly used in search filters

one type of matching. A similar need for flexibility of approach is often required when trying to match human name input: is a query of “James” a forename, a surname or a userid?

The X.500 standard offers some guidance in how to choose the most appropriate attribute type for matching. The searchGuide attribute may be included as an attribute of entries which are likely to be base objects of search operations: e.g., country or organisation entries. The searchGuide suggests which attributes should be used to find entries, for a given object class, in the subtree below that entry. We examine if searchGuide is used in practice in Chapter 4.

The 1993 standard has a feature that promises to simplify matching: extensible matching allows a DUA to request typeless matching, such that a query name is matched against all string attributes in entries within the scope of the search operation.

The 1993 standard also includes additional attributes for representing people’s names. Whereas the 1988 standard has commonName and surname (userid is defined in RFC 1274), the 1993 standard has these and several new ones: givenName, initials and generationalQualifier.

2.5.4 Boolean filters

A DUA can use more than one filter item, combined using the Boolean operators AND, OR and NOT. For example, we mentioned above that a DUA may not always know the exact semantics of the user input. If a user enters the string “barker” as a person name query, there are at least three common possibilities: the input is a surname; the input is a forename; the input is a userid. A DUA can test for all three possibilities by using the filter:

\[(\text{sn}=\text{barker}) \text{ OR } (\text{cn}=*\text{barker}*) \text{ OR } (\text{userid}=\text{barker})\]

A white pages DUA is only interested in certain types of directory entry; we noted earlier that the type of an entry is indicated by its objectClass. We can use a Boolean filter so that only entries of the appropriate type are selected. For example if we want to search for person entries with a commonName value of “paul barker”, we can use the filter:
(cn=paul Barker) AND (objectClass=person)

2.5.5 Types of Matching

X.500 offers several different types of matching. The 1988 standard has exact matching, several types of substring matching and approximate matching for the string based attributes used in a white pages directory. In all cases, matching is done case-independently: “Barker” matches “barker”.

So far, almost all the examples have used exact matching. Before considering the other types of matching, we need to clarify what is meant by exact matching. User input is an exact match for a directory value if all the characters are exactly the same, except that multiple instances of space characters are treated as a single space for matching. No allowance is made for other minor syntactic differences. For example, “P. Barker” does not exactly match “P Barker”.

The standard provides several types of substring matching. The following filters, where the asterisk is used as a wild-card for zero to many characters, show the capabilities of this type of matching:

- cn=paul*
- cn=*fred*
- cn=*barker
- cn=p*barker
- cn=p*f*barker

Misspelled input, such as “barkar”, can only match the author’s entry using approximate matching. One complication, from a user’s perspective, is that the standard does not define what approximate algorithm should be used. Identical queries may not be handled alike by different DSA implementations. In practice, however, most DSAs use Soundex [Knu73] for their approximate matching algorithm.

There is a possible further role for approximate matching. An approximate matching algorithm such as Soundex compares two tokens for similarity. However, another matching problem with a directory service occurs when user input is correctly spelled, but is in a different form to the directory name. Thus, a basic approximate algorithm such as Soundex needs to be used in conjunction with sets of rules that cope with mis-matches due to differences of name forms. Examples in Table 2.2 illustrate some mis-matches of form that could usefully be handled by approximate matching.

Quipu’s approximate matching algorithm works in some of these cases. Details of the Quipu implementation of approximate matching are given in Appendix D. We examine approximate matching issues in detail in Chapter 6.

The 1993 standard offers some new matching facilities. Word matching is used to match the input against any word in the directory name, although the definition of “word” is not specified.
### 2.5. X.500’S SEARCH OPERATION

<table>
<thead>
<tr>
<th>User Input</th>
<th>Directory Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>barker, paul</td>
<td>Paul Barker</td>
</tr>
<tr>
<td>paul barker</td>
<td>P Barker</td>
</tr>
<tr>
<td>p barker</td>
<td>Paul Barker</td>
</tr>
<tr>
<td>department of chemistry</td>
<td>Chemistry</td>
</tr>
<tr>
<td>university of cambridge</td>
<td>Cambridge University</td>
</tr>
<tr>
<td>u.k.</td>
<td>United Kingdom</td>
</tr>
<tr>
<td>uk</td>
<td>United Kingdom</td>
</tr>
</tbody>
</table>

Table 2.2: Examples of form mis-matches

**Keyword matching** is more flexible still: it is similar to word matching except that the exactness of match is also left undefined.

Further potential is offered by the 1993 standard’s *extensible matching* facility. This allows a user to specify that a non-standard matching rule be used to match a query value. The usefulness of this feature depends on DSA implementors providing additional matching algorithms.

Given this array of matching facilities, a DUA designer has to devise a strategy for using these capabilities as effectively as possible. The *searchGuide* attribute potentially helps as it can specify which matching types are most appropriate. The 1993 standard also offers facilities to indicate which matching rules can be used for a given portion of the DIT: this is achieved by the *matchingRules* and *matchingRuleUse* attributes.

The reader who is familiar with the X.500 standard may have noted that there has been no mention of *ordering* matching, where values may be matched according to whether they are *greater* than or *less* than the presented value. This type of matching is implemented for strings in the Quipu system. The feature is potentially useful as it could be used to find, for example, all the entries with surnames beginning with ‘s’ by the query:

\[(sn>='s') \text{ AND } (sn<='szzzz)\]

The problem with ordering matching, as Chadwick notes in [Cha94], is that there is no agreed collating sequence for the string syntaxes used by all X.500’s white pages naming attributes. Nevertheless, as the experience with Quipu shows, implementors may provide this facility. However, DUA designers cannot rely on its implementation.

#### 2.5.6 Combinations of Search Filters for Different Types of Matching

Due to the issues discussed in the previous sections, it is often appropriate for DUAs to use more than one type of matching within their searches. However there are several ways of doing this.
Boolean OR: The filter items can be combined as a Boolean filter in a single search operation. For example, if the user input is "paulb", the search filter might be:

\[(\text{sn}=\text{paulb}) \text{ OR } (\text{sn}'\text{=paulb})\]

This finds any entries with a surname with the value equal to or approximately equal to "paulb". This is the approach adopted by the UFN algorithm described in [Kil95b].

Sequential searches: The filter items can be tried sequentially in separate search operations, such that looser matches are only tried if more precise matches fail. Using the filter items in the previous example, the first search uses the filter

\[\text{sn}=\text{paulb}\]

and if that fails, the filter

\[\text{sn}'\text{=paulb}\]

is tried afterwards. This is the approach used by DE. Throughout this thesis a sequence of searches is indicated by comma separated search filters. The above example, requiring two searches of the directory, is thus represented by:

\[\text{sn}=\text{paulb}, \text{sn}'\text{=paulb}\]

Parallel searches: The filter items can be tried in parallel as separate search operations. This is the approach used by XLookUp [Mah91], now usually known as XLU [MF96]. XLU waits for the exact match to return. If this contains the required result, the result from the approximate match operation is discarded if the operation has completed, or the approximate match operation is abandoned. If the exact match does not contain the required result the approximate match results are used.

There are pros and cons to each of these strategies. The Boolean OR strategy only ever requires a single search operation to be processed, but the results are (in the example above) a mixture of exact and approximate matches. This has two adverse effects. First, the result sets tend to be larger as approximate matches are always returned even when an exact match is achieved. Second, the DUA has to process this mixture of matches if the exact matches are to be offered to the user ahead of the approximate matches, a feature, not surprisingly, that users find desirable [Bar91].

The sequential searches strategy automatically presents the user with exact matches ahead of approximate matches. A cost of this strategy is that more than one search operation is required if the user input does not match a directory name attribute value exactly. This also adds to the resolution time of the query. The author has proposed in [Bar94] that a useful enhancement to X.500 would be to allow users to specify a sequence of filters in a single search operation: the filters should be applied in turn until results are found. Peterson had previously specified
this functionality in the Profile naming service using interpret functions [Pet88]. The SOLO protocol [HPZW94] allows servers to pass suggested matches back to users, if the query cannot be matched exactly: in effect SOLO servers try exact matching first, and then try approximate matching if no exact matches can be found. The WhoWhere system [WW96] provides a similar feature. It returns three classes of matches in ranked order: first, “highly relevant” matches; second, “probably relevant” matches; finally, “possibly relevant” matches. The 1993 standard provides a mechanism by which sequential searches can be provided. The extensible matching facility could be used to request a matching technique that applies a predetermined sequence of filters to the query data.

The parallel searches strategy distinguishes between exact and approximate matches, and, if approximate matching is required to resolve the query, it should be faster than the DE strategy as there is no delay in dispatching the approximate match query. However a cost is that two queries are always issued, and this places more burden on the DSA answering the queries.

2.6 Other Parameters Controlling Querying Operations

X.500 provides some other parameters that influence the querying strategy. The main role of these parameters is to control the amount of information that is returned to the user. The most important parameters are:

- search scope;
- entry information selection;
- size limits;
- paged results.

Search Scope

The subset argument of the search parameters allows a DUA to specify whether it wishes to search: one level below the base object; the whole subtree beneath the base object. Some of the issues are explained by considering how a DUA could look for a person entry within an organisation in the directory. If the directory user does not know which department to search for the entry they are seeking, the user has to do a subtree search of all entries within the organisation. However, if the user does know the department, the user may be able to find the required entry by doing a one level search beneath the department entry. The smaller scope of the one level search means that the search may be quicker and that result sets will be smaller: for example, whereas there may be a hundred “Smiths” in an organisation, there will only be a few “Smiths” in any department.
CHAPTER 2. MATCHING ALGORITHM ISSUES

Entry Information Selection

A user can control how much information is returned for every matched entry by means of the \textit{entry information selection} parameter. It applies to both \textit{read} and \textit{search} operations. The parameter controls two things. First, a user can request that attribute types only, or types and values, are returned.

Second, a user can request that all attributes are returned, or specify a list of attributes that should be returned. This feature was useful with 1988 systems as it meant that large attributes, such as \textit{photo} and \textit{audio} (both defined in RFC 1274 [BK91], could be omitted from the request: retrieving large attributes has often been the cause of poor directory response times. The 1993 standard offers an improved facility to achieve this end: the \texttt{attributeSizeLimit} control allows a DUA to indicate that attributes larger than a certain size should not be returned.

This parameter has no direct bearing on which entries are found. However, the ability to restrict the size of results means that it may be practical to use an algorithm that tend to match more entries, which we might otherwise dismiss for requiring too much bandwidth.

Paged Results

The 1993 standard offers another very useful facility to the DUA designer for use with list and search operations. The standard now includes a \textit{paged results} facility, which allows a DUA to stipulate the maximum number of results that should be returned in one go. If the result set is bigger than the specified limit, the DUA can request further results in similar size chunks.

Another useful aspect of paged results is that the user can specify that results be ordered according to sort keys. This could be used, for example, to present person entries to the user in surname order.

Size limits

The \texttt{sizeLimit} allows a DUA to specify the maximum number of entries that should be returned for a \textit{list} or \textit{search} operation. This limit has been widely used in practice, as the Quipu system's DUA library sets a default size limit of 20 entries; this value is often unaltered by DUAs. Experience with this setting has shown that it is often too restrictive: it prevents operations such as listing department names; it also restricts search operations more than might be imagined – some evidence is presented in Chapter 5 which shows how often large result sets can occur with actual queries. One approach to these problems with this limit is not to set it: this is DE's strategy. If it is used, it must be set carefully and in harmony with the overall querying approach and the likely number of results that will be returned.
2.7 The Role of the DUA in Name Matching

So far in this chapter, we have examined the facilities offered by X.500 to match user input against values in the directory. An important element of a DUA design is which X.500 facilities it uses to provide a service. An equally important element is the support the DUA gives to the user in framing queries, and matching them as efficiently as possible. This section reviews the role of the DUA in name matching.

2.7.1 Precision and Recall

A querying strategy that, other things being equal, delivers the correct result, but does so along with 1000 unwanted results, is obviously poorer than a strategy that finds as many correct results, but that returns result sets of two or three entries. Ideally, we want a search algorithm which has high precision and good recall, where these terms are defined in [Sal68] as:

**Precision:** The proportion of retrieved entries that are relevant.

**Recall:** The proportion of relevant records that are retrieved.

Using these definitions we can see one of the problems with browsing operations: while they have 100% recall, they have very low precision.

2.7.2 DUA Support when Phrasing Queries

Let us assume that a DUA is search-oriented and requires a user to enter name information. DUAs offer widely differing levels of support for phrasing users’ queries. There are several issues that a DUA has to handle explicitly or implicitly:

**Input types:** A DUA may explicitly categorise, or type, its input. This is usually done by the DUA prompting for certain types of input, or by the DUA providing certain boxes to fill in. Alternatively, user input may be free form, with the DUA algorithm assigning types to the input by trial and error, usually guided by certain schema constraints. This is the approach used by UFN.

**Format of name parts:** In general, DUAs allow users to type what they want for a given field. This is true for DE and any DUA using UFN queries. In some cases, DUAs give guidance when entering personal names, either by splitting the input into forename/initial and surname fields, or by rigidly specifying acceptable formats: the “four11” directory does this [FE96].

The way these issues are tackled leads to DUAs with radically different querying algorithms. In general, the more freedom that a user is given in the way he/she formats a query, the more ways there are that the query can fail, and the more work that a DUA must do behind the scenes to interpret the query correctly.
The DE DUA [Bar91] and the UFN format and algorithm [Kil95b] provide contrasting examples. DE presents a fixed object class hierarchy to the user, although DE's algorithm allows for some flexibility of DIT structure. DE's simplicity of view is meant to encourage correctly structured queries, and to make the directory easy to use in most cases, but there are some DIT structures that cannot be queried successfully by DE – the main examples are structures which make extensive use of locality entries.

At the other extreme, the UFN algorithm is very flexible, and almost any DIT structure can be queried. However the price of the flexibility is that it is easy for users to specify UFN queries that look reasonable, but which bear little relation to the DIT and which are thus very difficult to resolve. I have presented evidence elsewhere on the difficulties users have with UFN queries [Bar95a].

2.7.3 Pre-processing Users' Input

The simplest option for a DUA designer is to take a user's input and include it, as it is, in search filters. However, in Table 2.2 we noted a number of examples where the form of the query and the directory entry differed enough that even approximate matching was not guaranteed to succeed. For example, consider a query of "Cambridge university" for the directory entry of "University of Cambridge". Although the essence of the query is correct, the search will fail unless:

- the approximate matching algorithm supports this type of matching;

- the DUA transforms the query into a filter such as:

\[(o=*cambridge*) \text{ AND } (o=*university*)\]

We will see in Chapter 5 that this type of form mismatch occurs quite often.

A particular problem with user input is that it may include stop list words. The notion of stop list words is common in bibliographic databases: stop list words include the definite and indefinite articles, prepositions and conjunctions. The words are removed from book title indexes, as these words add little to the semantics of the input, but often inhibit matching. Similar issues affect matching in a directory service. For example, user input of "the university of foobar" does not straightforwardly match a directory name of "University of Foobar". We examine the need to strip out stop list words in Chapter 5. Again, an alternative approach to explicitly removing stop list words is to regard this type of matching as within the scope of approximate matching. We examine possible roles for approximate matching in Chapter 6.

Another common problem occurs where a user types too much when forming a query; it is easier to match short strings than long strings. A typical example of this is where a user enters a query of "computing science" whereas the department name is "Computer Science". If the user is unsure of the department name, a query of "comp" will substring match either of these variants. We will examine the role of truncating queries to help with matching in Chapter 5.
2.7.4 Post-processing Results from the Directory

A DUA may manipulate result sets in several ways, rather than just handing the results to the user. A DUA may select some results as being “better” than others, and offer them to users before the less good matches. For example, a DUA that has used any substring matching on input of “smith” may prefer a result of “Smith” over “Arrowsmith”. The UFN algorithm uses this approach, preferring exact matches, over substring matches, over approximate matches.

DUAs may sort results into lexical order. The 1988 standard allows a DSA to return matched results in any order, whereas users generally prefer to see alphabetic ordering by surname. DE sorts its results. The 1993 standard provides the paged results facility that obviates the need for this type of post-processing: see Section 2.6.

2.7.5 How Hard Does the DUA Try to Match User Input to Entries?

Another factor which has an impact on the querying algorithm is how hard the DUA should try to find the correct result. At one extreme, a DUA might expect users to formulate their queries closely in line with DIT names and with no spelling errors. For example, a mail user agent DUA might insist that a directory query is resolved to a single directory entry, using exact matching at every step [AH92].

Other applications might allow a more permissive DUA which can accept misspellings, or is prepared to make exhaustive searches of the directory if some name components of the query cannot be matched straightforwardly. Consider the following two queries, specified in the UFN format:

```
barker, cs, ucl, ac, uk
barker, computing, ucl, uk
```

The first of these queries contains an extra component, “ac”, but can otherwise be resolved against the author’s entry. The second example contains a component, “computing”, that is nearly right but that does not match “computer science” using the standard matching facilities. A DUA can either give up when it finds a component that does not match, or it can ignore a failure, accept that users make mistakes, and try to get the best match using the remaining components of the query.

2.8 The Directory Environment

The directory is not a uniform environment where every query is equally successful and takes the same time to service. The directory varies in many ways: some types of X.500 operation may take longer to service than others; queries for remote information may take longer to resolve than local queries; querying strategies that work well on small databases may not be suited to larger databases; users may have high or low bandwidth access to the directory; and so on. Furthermore,
DUA designers need to be aware that DSA and data administrators may have imposed a variety of restrictions on querying their data. These may, for example, prevent certain attributes being used in search terms, or limit the number of results that can be returned to a user.

This section examines these environmental factors that influence and constrain the design of DUA querying algorithms. Understanding these issues will allow a DUA designer to tune a DUA to get the most effective performance.

2.8.1 Response Times

It is hard to make judgements about the relative merits of any querying strategy, unless we understand the factors determining response times. It may be that strategy A, which is 10% better at finding correct entries than strategy B, is twice as slow. The DUA designer must be aware of these trade-offs.

The following issues may all have significant bearing on query response times. Some of the issues are specific to searching the directory, others apply to all querying strategies. We examine many of these response time issues in the NameFLOW-Paradise environment in Appendix G.

Filter Complexity It is unlikely that all search operations on a given DSA will take the same time to process, irrespective of filter complexity. It would help a DUA designer to know characteristics such as:

- the relative response times of exact, substring and approximate match operations;
- the relative response times for simple and Boolean search filters.

First Queries and Subsequent Queries X.500 is a connection oriented protocol. A connection has to be established to the directory before any operations can be invoked. A consequence of this is that the marginal cost (in time) is greater for the first operation on a DSA than subsequent operations. An initial query may also have to bear operating system paging overheads. These factors complicate the model for operation response times. We investigate this issue in Appendix G.

Database Size A search that works well on small amounts of data may not work so well on large databases. Possible reasons for this include: the technique returns too many matches from large databases; the search is not indexed and the search time increases linearly with database size.

Database Indexing Indexing searches can provide enormous increases in search speed. Searches that cannot readily be indexed are likely to be orders of magnitude slower than indexed searches.

Local or Remote Queries Queries for information about people in the same organisation as the user will generally be answered by a single, local DSA. Queries for information about
people in other organisations will generally have to be passed on to one or more further DSAs: how many DSAs depends on several factors, in particular how much directory knowledge is held by the local DSA. The overhead of connecting to additional DSAs means that queries for remote information will generally take longer than local queries.

**The Local System is Well Understood** The characteristics of the local DSA are probably well understood by the person configuring a DUA. This allows queries to be tuned to exploit the capabilities of the local system. No such tuning is possible for queries to remote systems, particularly if they use different implementations of X.500.

**Load on a DSA or the Directory in General** The load on a DSA may be such that DUAs will do better to be economical with their queries, favouring simpler filters and fewer operations. Furthermore some parts of the directory may usually be slow to respond; in such cases a more economical querying strategy might work better.

The Quality of Service database described by the author in [Bar93], which keeps a record of response times for different parts of the DIT, could help a DUA to tune its querying strategy.

**Number of Results** It is quicker to return a single result to a user than a thousand results: one result is quicker to encode, quicker to transfer and quicker to decode. The speed difference depends on CPU power and network bandwidth.

This would be less of a problem if results could be used by an application as soon as they arrived. The Quipu implementation insists that the full result set has arrived before any results are passed to the user. Kille notes in [HK92] that the “streaming” of results is theoretically possible in X.500, although very difficult to implement.

We noted in Section 2.6 that the 1993 standard has a paged results facility which allows large result sets to be handled more gracefully.

**Size of Individual Results** Big results are slower to transfer than small results. It may be more efficient to ask initially for limited attribute sets (using the entry information selection parameter described in Section 2.6) when querying the directory, as the search process may yield a substantial number of matches. The user can then request the full attribute set when the required entry is positively identified. Note that this facility is built into Whois++ with its summary records [DSFW95].

We noted in Section 2.6 that the 1993 version of X.500 has an attributeSizeLimit service control to request that attributes over a specified size are not returned.

**DIT Depth and Naming** We noted earlier that the search operation returns results that include the matched entries' DNs. If these DNs have many components, implying a deep DIT, then the amount of result data will be larger than if the DIT has a flat structure. (This
assumes that the length of name components does not differ radically between shallow and deep DIT structures.)

Radicati suggests that a DIT depth of between four and eight components is reasonable [Rad94]. The evidence in Chapter 4 is that directory administrators are currently favouring names at the shorter end of this spectrum. However, we should note that if longer names are used, this will have some impact on network bandwidth requirements.

**Network Bandwidth** The network between the DUA and the directory may be low bandwidth. A user connected to the directory over a low speed line may prefer a search strategy that minimises the number of bytes that have to be transferred; such a user may wish to limit the number of requests sent and the number of results that are returned. A user connected to the directory over high-speed networks may prefer the superior capabilities of a more profligate search strategy.

Similar arguments apply if the directory as a whole suffers from congested networks.

**Bandwidth Variations** A further possibility is that the networks are loaded unevenly throughout the day. In general, the UK-US link is less busy in the UK's morning as the five+ hour time difference means that US users do not start work until early afternoon UK-time. A search strategy that uses less bandwidth when the network is heavily loaded might help users to get at least some results.\(^1\)

**Replication (or Shadowing) and Caching** A DUA can indicate whether shadowed entries may be used in answering a query. Results from shadow entries are usually acceptable for querying DUAs: the rate of change of the underlying data is typically quite slow for a white pages directory; accessing copies of entries can lead to significant speed-ups.

In fact, some operations on the directory may only be feasible if the DUA designer knows that data is replicated. For example, DE uses a search strategy to resolve country and organisation names. This relies heavily on the fact that the top two levels of the DIT are widely replicated; if they were not, then these search operations would have to be distributed to other DSAs and the operations would thus be much slower.

Additionally the DUA or DSAs may cache entries that have recently been accessed. An advantage of DSA caching is that the cached entries can be shared by a number of DUAs, providing that access controls are not violated. DUA caching has the advantage that it can specifically cache the data most useful to the application.

**System Heterogeneity** In some circumstances it is possible to assess the likely performance of the system quite accurately. This is usually true for a DUA accessing data in a local DSA, where the DSA is connected to the same LAN. The person configuring the DUA knows

\(^1\)Such an approach might have helped in the last quarter of 1995, when the academic community's UK-US link was heavily overloaded.
2.8. THE DIRECTORY ENVIRONMENT

the performance characteristics of the DSA, which attributes are indexed, the bandwidth available, which data is replicated, and so on.

However, the distributed, heterogeneous nature of the directory makes these calculations as much an art as a science when we consider how to optimise the response times for a query which requires access to a remote DSA.

Cost While at present the X.500 directory is free to use, it is conceivable that in the future users might have to pay for using the directory. There is no consensus on how directory services might be charged for. One possibility is that charging might be related to the number and type of operations required to resolve the query, as these measures represent the load on the directory. Such a charging policy would encourage economical use of the directory.

2.8.2 Administrator Limits on Searching and Listing

Organisations differ in how open they are with their data, with organisations such as universities and the military likely to have starkly different policies.

We can reasonably assume that most/all organisations will have some minimum access control policy, to avoid unauthorised modification of their data. However, many other restrictions may be in place and these may all have some impact on a querying strategy.

The first thing to note is that the legal framework on access to data differs from country to country [TH92][JH93]: some countries such as the US tend to openness and allow disclosure [Jen96]; other countries such as Germany place much greater emphasis on safeguarding individuals' privacy. While the law may control what can be stored in the directory and who can access the data, the law may also restrict the way the directory can be searched. For example, French law prevents the use of certain attributes in search filters in a public directory [Lan96].

DSAs may be configured to refuse queries that are likely to consume a lot of DSA resources: one problem with "expensive" queries is that they may temporarily deny service to other DSA users.

A DSA administrator is also likely to limit the number of entries the DSA will return on a list or search operation. Experience with the directory has shown that even when such limits are seemingly set quite high, for example at 100 entries, operations such as listing department names may not be possible.

A more sophisticated way of restricting the number of results that can be returned has been defined in [HKH91]. This paper describes a set of special Search and List ACLs, which are designed to prevent users trawling data from the directory: this is an important protection mechanism required by many organisations, who may be happy to allow access to individual entries but do not want large portions of their database to be copied. As well as allowing an administrator to define specific result set size limits for different parts of the DIT, these special ACLs also allow an administrator to prevent operations that might be used to retrieve a database
in small chunks. This is achieved by setting a minimum length on substrings that may be used in search filters: this prevents repeated operations with search filters of the form \( sn=aa^* \), \( sn=ab^* \), and so on. A problem with this approach is that it may prevent a small proportion of legitimate queries.

### 2.9 Comparison with Other Directory Systems

In this section, we briefly compare X.500's querying facilities with those of some other popular directory systems. We are interested in these other systems as, if they offer broadly similar facilities, it follows that the findings described in this thesis are largely applicable to these other directory technologies. These other systems are:

**LDAP**: The Lightweight Directory Access Protocol [YHK95] is functionally very similar to X.500. This is not surprising as the protocol was developed as part of an attempt to provide access to the X.500 directory from clients that were small enough that they could easily be fitted onto personal computers. The vast majority of access to X.500 servers in the Internet is generated by requests from LDAP clients, passed through LDAP-to-DAP protocol converters. Furthermore, most LDAP queries are ultimately answered by X.500 systems; however, the number of stand-alone LDAP servers is increasing rapidly in mid 1996.

**Whois++**: Whois++ [DSFW95], which is a superset of the older Whois protocol [FHS85], was designed within the Internet community by those who feel that X.500 is too heavyweight for simple tasks and that it fails to solve some important directory service problems.

**CCSO**: CCSO (sometimes called CSO or PH) [HDP96], is a descendant of the CSNET nameserver [SLN82]. CCSO is widely used (over 300 servers) as a simple site directory service, ideally suited for supporting mail and phone look-ups within an organisation. Although CCSO was not designed as a distributed database, work is progressing on a scheme to embrace CCSO servers within a fully distributed system [Ord95].

These three systems, along with X.500, account for the vast majority of white pages systems available on the Internet.

### 2.9.1 Querying Operations

Only X.500 offers read and list operations: the other three protocols only offer search functionality. However, LDAP shares X.500's hierarchical naming model and supports listing emulated by search operations. CCSO and Whois++ have flat name-spaces within their servers, and thus lack the sub-division of the database provided by a naming hierarchy required to make listing/browsing practical.
2.9. COMPARISON WITH OTHER DIRECTORY SYSTEMS

2.9.2 Types of Matching

As one would expect, all the technologies support exact matching. They also all support some form of substring matching: CCSO does not support trailing substring matching as the database does not build indexes for this type of matching.

All the technologies support approximate matching. X.500, LDAP and Whois++ all leave the choice of algorithm open to the implementor. In practice, the choice is almost always Soundex [Knu73]. CCSO includes its own, undocumented, phonetic matching algorithm.

Whois++ and CCSO offer some matching types not supported by X.500. Whois++ supports regular expression matching, and CCSO supports some of the wild-carding features supported by the UNIX shell.

Case-sensitivity in matching differs from system to system. In X.500 and LDAP, the attribute matching rules determine whether matching is case-sensitive or not. In Whois++, the user can specify whether matching should be case-sensitive. In CCSO, all matching is case-insensitive.

2.9.3 Attribute Types

In X.500 (1988) the user has to specify the attribute type to be matched in a search filter. X.500 (1993) allows for typeless matching as one of the features of extensible matching. CCSO matching is based on specified attributes. However the attribute types can be omitted: if they are omitted the system assumes default attribute types of name and nickname. Whois++ allows typed and untyped matching.

2.9.4 Object Classes

X.500 and LDAP use object classes to indicate what type of object an entry represents. Whois++ and CCSO both have a similar feature: Whois++ has templates; CCSO has types.

2.9.5 Booleans

X.500, LDAP and Whois++ all support arbitrarily complex filters built using Boolean ANDs, ORs and NOTs. CCSO allows multiple filter items, and these are implicitly ANDed together.

2.9.6 Other Controls

All systems allow the user to specify whether all attributes should be returned, or just a subset. X.500 (1993) has the attributeSizeLimit service control which allows large attribute values to be filtered out.

All systems allow the system administrator to limit the number of results returned to the user. X.500, LDAP and Whois++ allow a user to limit the number of entries returned.

Whois++ has a feature where result sets with more than a specified number of entries are
CHAPTER 2. MATCHING ALGORITHM ISSUES

returned in a summary form. This type of facility is provided in bibliographic querying proto-
cols [NIS88].

2.9.7 Directory Environment Issues

Many of the issues described in Section 2.8 apply to any type of directory system. For example, database size, network bandwidth, system load, system heterogeneity apply equally to all systems.

All these systems are based on access protocols, and do not mandate a particular type of database. The performance of these systems is in practice largely governed by the speed of the database back-end. There is some evidence that LDAP's protocol simplicity means that its packets may be encoded and decoded faster than X.500's equivalent packets [How95a]. However, it is not clear how much of this disparity is due to differences in implementation quality.

Furthermore, all the systems use connection oriented protocols, although a connectionless version of LDAP has now been developed as CLDAP [You95] which may offer faster access in some circumstances.

2.9.8 Facilities Not Provided by These Systems

Some useful matching facilities provided by non-directory systems are not provided by any of the systems reviewed in this section. I did a brief study of nine different on-line bibliographic catalogue systems as these systems all offer look-ups by author name. I found that for seven of the nine systems, if an author name could not be found, the system showed the user the names lexically surrounding the point in the author name index where the user's input would have occurred. Maybe this facility could be provided using extensible matching? I think it would be useful.

Another type of matching, sometimes used in text editors and word processing software, is for a system to match a user's input character by character: in many cases the target word/name can be uniquely identified before all the input has been entered. It is hard to see how this sort of facility could be provided efficiently using X.500's query-response protocol.

2.9.9 Summary

Although there are many differences in detail between the various directory service systems, my view is that the similarities considerably outweigh the differences. Consequently, much of this thesis is relevant to these other directory service systems.

2.10 Conclusions

In this chapter we have considered the tools for constructing an algorithm to find entries in an X.500 directory. We noted that there are two main querying abstractions: browsing and name input driven. While browsing has some appeal, it has some key limitations: the main problem is
that it does not work well if the name space is flat and populous, as a user may have to browse through thousands of entries. Therefore most directory services use a strategy based on users entering query names. This type of querying strategy depends on three things:

• the user input;
• the names in the directory;
• how user input is matched with names in the directory.

We can only sensibly discuss a matching algorithm if we understand the name forms we are trying to match: we discuss user input in Chapter 3 and directory names in Chapter 4.

There are several aspects to matching input to directory names. The main component is the range of search facilities provided by X.500. These include the various types of matching, Boolean search filters, and other parameters such as service controls that influence which entries, and what part of them, are returned to the user.

Another important aspect to be considered is the role of the DUA. For example, a DUA may modify the form of a user's input, or even ignore some of it, if the DUA believes that this will help the query to be answered successfully. It may also process any results received to provide facilities that are not provided directly: one example of this is sorting the results. We examine name matching in detail in Chapter 5. We look in detail at approximate matching in Chapter 6.

A third influence on the matching algorithm is the directory environment. We can anticipate a directory that is built from heterogeneous software distributed across an environment that encompasses fast and slow networks, well managed and poorly managed DSAs, data that is openly available and some that is closely access controlled, and so on. Furthermore, we might expect that exact matching and substring matching, for example, would typically have different response times. We will examine response time issues in the current NameFLOW-Paradise environment in Appendix G.
Chapter 3

User Input

3.1 Introduction

The purpose of this chapter is gain an understanding of what people enter in their queries. There are two sources of motivation for such a study. First, a study is required to inform those who wish to write DUAs about the nature of user input. Since DE provides a very general abstraction of a directory service, it is believed that the findings here should be applicable even to non-X.500 services.

The second motivation for this study is to provide background for my own work on matching algorithms which I describe in Chapter 5. A clear understanding of user input coupled with a knowledge of how entries are named in the DIT provides the basis for developing successful search algorithms.

This chapter seeks to do the following:

• To determine the basic form of each input field. For example, if we consider organisation names, we are interested whether users prefer to enter full organisation names, abbreviated names or sets of initials; for people's names we are interested whether users generally enter surnames only, initial(s) and surname, forename(s) and surname, or possibly a uid.

• To determine how closely the user input corresponds to the RDNs of the directory entries.

• To determine the frequency of particular common formats, such as “University of Anyplace” or “Anyplace University” for organisations.

• To look for potential matching problems, caused by various styles of input, and to determine how often these potential problems arise.

• To examine how often spelling mistakes occur.

The structure of this chapter is as follows. First, in Section 3.2 there is a brief review of the style of input required by various other directory systems. This section also reviews the input style for author names required by a number of library systems. Finally, there is a discussion of the format of electronic mail names, since I suspect that the format of these widely used names must have an impact on how directory users formulate their person name queries.

The main analysis in this chapter is based on a study of four sources of directory service user input. These sources, and the methodology of the data gathering, are described in Section 3.3.

The input is analysed type by type in Sections 3.4 to 3.7. There is also some analysis of UFN input in Section 3.8. Other lessons from the analysis are discussed in Section 3.9.

A number of comments on and caveats concerning the methodology are discussed in Section 3.10.
There is a summary of the results of the query analysis and conclusions in Section 3.11.

3.2 Influences on User Input

In this section I briefly review the style of input required by other directory systems that have been described in the literature or widely deployed. The interest here is twofold. First, it is instructive to see what other system designers have considered to be useful query formats. Second, we must be aware of the influence of previously deployed systems on users' perceptions of query formats. I have also looked at the way that library systems handle bibliographic searches based on author name, since this type of search is essentially the same as people name look-ups in a white pages directory. Finally I summarise electronic mail name formats. Electronic mail is used so much that mail names probably have an influence on the names we use when referring to others.

3.2.1 A Review of Other Systems

I now describe some other directory service systems that are described in the literature. Some of these systems have been widely deployed: this is the case for Whois, Finger, CCSO and Netfind. The reader should note that there is sometimes a distinction between what is described in the papers and what is implemented. In some cases, implementations seem to go further than the basic specification: this is true for Finger. In other cases, for example the CCSO Nameserver, the implementation I tried did not support all the options described. I have highlighted these differences where they occur.

Whois

The Whois specification [FHS85] allows a variety of query formats. The two main forms are surname-only and surname-comma-forename. An initial can be specified instead of the forename. Space characters around the comma can be given or omitted. Users can abbreviate queries by specifying leading characters followed by an elipsis: e.g. "Bark..." matches any entry where the surname starts with those letters.

It is also possible to do searches based on a mailbox by specifying an at-sign in the query: e.g. “paulb®” would look for entries with a mail name of “paulb” in any mail domain. Entries also have an associated handle, an alpha-numeric code; queries that should be matched against handles only should have a preceding exclamation mark, as in “!pau38”.

CCSO Nameserver

The CCSO nameserver [HDP96] package produced by the University of Illinois [WG93] includes a simple user agent called ph. This program allows considerable flexibility of format. In general a user can enter as many name tokens as he/she wants: a matched entry must include all these names. The order of the tokens is irrelevant. Thus, “Paul Barker” and “Barker Paul” are
3.2. INFLUENCES ON USER INPUT

equivalent. Single token queries are also acceptable: these can be either forenames or surnames. However queries including forenames only work if the database holds the forename as part of the name; similar arguments apply to queries including initials. There is no automatic support for matching initials and forenames, although the user can achieve this type of matching, as CCSO also supports the use of the "*" and "?" characters to match several characters or one character respectively. It is possible to specify alternative attributes on which to search other than names and nicknames.

**Finger**

The Finger protocol [Zim91] specifies a very simple querying protocol. Person name look-ups are allowed on username only, although a hostname can optionally be supplied to identify a remote user. The full form is thus "username@hostname". In practice, implementations may be more flexible than only allowing usernames. The finger provided with SUNOS v4.1.3 allows the name token to be any one of: a surname, a forename, or a username. A user can indicate that he/she wants the search to be usernames only.

**Netfind**

Like Finger, Netfind [ST91] only allows single tokens for personal name queries. As for Finger, the token may be a surname, a forename or a username. Netfind does not use an "@" to distinguish between username and organisation name information; the first token is the username and subsequent tokens are all treated as organisation or location information.

**Profile**

The paper on the Profile system [Pet88] describes a set of facilities that can be used to construct user interfaces, rather than the capabilities of an implementation. However, the paper suggests person name queries of the form forename-surname; this form can be extended by appending "@ site" to cover remote databases. The basic style of searching is using untagged queries, but the user has the option of specifying the attributes that should be used for searching. Profile also supports the use of the "*" and "?" characters to match several characters or one character respectively.

**CSNET**

CSNET [SLN82] uses a "username@host" form to uniquely identify an entry. Users may also search using surname only. Forename(s) and surname queries are also allowed; the order of the names is irrelevant. Names in queries may be specified as mandatory or optional: mandatory names must be in the required entry; optional names may be used to disambiguate between multiple alternative matches. The asterisk character may be used for wild-card matching.
WhoWhere

The WhoWhere service [WW96], designed for email address look-ups, allows users to format their queries as they wish: the system explicitly copes with *surname first* and *surname last* name orders.

Bickel

Bickel does not explicitly discuss user input in his paper [Bic87] on a new approximate matching algorithm, but illustrates his paper with names in two formats. These are *forename-surname* and *surname-comma-forename-middleInitial*; e.g. “robert jones” and “jones, robert a.”.

Afifi and Huitema

Afifi and Huitema [AH92] also do not explicitly discuss user input. However, their examples are based on surname-only queries. They also mention the possibility that a user might enter a query of the form *forename-surname*: e.g. “michel dupont”.

Summary

The overall picture is that there is no consistent approach. Some systems favour computer username look-ups, while others do not support them at all. Most systems support some form of wild-carding. Most systems (Netfind is an exception) support queries that resemble email addresses, and use the at-sign to differentiate between the person name and the organisation name part of the query.

3.2.2 Bibliographic Databases

Bibliographic databases are similar to white pages directories in that one form of querying for a monograph or journal article is for a user to supply an author name. In fact, bibliographic databases are typically larger than white pages databases; most on-line UK university bibliographic databases have hundreds of thousands of entries [Sto91]. As the querying problem is essentially similar to white pages searching, it is interesting to note the querying formats supported by such systems.

I examined nine distinct bibliographic systems used by universities in the UK. These were the systems for the universities of Brighton, Cambridge, Glasgow, Kent, Southampton, Sussex, UCL, Warwick and York. In all cases, the systems were prescriptive on the format of author names. Some were very rigid in what was acceptable, others allowed some flexibility of format. The instructions on the supported formats were usually very clear, often reinforced by examples. The following summarises the nine systems:

- Seven of the nine systems asked for the author name as input to a single prompt; two systems offered a surname prompt first, followed by a prompt for forenames and/or initials.
3.2. INFLUENCES ON USER INPUT

- All systems used surname first formats: e.g. "Barker, P" rather than "P Barker".
- Three systems suggested, amongst others, a surname-only format
- All systems allowed entry of a surname and initials, while six systems allowed forenames as well as initials.
- Of the seven systems where the name was entered at a single prompt, six required a comma immediately following the surname if initials or forenames were entered; the other system used a space character as a separator.
- Of the seven systems where the name was entered at a single prompt, some insisted on a space between initials, some insisted that there was no space, some accepted both formats.
- One system allowed two or more initials to be concatenated into a single token: e.g. "Thribb, E J", rather than "Thribb, E J".
- One system gave clear instructions on omitting name parts such as "de" or "von".

3.2.3 Email names

It is reasonable to expect that the format of electronic mail names has some influence on the format of queries, since electronic mail names have such a high profile in inter-personal communication. I was interested to see whether electronic mail names were largely based on users' names, or on identifiers not closely related to users' names.

I analysed the membership of a large distribution list, <lists-link@mailbase.ac.uk>, to determine the format of typical electronic mail names. The analysis is broad brush, and only intended to give a feel for the prevalent formats.

The list is a discussion group for UK librarians. At the time I analysed the membership, it had 1861 members, of which 71 were for non-human users: these were either distribution lists or pseudo-users (61 members had the mail name "library"). Excluding the obvious non-human members resulted in a sample of 1790 members. The breakdown of the main categories of electronic mail name is shown in Table 3.1. The categories are defined by illustrative electronic mail names, premised on a user name of "Alan Bruce Smith". Components in square brackets are optional.

Almost 40% of all electronic mail names were multi-part and based on the user's name: initial(s) or forename, followed by a dot, and then by the surname. This type of mail name closely resembles directory names. Two less usual formats, nevertheless together comprising over 5% of all mail names, included either a hyphen or an underscore character, usually as a separator between the initials and the surname. The single most common form (almost a quarter of all input) was a userid containing numeric characters. Sometimes these identifiers contained the user's initials, or leading characters from the surname, sometimes not. There were several other forms based on combinations of initials and surnames, or just initials.
3.2.4 Free-form or Stringent Name Formats

The stringent formats used by library systems are well-suited to systems where the format of names in the database is controlled by a single authority. Furthermore the library system administrators know the types of search that are optimised with indexes. It is arguable whether white pages DUAs should also insist on strict formats, or whether they should allow users to type what they will, albeit with some guidance available from a help system. An argument for strict formats is that this obviates the need for interpreting the tokens of user input into the most tractable forms: e.g. recognising that “barker, paul” is semantically equivalent to “paul barker”, and re-ordering surname-first queries before trying to match them. An argument against insisting on fixed formats is that there is no common format for entry names in the directory due to its distributed management. This suggests that even if query formats are fixed, a DUA will still need to do some transformation of input in order to match the variant name forms in the directory. There is also the aesthetic consideration that it seems wrong to the author, given the power of modern computers, to have to strait-jacket users; the directory system should be clever enough to find entries even when name formats are not closely aligned.

3.3 The Query Data

The queries analysed in this chapter are taken from the logs of four instances of the DE DUA. In each case the DUA is used to provide a real service. Furthermore, each service could be considered to be mature at the time the logs were taken, having been running for months or even years. It is thus hoped that the logs should mostly be free of biases due to users experimenting with a
3.3. THE QUERY DATA

system to test its capabilities.

Inevitably, there will be some small biases. Some usage will have been experimental; the user population for any service evolves over time, and new users will familiarise themselves with the features of, what is to them, a new service.

One factor which minimises any bias introduced by experimental usage is that only the first query of each user session is considered. This is for two reasons. First, taking one query per session neutralises any influence on the analysis caused by users having long, experimental sessions, which are likely to be atypical of normal directory service usage. A second reason for not considering follow-up queries is that they may not consist entirely of the user's guesses at names. Instead they may be a mixture of new user input, input from previous queries (which are accepted as default values for subsequent queries), and names selected from lists of entries displayed to the user.

The intention of this approach is that I am measuring, as much as possible, the intuitive input of users to a directory service. Clearly some users will have modified the style of their input as they have discovered through repeated querying what works best. Limitations of the approach are discussed further in Section 3.10.

The query data was gathered from logging four separate DUA services, all using the BE user interface. The four services are offered to different user communities.

First, the PARADISE project [Goo91], latterly NameFLOW-Paradise, runs a DUA. Although X.500 follows the client-server model, many sites do not as yet run X.500 clients. The primary intention behind the PARADISE service was to ensure that these sites have some level of access to the X.500 directory, although many others use the service because the high level of data replication used in the PARADISE DSAs and good communications links mean that PARADISE offers an above average quality of service. An unusual feature of the PARADISE service is that no initial default values are offered for any of the input fields. This is due to the international nature of the service; there is no common context for queries. While this is atypical, it offers the benefit for my purpose here that users have to enter a name at all prompts. The logs were taken from randomly selected days in the period from January to May 1994. At the time the logs were taken, the service was used quite heavily at about 600 queries per day. Access to the service is anonymous and so it is not possible to identify individual users. Although the PARADISE service was aimed at users from all countries, particularly European countries, in practice the vast majority of usage was by users from the US (approximately 60%) and the UK (approximately 35%).

The second source of query data is the DE user agent run by ULCC on behalf of UKERNA for the UK academic community. Whereas the PARADISE service is aimed at an international user base, the target community for the ULCC service are users of JANET. Much of the access to this user interface comes through the National Information on Software and Services (NISS)

\[1\text{ Estimates provided by PARADISE service manager}\]
server at the University of Bath; NISS provides an entry point to a variety of networked services. Since this service is a national service, the country name field is defaulted to “GB”. The logs were taken from various days in late 1994 and early 1995. Service usage averages between 50 and 100 queries per day. As for the PARADISE service, access to the service is anonymous. This data source is categorised as “ULCC”.

The third source of query data is the DE run by UCL’s Information Systems Division. This service is only accessible by UCL’s staff and students. Due to the principle of locality, the majority of queries are for people within the college and so both the country name and organisation name fields are defaulted to “GB” and “University College London” respectively. The logs are for service usage between July and December 1994. Usage averaged at about 10 queries per day. Although service access is not anonymous, the logging for this service does not record users’ names. This data source is categorised as “UCL”.

The fourth source of query data is the version of DE run by the Computer Science department at UCL. This is for Computer Science staff and students only. This group of users should be the most computer literate of the four user groups. The defaults for country name and organisation name are set to “GB” and “University College London” respectively. The logs are for service usage between August 1993 and November 1994. Average usage was seven queries per day. The service was used by 222 individual users. This data source is categorised as “UCL-CS”.

Users access these services in slightly different ways. The PARADISE and ULCC services are available using rlogin, telnet or X.29 remote login protocols over WANs, whereas the UCL and UCL-CS services are accessed by users running the DE program directly on the host on which they are logged on. WAN access is likely to cause more problems using DE for two reasons. First, remote login protocols require users to set-up their terminal emulations correctly. This is not always done. One university systems administrator told the author that many users are unfamiliar with the notion of a terminal type and, if asked to provide one, often type the name of the manufacturer of their monitor or keyboard [Lea91]. Although DE works satisfactorily in dumb terminal mode, users may experience difficulties if they select the wrong terminal type.

Second, connections over WANs may suffer from long delays in getting their input echoed. The author’s experience is that this can make correcting typing errors difficult. These two factors may account for a proportion of spelling errors or garbled input with the PARADISE and ULCC services.

### 3.4 Country names

All four sources of query data were analysed for country name input. The PARADISE query data consists purely of user input, since there is no default country name; the other three sources all offer “GB” as the default country name.

The four sources of data are broadly categorised in Table 3.2.
3.4. COUNTRY NAMES

<table>
<thead>
<tr>
<th>Source of queries</th>
<th>Sample size</th>
<th>Different names</th>
<th>Names occurring 10+ times</th>
<th>Names occurring once</th>
<th>% of input as def. value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARA</td>
<td>2865</td>
<td>214</td>
<td>23</td>
<td>130</td>
<td>n/a</td>
</tr>
<tr>
<td>ULCC</td>
<td>5958</td>
<td>236</td>
<td>31</td>
<td>130</td>
<td>29.80</td>
</tr>
<tr>
<td>UCL</td>
<td>3108</td>
<td>77</td>
<td>13</td>
<td>38</td>
<td>73.68</td>
</tr>
<tr>
<td>UCL-CS</td>
<td>2075</td>
<td>69</td>
<td>10</td>
<td>34</td>
<td>76.05</td>
</tr>
</tbody>
</table>

Table 3.2: Basic characteristics of four sets of sample query data

The PARADISE and ULCC services both show far greater diversity of input. In part this appears to be due to the greater proportion of spurious input values, such as "ykjhjkiljkuy" and the like. However it is also due to the relatively tight focus of the UCL and UCL-CS services; in both these cases over 70% of all input was merely accepting the default value.

Although the PARADISE and ULCC services have quite diverse input, they are both dominated by very few items of input. For the PARADISE service, "usa" and "uk" occurred 880 and 539 times respectively, representing almost half (49.53%) of the sample. For the ULCC service, the most popular input was "gb" (as opposed to the default value of "GB" in capitals). Together the two "GBs" formed over half the input for the ULCC service.

The country name input was analysed according to the following categories:

Two letter codes: ISO 3166 defines a two-letter code for each country. The X.500 directory mandates that these codes are used as the Relative Distinguished Names (RDNs) of country entries.

Other short forms: This category includes all other short forms of country names of up to three alphabetic characters, and any other names that are sets of initials. This includes "usa", "u.s." (the RDN is simply "US" without the dots), "uk" (the RDN is "GB"), other short forms such as "ger" for Germany, and "ussr" for the former Soviet Union.

Longer abbreviated forms: This includes forms such as "austral", "brit" and "ital*" where the user has entered an abbreviated form of a country’s full name, usually sufficient to uniquely identify the intended country.

Full country names: This category includes commonly used longer forms of country names. It may include a number of forms for any one country. For example, for the USA it includes: “united states of america”, “united states” and “america”.

List option: The DE interface allows the user to enter an asterisk (wild-card) character to list country names, and then to select the required country from the list.

Other: This catch-all category includes several types of incorrect input including: place names,
organisation names, garbled user interface commands, rubbish such as "sdlff" plus some input where I was not sure what the user intended.

3.4.1 The Query Formats

Table 3.3 gives a breakdown of how the input matched these categories. The PARADISE analysis is markedly different from the other three sets of data, due to the absence of a default country name. However, almost 90% of the input was of a recognisable country name. The list option was used in 6% of queries, with over 3% of queries being rubbish or not a country name.

The other three data sources are broadly similar; in each case a default country name of "GB" is offered to the user, and this value predominates in each samples of queries. This defaulting has advantages and disadvantages. A plus point is that the queries reflect what happens in most services, with "local" queries predominant. A negative point is that since a substantial proportion of the input is merely a default being accepted, there is less country name input per se to analyse.

<table>
<thead>
<tr>
<th>Form of input</th>
<th>Each category as a percentage of all data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PARA</td>
</tr>
<tr>
<td>Two letter codes</td>
<td>12.01</td>
</tr>
<tr>
<td>Other short forms</td>
<td>51.59</td>
</tr>
<tr>
<td>Long abbrev forms</td>
<td>0.24</td>
</tr>
<tr>
<td>Full country names</td>
<td>26.01</td>
</tr>
<tr>
<td>List option</td>
<td>6.46</td>
</tr>
<tr>
<td>Other input</td>
<td>3.49</td>
</tr>
</tbody>
</table>

Table 3.3: Summary of all country name input analysed by form

The two letter country codes predominate in the ULCC, UCL and UCL-CS query sets, with 70+% of all input being an ISO 3166 code; this is hardly surprising since the default value of "GB" is an ISO code. The majority of the remainder of the input is divided between other short forms and full country names, with three or four per cent of users opting to list the country names.

In order to get a better understanding of what users actually enter if defaults are not selected, I repeated the analysis but excluded all instances of the default value. A note of caution must be registered here. It is possible that some of the instances of "GB" were in fact typed by users, rather than accepted as the default value. The logs do not allow me to detect how much this is the case. However, the logs reveal that users type capitals relatively infrequently. Rather than try to estimate the proportion of "GBs" that were actually typed, I have chosen to ignore this problem for the sake of simplicity. The results are presented in Table 3.4.

The picture that emerges from this analysis of country names typed by users is that two letter
codes and other short names are the two most popular forms of input, followed by full country name forms. The amount of abbreviated country name input is very small for all data sets.

The relative popularity of the two most favoured forms differs widely between the different sets of query data. Some of this discrepancy can be attributed to the high proportion of queries for GB and the US, since both these countries have popular short forms that are non-RDNs; “uk” and “usa” respectively. To see what happens for non-UK and non-US queries, I repeated the analysis using just the non-UK and non-US data. The results are shown in Table 3.5.

<table>
<thead>
<tr>
<th>Form of input</th>
<th>Each category as a percentage of country name data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PARA</td>
</tr>
<tr>
<td>Two letter codes</td>
<td>13.33</td>
</tr>
<tr>
<td>Other short forms</td>
<td>57.29</td>
</tr>
<tr>
<td>Long abbrev forms</td>
<td>0.27</td>
</tr>
<tr>
<td>Full country names</td>
<td>29.11</td>
</tr>
</tbody>
</table>

Table 3.4: Country name input excluding default values

While the relatively popularity of two letter codes and full country names differs for each data set, in all cases these two categories together form over 95% of country name queries.

Before moving on to examine some other aspects of the data, we can draw some initial conclusions.

- The UK and US are special cases, as both have widely used short forms that are not two-letter country codes. Such short forms are hardly used for other countries.

- There was considerably less use of two-letter codes in the PARADISE data than with the other three services, even for non-UK and non-US data. My belief is that the existence of the default value “GB” in the other three services suggested that this type of input was the most appropriate.
• Listing of country names was used infrequently for all services, and much less than for the other types of input: see Sections 3.5, 3.6 and 3.7. I would surmise that users generally feel confident about their country name input, and are thus not tempted to list the countries to see what is there.

The PARADISE service had the highest proportion of what I classed other input (3.49%), followed by the ULCC service (2.81%). In contrast both the UCL services had less than a quarter as much non-country name input. To some extent this may be explained by the use and acceptance of default values. All PARADISE queries were typed in; there was thus more scope for error. The PARADISE interface had more garbled input than the other sources of queries; my guess is that this may be partly explained by the difficulty of editing incorrect input over virtual terminal sessions on a congested network to a service run on what is often a very remote host.

The ULCC queries suffered from many users failing to interpret the typing instruction “<CR>” as meaning a RETURN character; nearly 1.5% of queries were affected this way. This problem occurred very little with the other data sources, either because the software was modified to detect this input (PARA), or simply because users did not make the mistake (UCL and UCL-CS).

### 3.4.2 Other Features of the Queries

#### Use of Dots with Initials

One area of interest is whether users enter country names that are initials with or without full-stops: e.g., do users generally enter “uk” or “u.k.”? This is of interest since a query string of “u.k.” does not exactly match a directory name of “uk”. Some evidence is presented in Table 3.6.

<table>
<thead>
<tr>
<th>Source of queries</th>
<th>No. queries with inits separated by dots</th>
<th>No of instances of “uk”</th>
<th>No of instances of “u.k.” or “u.k”</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARA</td>
<td>35</td>
<td>539</td>
<td>10</td>
</tr>
<tr>
<td>ULCC</td>
<td>22</td>
<td>345</td>
<td>8</td>
</tr>
<tr>
<td>UCL</td>
<td>2</td>
<td>149</td>
<td>1</td>
</tr>
<tr>
<td>UCL-CS</td>
<td>3</td>
<td>98</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 3.6: Country names and dots in input

Examination of the data revealed that dots were only ever used with name forms for the UK and the US, and that they were not used very often even for those countries. Table 3.6 show how often this name form was used. It also shows a comparison of how often users typed “uk” with how often they typed “u.k.” or “u.k”. The answer is that the form without the dots is preferred in over 98% of all cases.
3.4. **COUNTRY NAMES**

**Spelling Mistakes**

The proportion of spelling mistakes is calculated as a percentage of user entered names (defaults and non-country name input excluded) for each set of queries. The results are shown in Table 3.7.

<table>
<thead>
<tr>
<th>Source of queries</th>
<th>Sample size</th>
<th>Spelling mistakes as per cent of country name input</th>
<th>No. of spelling mistakes</th>
<th>Damerau errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARA</td>
<td>2580</td>
<td>0.39</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>ULCC</td>
<td>3792</td>
<td>1.00</td>
<td>38</td>
<td>30</td>
</tr>
<tr>
<td>UCL</td>
<td>818</td>
<td>0.49</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>UCL-CS</td>
<td>497</td>
<td>0.80</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 3.7: Country name input and spelling mistakes

Spelling mistakes were fairly infrequent for each service, with a maximum of 1.00%. This low rate of spelling errors probably reflects the fact that country name input is on average quite short. Nearly all spelling input could be classed as "Damerau errors": these are common typographical errors; a definition is given in Section 6.2.1.

**Mapping Country Names to RDNs**

We noted earlier that, aside from a few special cases of input such as "UK" and "USA", the majority of input was either a two-letter country code or a full country name. While one can argue that full country names are more intuitive to users, there are some advantages to the X.500 system if users provide two-letter codes, since these are the country entry RDNs. This allows some potential optimisation in querying strategies, since there is no need to resolve the country name with a directory operation.

There is also an advantage to users of using two letter codes: the two letter code is guaranteed to be in the directory entry; the longer friendly name form may not be. This is clearly of relevance to the international directory, where friendly country names are unlikely to exist for all languages for all country entries. Some evidence on this is presented in Chapter 4.

Fortunately we can usually have the best of both worlds, with users entering what they prefer while the directory system uses its preferred two letter codes. This is possible due to the skewed nature of the query data, with a few countries being queried frequently. In Table 3.8, we can see the effect of using a mapping table in a DUA to transform the most popular non-RDN country names to their RDN equivalents. For example, user input of "uk" could be transformed to "GB" by such a mapping table entry. Table 3.8 shows the effect of mapping tables of sizes five, ten and twenty entries.

The percentages of input that can be classed as RDN input is calculated as a proportion of
country name queries: i.e. commands to list entries and garbled queries are excluded. For the three sources where "GB" is offered as a default, the percentages are calculated for the input both including and excluding accepted default values.

<table>
<thead>
<tr>
<th>Source of Queries</th>
<th>Entries in mapping table</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>PARA</td>
<td>13.33</td>
</tr>
<tr>
<td>ULCC-all</td>
<td>74.92</td>
</tr>
<tr>
<td>ULCC-noGB</td>
<td>63.53</td>
</tr>
<tr>
<td>UCL-all</td>
<td>84.82</td>
</tr>
<tr>
<td>UCL-noGB</td>
<td>37.47</td>
</tr>
<tr>
<td>UCLCS-all</td>
<td>86.88</td>
</tr>
<tr>
<td>UCLCS-noGB</td>
<td>38.26</td>
</tr>
</tbody>
</table>

Table 3.8: Percentage of country name queries that are RDNs with various sizes of name-to-RDN mapping tables

The results show that even a name-to-RDN mapping table of five entries for an organisational or departmental service would allow over 95% of all country name input to be regarded as an RDN. Less focused services, such as national or international servers, would need mapping tables of ten or twenty entries to achieve the same proportion of RDNs. A ten entry mapping table appropriate to the UCL and UCL-CS query data is shown in Appendix B.

3.5 Organisation Names

While I analysed all country name data, I decided to restrict my analysis of organisation name data to that for UK queries. The reasons for doing this are pragmatic. First, there are far more organisation names than country names, and the process of analysing names is very labour intensive. Second, my lack of familiarity with non-UK organisation names means that I could not classify them as well as I can UK organisation names. While this decision to restrict the analysis to UK names allows more informed judgements to be made, there is the attendant risk that the UK data is unrepresentative of query data as a whole.

All four sources of query data were analysed for organisation name input. UK entries formed about one third of the PARADISE data. The ULCC data was a smaller set than that analysed for country names; the original set was unmanageably large for the detailed line by line analysis required. About 60% of all ULCC input is for UK organisations. Neither the PARADISE nor the ULCC services offered a default organisation name.

In contrast, both the UCL and UCL-CS query sets had "University College London" as a default value. Approximately 85% of both the UCL and UCL-CS input were for UK organisations.
The four sources of organisation query data are broadly categorised in Table 3.9.

<table>
<thead>
<tr>
<th>Source of queries</th>
<th>Sample size</th>
<th>Different names</th>
<th>Names occurring 10+ times</th>
<th>% of input as def. value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARA</td>
<td>887</td>
<td>390</td>
<td>7</td>
<td>n/a</td>
</tr>
<tr>
<td>ULCC</td>
<td>2488</td>
<td>864</td>
<td>45</td>
<td>n/a</td>
</tr>
<tr>
<td>UCL</td>
<td>2581</td>
<td>312</td>
<td>12</td>
<td>56.92</td>
</tr>
<tr>
<td>UCL-CS</td>
<td>1738</td>
<td>226</td>
<td>10</td>
<td>52.47</td>
</tr>
</tbody>
</table>

Table 3.9: Basic characteristics of four sets of UK organisation query data

The local focus of the UCL services, along with the existence of a default value, resulted in less diversity of input for the UCL services than the PARADISE and ULCC services. Whereas the default value is accepted in over half the UCL and UCL-CS queries, the most common query in the PARADISE and ULCC services occurred less than 2% of the time.

The organisation name input was analysed according to the following categories:

**Full organisation names:** The user attempted to provide the full name of the organisation, such as “university of foo” or “foo university”.

**Other long names:** The user provided a considerable part of a name, but omitted a part of the name that would fully identify the organisation. In many of these cases users supplied just a town or city name, when it was clear that they meant the university of that town.

**Abbreviated names:** The user supplied an abbreviated name of an organisation, such as “camb” for “Cambridge University”.

**Initials:** The user used a form of organisation name consisting of initials: e.g. “ucl” for “University College London”.

**Domain names:** The user supplied a name that was clearly dependent on the user being familiar with Domain Name Server (DNS) names[Moc87a][Moc87b], although the format was not always precisely that of DNS names. Examples of names in this category are “glasgow.ac.uk” and “glasgow ac uk”.

**Non-specific:** The user has deliberately supplied non-specific input in order to match a number of possibilities. Examples include “univ”.

**List option:** The DE interface allows the user to enter an asterisk wild-card character to list organisation names, and then to select the required organisation from the list.

**Other input:** This catch-all category includes several types of input including: non-UK place and organisation names, garbled user interface commands, rubbish such as “sdfg”, plus some input where I was not sure what the user intended.
Table 3.10 gives a breakdown of how the input matched these categories.

<table>
<thead>
<tr>
<th>Form of input</th>
<th>Each category as a percentage of all data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PARA</td>
</tr>
<tr>
<td>Full org names</td>
<td>49.77</td>
</tr>
<tr>
<td>Long org names</td>
<td>23.48</td>
</tr>
<tr>
<td>Abbreviated names</td>
<td>2.71</td>
</tr>
<tr>
<td>Initials</td>
<td>8.01</td>
</tr>
<tr>
<td>Domain names</td>
<td>0.79</td>
</tr>
<tr>
<td>Non-specific</td>
<td>1.47</td>
</tr>
<tr>
<td>List option</td>
<td>11.85</td>
</tr>
<tr>
<td>Other input</td>
<td>1.92</td>
</tr>
</tbody>
</table>

Table 3.10: Organisation name input analysed by form

The results show remarkably consistent patterns. The two services not offering a default organisation name have very similar results, while the two UCL-based services are also similar to one another. For all the services, at least 70% of input (defaults included) was a long form of organisation name. Organisation name initials were used for between five and ten per cent of queries. Users listed organisation names in more than 10% of all queries. This was much more frequently than they listed country names.

To make the analysis of the four data sets more comparable, I removed all instances of querying UCL from the two sets of UCL data. The percentages of the different query formats are shown in Table 3.11 as relative frequencies of the five reasonable query formats.

<table>
<thead>
<tr>
<th>Form of input</th>
<th>Each category as a percentage of organisation name data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PARA</td>
</tr>
<tr>
<td>Full org names</td>
<td>58.72</td>
</tr>
<tr>
<td>Long org names</td>
<td>27.70</td>
</tr>
<tr>
<td>Abbreviated names</td>
<td>3.20</td>
</tr>
<tr>
<td>Initials</td>
<td>9.45</td>
</tr>
<tr>
<td>Domain names</td>
<td>0.93</td>
</tr>
</tbody>
</table>

Table 3.11: Organisation name input excluding default values

The results in Table 3.11 show a very clear pattern. Between 50% and 60% of all input was of a full organisation name. Between 25% and 35% for each query set was of a long form;
3.5. ORGANISATION NAMES

together these longer forms accounted for between 85% and 90% of all input data for each query set. Use of initials was in all cases very close to 10%. Usage of abbreviated organisation names and domain names was very low.

3.5.1 Other Features of Organisation Name Queries

Very Short Input

There were 462 queries of two or three characters. Of these, the vast majority – 428 (92.64%) – were initials, with the remainder being abbreviated names such as "cam" for "Cambridge".

Use of Dots with Initials and Abbreviations

With organisation names we are interested to see whether users enter names of the form "ucl" or "u.c.l.". Another area of interest is whether abbreviated names have dots appended. Some evidence is presented in Table 3.12.

<table>
<thead>
<tr>
<th>Source of queries</th>
<th>No. queries with inits separated by dots</th>
<th>No of instances of 'st'</th>
<th>No of instances of 'st.'</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARA</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ULCC</td>
<td>7</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>UCL</td>
<td>2</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>UCL-CS</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3.12: Organisation names including dots

Analysis of the data revealed that names with dots were very uncommon. For example, in the UCL data, there were 97 instances of the string "ucl" and one instance of "u.c.l". The only case where dots were used relatively frequently was with "st.", the abbreviation for saint. Even then, users more often than not omitted the dot.

Spelling Mistakes

The proportion of spelling mistakes is calculated as a percentage of user entered names (defaults and non-organisation name input excluded) for each set of queries. The results are shown in Table 3.13.

Spelling mistakes occurred much more frequently in organisation name input than with country names; the average error rate for the four sources of data was 3.29%. The majority of the spelling mistakes appeared to be typographical errors, rather than cognitive errors; one can surmise this on the basis that many of misspellings were of the word "university" rather than the
CHAPTER 3. USER INPUT

<table>
<thead>
<tr>
<th>Source of queries</th>
<th>Spell mistakes as per cent</th>
<th>No. spelling mistakes</th>
<th>Damereau errors No. %</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARA</td>
<td>2.66</td>
<td>20</td>
<td>17</td>
</tr>
<tr>
<td>ULCC</td>
<td>4.23</td>
<td>86</td>
<td>85</td>
</tr>
<tr>
<td>UCL</td>
<td>2.66</td>
<td>23</td>
<td>20</td>
</tr>
<tr>
<td>UCL-CS</td>
<td>1.65</td>
<td>9</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 3.13: Organisation name spelling mistakes

associated city name. Whatever the reason for the spelling error, more than 90% of spelling mistakes could be classed as Damereau errors.

One reason for the higher proportion of misspelled organisation name input is that organisation names are much longer than country names. Furthermore, we can see from the results in Table 3.14 that misspellings tend to occur in longer organisation names.

Table 3.14: Organisation name spelling mistakes and length of input

Table 3.15 shows an analysis of which character position contained the first misspelled character in misspelled words in organisation names. Note that we are measuring the position within the misspelled word, and not within the entire input string.

Only 18% of misspellings occur in the first three characters and less than a third in the first four characters. We can exploit this fact in matching algorithms as we can remove the effect of many spelling mistakes by truncating input words to a few characters.

"University of Anyplace" or "Anyplace University"

The query data is heavily biased to the academic community. Many of the queries contain the word "university" or an abbreviated form of it. What are the predominant forms? Some results are presented in Table 3.16.

The two most common forms of "university of anyplace" and "anyplace university" are almost equally favoured by users; unfortunately they do not always guess the correct one for a given institution. If we consider just the queries including the tokens "uni", "univ" or "university", 6%
### 3.5. ORGANISATION NAMES

<table>
<thead>
<tr>
<th>Char pos. of spelling mistake</th>
<th>No. of spelling mistakes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>19</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
</tr>
<tr>
<td>6</td>
<td>17</td>
</tr>
<tr>
<td>7</td>
<td>29</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>138</strong></td>
</tr>
</tbody>
</table>

Table 3.15: Character position of first incorrect character in misspelled organisation name input

<table>
<thead>
<tr>
<th>Source of queries</th>
<th>Common forms of university query</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>university of anyplace</td>
</tr>
<tr>
<td>PARA</td>
<td>146</td>
</tr>
<tr>
<td>ULCC</td>
<td>445</td>
</tr>
<tr>
<td>UCL</td>
<td>102</td>
</tr>
<tr>
<td>UCL-CS</td>
<td>67</td>
</tr>
</tbody>
</table>

Table 3.16: Form of university queries with "university" or "uni" in the name
were of one of the abbreviated forms. This has implications for matching; although “anyplace uni” is a substring of “anyplace university”, “uni of anyplace” is not a substring of “university of anyplace”. We will pursue this issue in Chapter 5 on matching.

**Punctuation and Stop List Words**

Punctuation and stop list words, semantically of little value, can inhibit matching if they are wrongly omitted or included by the user. For example, is the directory name “Heriot Watt University” or “Heriot-Watt University”? Alternatively, is the user likely to guess the name “University of Technology, Loughborough”? If the user omits the comma, exact or substring matching will not succeed. Table 3.17 shows how frequently these punctuation characters and stop list words occur.

<table>
<thead>
<tr>
<th>Source of queries</th>
<th>the...</th>
<th>...and...</th>
<th>...of...</th>
<th>...of... excluding University of apostrophe</th>
<th>hyphen</th>
<th>comma</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARA</td>
<td>3</td>
<td>5</td>
<td>170</td>
<td>24</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>ULCC</td>
<td>8</td>
<td>18</td>
<td>512</td>
<td>59</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>UCL</td>
<td>0</td>
<td>11</td>
<td>138</td>
<td>36</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>UCL-CS</td>
<td>3</td>
<td>6</td>
<td>79</td>
<td>12</td>
<td>9</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 3.17: Queries containing stop list words and punctuation

Apart from the word “of” which occurs chiefly in the phrase “university of”, none of the punctuation characters or stop list words occurs very frequently. Taken together, even excluding all cases of “uni(versity) of”, queries with punctuation and stop list words form 2.67% of all organisation input.

**User Input and DNS Names**

I examined the extent to which users’ input corresponded to DNS names; a very close correlation would indicate that search filters based on domain names might be effective. I tested this relationship for all single token queries which were one of the organisation name forms, where the organisation queried was in the UK academic community. The percentage of such queries that were DNS names is shown in Table 3.18.

The results show that as many as 80% to 90% of single token queries may be DNS names. Matching on DNS names thus appears as though it may be a useful option to consider.
3.5. ORGANISATION NAMES

<table>
<thead>
<tr>
<th>Source of queries</th>
<th>Percentage of single token queries that are DNS names</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARA</td>
<td>80.94</td>
</tr>
<tr>
<td>ULCC</td>
<td>84.25</td>
</tr>
<tr>
<td>UCL</td>
<td>87.36</td>
</tr>
<tr>
<td>UCL-CS</td>
<td>78.23</td>
</tr>
</tbody>
</table>

Table 3.18: Percentage of single token queries organisation name queries that are DNS names

How Often is User Input the Organisation RDN?

Querying algorithms can be simpler, and probably more efficient too, if users enter organisation names that are exactly the same as the RDNs in the directory. How often is this the case? The results of some analysis on the query data are presented in Table 3.19. The percentages are calculated first, for just those queries where the target entry was in the directory, and second, as a percentage of all queries. For the two UCL data sets, I repeated the calculations excluding all local queries: these percentages are then directly comparable with the results for the PARADISE and ULCC data sets.

<table>
<thead>
<tr>
<th>Source of queries</th>
<th>Percentage of queries that are RDNs for ...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>queries where org in directory</td>
</tr>
<tr>
<td>PARA</td>
<td>29.56</td>
</tr>
<tr>
<td>ULCC</td>
<td>34.28</td>
</tr>
<tr>
<td>UCL</td>
<td>72.45</td>
</tr>
<tr>
<td>UCL (excluding UCL queries)</td>
<td>25.98</td>
</tr>
<tr>
<td>UCL-CS</td>
<td>77.05</td>
</tr>
<tr>
<td>UCL-CS (excluding UCL queries)</td>
<td>39.34</td>
</tr>
</tbody>
</table>

Table 3.19: Percentage of organisation queries that are RDNs

The user enters an organisation name equivalent to the RDN in between a quarter and a third of queries when there is no default organisation. If we consider organisational services (UCL and UCL-CS), where the majority of the queries are for within the local organisation, the proportion is between 70% and 80%. In fact, we can do better than this if we use a name-to-RDN mapping table as described for country names. The effect of using such a mapping table is shown in Table 3.20.

The results show that with mapping tables of twenty entries, nearly 90% of UCL-CS queries
could be treated as RDNs; the figure is over 85% for queries to the UCL service. If resolving organisation names proved to be a costly process, this mapping table technique can clearly help with response times and reducing the load on the directory. A ten entry mapping table appropriate to the UCL and UCL-CS query data is shown in Appendix B.

The reader should remember that these figures are specific for queries within the UK. The percentage of queries mapped to RDNs would be lower when considering all queries. Clearly the technique could be extended to organisations from other countries if they were queried frequently enough to make such a step worthwhile.

### 3.6 Department names

As for organisation names, the analysis of department name input is restricted to queries within UK organisations. Again the reasons are pragmatic: to reduce the volume of the data to be analysed, and to make more informed judgements about the nature of the input.

The query data sets are the same as those used for the organisation name analysis. The UCL and UCL-CS services were mostly for queries within UCL; some further analysis is made of these two data sets for queries within UCL to give an indication of how closely user input corresponds to directory names.

The four sources of organisation query data are broadly categorised in Table 3.21.

<table>
<thead>
<tr>
<th>Source of queries</th>
<th>Sample size</th>
<th>Different names</th>
<th>Names occurring 10+ times</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARA</td>
<td>887</td>
<td>163</td>
<td>5</td>
</tr>
<tr>
<td>ULCC</td>
<td>2488</td>
<td>380</td>
<td>17</td>
</tr>
<tr>
<td>UCL</td>
<td>2581</td>
<td>321</td>
<td>24</td>
</tr>
<tr>
<td>UCL-CS</td>
<td>1738</td>
<td>193</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 3.21: Basic characteristics of four sets of UK organisation query data

I analysed the department name input according to the following categories:
3.6. **DEPARTMENT NAMES**

**Full names:** The user entered a full department name such as “chemistry”, “physics” or “department of physics”.

**Abbreviated names:** The user entered some abbreviated form such as “comp sci” or “elec eng”, sometimes with explicit wild-card characters.

**Initials:** The user entered a set of initials only, such as “cs” for Computer Science.

**Null entry:** DE allows users to omit a department name if they wish.

**List option:** The DE interface allows the user to enter an asterisk wild-card character to list department names, and then to select the required department from the list.

**Interface commands:** The user’s input was intended to be a user interface command, but was mal-formed and treated as a query.

**Other input:** This catch-all category includes several types of erroneous input including: organisation names, person names, dot-separated domain names and rubbish such as “asdf”.

Table 3.22 gives a breakdown of how the input matched these categories.

<table>
<thead>
<tr>
<th>Form of input</th>
<th>Each category as a percentage of all data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PARA</td>
</tr>
<tr>
<td>Full dept names</td>
<td>29.09</td>
</tr>
<tr>
<td>Abbreviated names</td>
<td>1.58</td>
</tr>
<tr>
<td>Initials</td>
<td>1.35</td>
</tr>
<tr>
<td>Null entry</td>
<td>45.55</td>
</tr>
<tr>
<td>List option</td>
<td>18.26</td>
</tr>
<tr>
<td>Interface comms</td>
<td>1.80</td>
</tr>
<tr>
<td>Other input</td>
<td>2.37</td>
</tr>
</tbody>
</table>

Table 3.22: Department name input analysed by form

The most striking feature of the data is that users' top preference is to omit the department name; on average the department name is omitted in more than four out of ten queries. UCL users entered a department name slightly more often than users of the PARADISE and ULCC services. Listing department names was also a popular option, occurring in between 10% and 23% of queries. If users entered department names, they usually (about 90% of the time for the PARADISE, ULCC and UCL services) entered a full name. The UCL-CS service differed in that there was a higher proportion of initials used; this can be explained solely by UCL-CS users entering “cs” for their own department.
CHAPTER 3. USER INPUT

<table>
<thead>
<tr>
<th>Form of input</th>
<th>Each category as a percentage of all data</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UCL</td>
<td>UCL-CS</td>
<td></td>
</tr>
<tr>
<td>Full dept names</td>
<td>45.06</td>
<td>35.10</td>
<td></td>
</tr>
<tr>
<td>Abbreviated names</td>
<td>2.61</td>
<td>3.02</td>
<td></td>
</tr>
<tr>
<td>Initials</td>
<td>3.06</td>
<td>11.46</td>
<td></td>
</tr>
<tr>
<td>Null entry</td>
<td>43.02</td>
<td>33.75</td>
<td></td>
</tr>
<tr>
<td>List option</td>
<td>5.74</td>
<td>15.62</td>
<td></td>
</tr>
<tr>
<td>Interface comms</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>Other input</td>
<td>0.51</td>
<td>1.04</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.23: Department name input for UCL users looking for entries within UCL

I repeated the analysis for the UCL and UCL-CS queries, for just those queries that were for entries within UCL itself. The results are shown in Table 3.23.

The within-UCL queries show that users use full department names a little more often, and list departments a little less often when querying within their own organisation. In other respects, the within the organisation department queries were similar to department queries as a whole.

**Spelling Mistakes**

The proportion of spelling mistakes is calculated as a percentage of user entered names (non-department name input excluded) for each set of queries. The results are shown in Table 3.24.

<table>
<thead>
<tr>
<th>Source of queries</th>
<th>Spell mistakes as per cent</th>
<th>No. spelling mistakes</th>
<th>Damereau errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARA</td>
<td>1.41</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>ULCC</td>
<td>2.84</td>
<td>22</td>
<td>20</td>
</tr>
<tr>
<td>UCL</td>
<td>2.59</td>
<td>31</td>
<td>24</td>
</tr>
<tr>
<td>UCL-CS</td>
<td>3.89</td>
<td>26</td>
<td>23</td>
</tr>
</tbody>
</table>

Table 3.24: Department name spelling mistakes

Spelling mistakes occurred almost as often for department names as they did for organisation names; the average error rate was nearly 3%. Again the majority of the errors appeared to be typographical rather than cognitive. There were a few instances where the user input was an American spelling: e.g. "center" rather than "centre". As with organisation names, most of the errors could be classed as Damerau errors.

Table 3.25 shows an analysis of which character position contained the first misspelled char-
3.6. DEPARTMENT NAMES

<table>
<thead>
<tr>
<th>Character position of spelling mistake</th>
<th>No. of misspellings by position in word</th>
<th>No. of misspellings by position in input</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>11+</td>
<td>1</td>
<td>21</td>
</tr>
<tr>
<td>Total</td>
<td>83</td>
<td>83</td>
</tr>
</tbody>
</table>

Table 3.25: Character position of first incorrect character in misspelled department name input

It shows that approximately two-thirds of spelling mistakes occurred after the fourth character in any misspelled word, and over 72% after the fourth character in any line containing misspelled input. As for organisation names, this suggests that truncating input before matching is one way to solve the problem of matching misspelled input.

Use of the Words "Department", "School" and "Institute"

We saw in the section on organisation name input that a substantial proportion of organisation name queries contained the word "University", either as "university of anyplace" or "anyplace university". A similar phenomenon occurs with department name input; however it is less frequent than with organisation names, and it occurs with three words: "department", "school" and "institute". Table 3.26 shows how often these forms of name are used. Note that DST is used to mean any one of "department", "school" or "institute".

2.38% of those queries that were department names (either a full name, an abbreviated name or a set of initials) contained one of the three cited words (or an abbreviated form such as "dept"). There is no problem with this so long as users guess the right form where appropriate; otherwise these words may impair matching if needlessly or erroneously supplied. We will see if this appears to be a problem in Chapter 5 on matching.
CHAPTER 3. USER INPUT

<table>
<thead>
<tr>
<th>Source of queries</th>
<th>DST of Something</th>
<th>Something DST</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARA</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>ULCC</td>
<td>21</td>
<td>4</td>
</tr>
<tr>
<td>UCL</td>
<td>12</td>
<td>17</td>
</tr>
<tr>
<td>UCL-CS</td>
<td>9</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 3.26: Department name queries containing any of the words: Department, School, or Institute

Use of Dots with Initials and Abbreviations

There was very little use of dots either with initials or following abbreviations; five instances in almost three thousand queries where a department name was supplied.

Punctuation and Stop List Words

<table>
<thead>
<tr>
<th>Source of queries</th>
<th>Input includes ...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>the...</td>
</tr>
<tr>
<td>PARA</td>
<td>0</td>
</tr>
<tr>
<td>ULCC</td>
<td>2</td>
</tr>
<tr>
<td>UCL</td>
<td>0</td>
</tr>
<tr>
<td>UCL-CS</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3.27: Queries containing stop list words and punctuation

6.75% of department queries where the user entered a name included one or more of the stop list words or punctuation characters shown in Table 3.27. Much of this is attributable to the use of the word “and” in queries for combined departments, such as “Physics and Astronomy” or “Greek and Latin”.

How Often is User Input the Organisational Unit RDN?

As for country and organisation names, if users specify names in their queries that are the department name RDNs, querying algorithms are likely to be more efficient as the role of the directory in performing name resolution is reduced. I examined how closely users' input matched the corresponding RDNs for the UCL and UCL-CS services for queries within UCL. Table 3.28 shows the results of this analysis.

The user enters a department name equivalent to the RDN nearly half the time a name is entered; this corresponds to between a fifth and a quarter of all queries for people within UCL.
3.7 PERSON NAMES

<table>
<thead>
<tr>
<th>Source of queries</th>
<th>Percentage of queries that are RDNs for ...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>queries where org name specified</td>
</tr>
<tr>
<td></td>
<td>all queries</td>
</tr>
<tr>
<td>UCL</td>
<td>49.75</td>
</tr>
<tr>
<td>UCL-CS</td>
<td>40.51</td>
</tr>
</tbody>
</table>

Table 3.28: Proportion of queries where department name is same as RDN

As with country and organisation names we can use a mapping table to improve this degree of correspondence. The results of using a mapping table are shown in Table 3.29.

<table>
<thead>
<tr>
<th>Source of Queries</th>
<th>Entries in mapping table</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>UCL</td>
<td>49.75</td>
</tr>
<tr>
<td>UCL-CS</td>
<td>40.51</td>
</tr>
</tbody>
</table>

Table 3.29: Percentage of queries that are RDNs with various sizes of department name-to-RDN mapping tables

The results show that with mapping tables of ten to twenty entries, 75% or more name queries can be treated as RDNs. Once again a small mapping table can be used to do a lot of query resolution. The reader should remember that the results of the technique described here are only applicable to local queries. However, since local queries predominate, this is a minor limitation.

3.7 Person names

One option, possibly the “obvious” one, would have been to have analysed the same basic query data as for the previous two sections. I have chosen not to do this for several reasons. First, the variety of human names is far greater than for the other categories. Very few of the names in the input occur more than once; consequently the task of analysis is harder as more individual names have to be assessed.

Second, some of the analysis has required close examination of UCL’s database in order to deduce users’ intentions. Such comprehensive analysis was made feasible by working with a smaller set of UCL query data.

Third, I have also analysed another set of PARADISE query data, but for queries for people in the US. This analysis is included as earlier investigations [Bar95a] showed that the format of queries for people in the US differed substantially to queries for people in the UK. The likely reason for this is that US users predominantly queried US data, while UK users looked for people
in the UK, and that these sets of users had different cultural and network service influences on how people should be named. This data set is tagged PARA-US.

Fourth, the data for UCL-CS is divided into those queries within UCL and those outside of UCL (but still within the UK). I was interested to see whether the format of local queries was different to those outside of the organisation. These two data sets are tagged UCL-CS-local and UCL-CS-remote.

The six sets of person name query data are summarised in Table 3.30.

<table>
<thead>
<tr>
<th>Source of queries</th>
<th>Sample size</th>
<th>Different names</th>
<th>Names occurring 2+ times</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARA</td>
<td>887</td>
<td>621</td>
<td>105</td>
</tr>
<tr>
<td>PARA-US</td>
<td>855</td>
<td>735</td>
<td>27</td>
</tr>
<tr>
<td>ULCC</td>
<td>1000</td>
<td>740</td>
<td>51</td>
</tr>
<tr>
<td>UCL</td>
<td>845</td>
<td>565</td>
<td>90</td>
</tr>
<tr>
<td>UCL-CS remote</td>
<td>777</td>
<td>417</td>
<td>65</td>
</tr>
<tr>
<td>UCL-CS local</td>
<td>961</td>
<td>578</td>
<td>131</td>
</tr>
</tbody>
</table>

Table 3.30: Basic characteristics of six sets of person name query data

A substantial difference between person name queries and those for countries, organisations and departments is that there is much more variety of form of person names. Consequently, the analysis of query forms is more complicated and comes in two parts: first, of very broad categories, as in Table 3.31; second, a detailed summary is shown in Table 3.32.

The broad categories of input are:

**Single token:** The user entered a single token, for example a forename, surname or userid.

**Multiple tokens:** The user entered a name that consisted of several tokens. This was presumed to always include a surname, but variously also included forename(s) and/or initial(s). The surname was either the first or last token in the input.

**List option:** The DE interface allows the user to enter an asterisk wild-card character to list people entries; the user can then select the required person from the list.

**Null entry:** DE allows users to omit a person name if they wish to look up organisational or departmental details.

**Other input:** This catch-all category includes several types of erroneous input including: non-person names, rubbish such as “asdf” and other input that I was unable to classify.

Overall, single token input was the most popular form. The PARA-US query set was the exception in that multiple token queries were the most popular. No clear picture emerges for
the option of listing entries; it varied between just over 5% and almost 25% of all queries. No person name was entered on average between 5% and 10% of the time; there is no way of knowing whether these were intended to be department or organisation queries, or whether the omission of the input was a mistake on the user's behalf, caused perhaps by unfamiliarity with the user interface. The principal component of the other input category varied between data sets; there was more rubbish and organisation names for the PARADISE query sets, while for UCL and UCL-CS, the queries in this category were primarily for people's roles – Postmaster, Helpdesk, Librarian etc – or for rooms.

We now examine the different forms of person name query input in more detail. Table 3.32 shows the breakdown of each query format for each of the six sets of query data. Each form of input is assigned a code letter which will be used in subsequent discussion. Each form of input is illustrated by an example of the format. The query examples are premised on the user seeking an entry for "Alan Bruce Smith". The table is ordered so that the most popular format overall comes at the head of the table, while the least popular format comes last. To simplify the analytical task, examples with full-stops are treated as follows: if the dot has no adjacent space, the dot is treated as a space character; if there is an adjacent space character, the dot is discarded. Thus "a.smith", "a smith" and "a. smith" are treated as identical; similarly, "alan.smith" and "alan smith" are treated as alike. We will examine later the extent to which users enter a dot following initials. A further simplification is that queries with more than three tokens are rounded down to three tokens: e.g., "a b c smith" is classed as "a b smith".

Before proceeding we must note a caveat concerning this analysis. Human names have enormous variety, even within one country or one culture. The data I am using for this analysis largely pertains to the academic community, which is more multi-cultural than for typical organisations, and thus shows wider variety still. Two examples show the extent of this variety. First, the University of Michigan, with just over 110,000 staff and students, has almost 45,000 distinct surnames. Second, a dictionary of first names for English speaking countries[DG93], which contains over 10,000 forenames, only claims to be 95% complete. Furthermore, many names can be used
both as forenames and surnames; obvious cases include "james"; less well-known cases include the author's surname "barker". A consequence of this is that definitive analysis is impossible without recourse to questioning the user of his/her intentions. However, the problem is greatly simplified if the target database is available, as it is then possible, given considerable effort, to make more informed guesses about users’ intentions. This type of detailed analysis has been undertaken for the UCL query data. A result of this is that the analysis of this data set is probably more accurate than that for other data sets when it comes to subjective decisions such as whether a name is meant to be a forename or surname.

Now for the analysis. Surname only (code A) is the most popular format followed by Forename Surname (code B) for all but the PARA-US data, where their relative popularity is reversed. Initial Surname (code C) is next most popular, used for between 5% and 10% of queries except for PARA-US. These three formats account for over 85% of all input in all sets of query data. Forename only (code D) is next most popular, and particularly so for the UCL services; searching by forename is probably more natural when looking for a local colleague's entry than when searching for someone at another institution.

There are several formats where the user has supplied two or more forenames or initials; these amount to 2.6% of all input.

The next most frequent group of formats is where the person’s name is entered in reverse order, i.e. surname first. Over 5% of the PARA-US queries and over 3% of the ULCC queries were in one of these reverse formats, but the reverse format was little used in the other query sets. This style of input is generally easy to detect: over 80% of these reverse format queries have a comma after the surname. In a few instances an underscore is used instead of a comma. Occasionally, such as for format types “smith a” or “smith a b”, there is no punctuation to give a clue as to the intended format; however the last token being a single letter mostly serves as an indication of the format. I say mostly as there is no simple way of distinguishing between these reverse formats and input of the form “alan s”; fortunately this last type of input is very rare.

Another potential problem in understanding the syntax of users' input occurs when users enter two initials as a single token as in “ab smith”. However, this does not appear to be a serious problem for several reasons. First, this format is very rarely used; six times in over 5000 queries. Second, two letter names occur very infrequently in user input; there were only ten cases, and only three as the first token of user input. On the evidence of my sample data, if the first token is two letters it is more likely to be a pair of initials rather than a short name. Third, since the name in the directory may include forenames or initials, we will see in Chapter 5 that the basis of a successful matching strategy must be to normalise the input to a simple standard form. “al smith”, where “al” is short for “alan”, and “ab smith”, where “ab” is a pair of initials, can both be reduced to “a smith”, and the problem of interpretation is no longer an issue.

A trickier case is where the user enters a name of the form “asmith”, “absmith” or “smitha”. Fortunately these formats are used infrequently, as they are very hard to detect. I suspect that
### 3.7. PERSON NAMES

<table>
<thead>
<tr>
<th>Code letter</th>
<th>Example input</th>
<th>Each category as a percentage of name input</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>PARA</td>
</tr>
<tr>
<td>A</td>
<td>smith</td>
<td>62.84</td>
</tr>
<tr>
<td>B</td>
<td>alan smith</td>
<td>21.66</td>
</tr>
<tr>
<td>C</td>
<td>a smith</td>
<td>6.46</td>
</tr>
<tr>
<td>D</td>
<td>alan</td>
<td>1.00</td>
</tr>
<tr>
<td>E</td>
<td>a b smith</td>
<td>3.01</td>
</tr>
<tr>
<td>F</td>
<td>smith, alan</td>
<td>0.00</td>
</tr>
<tr>
<td>G</td>
<td>alan bruce smith</td>
<td>1.00</td>
</tr>
<tr>
<td>H</td>
<td>&lt; UID &gt;</td>
<td>0.43</td>
</tr>
<tr>
<td>I</td>
<td>&lt; UFN &gt;</td>
<td>0.29</td>
</tr>
<tr>
<td>J</td>
<td>alan b smith</td>
<td>0.72</td>
</tr>
<tr>
<td>K</td>
<td>&lt; WILD – CARD &gt;</td>
<td>0.57</td>
</tr>
<tr>
<td>L</td>
<td>smi</td>
<td>0.00</td>
</tr>
<tr>
<td>M</td>
<td>smith, a</td>
<td>0.00</td>
</tr>
<tr>
<td>N</td>
<td>smith a</td>
<td>0.72</td>
</tr>
<tr>
<td>O</td>
<td>asmith</td>
<td>0.00</td>
</tr>
<tr>
<td>P</td>
<td>smith alan</td>
<td>0.29</td>
</tr>
<tr>
<td>Q</td>
<td>ab smith</td>
<td>0.14</td>
</tr>
<tr>
<td>R</td>
<td>a bruce smith</td>
<td>0.29</td>
</tr>
<tr>
<td>S</td>
<td>absmith</td>
<td>0.00</td>
</tr>
<tr>
<td>T</td>
<td>smith a b</td>
<td>0.29</td>
</tr>
<tr>
<td>U</td>
<td>alans</td>
<td>0.14</td>
</tr>
<tr>
<td>V</td>
<td>smith a</td>
<td>0.00</td>
</tr>
<tr>
<td>W</td>
<td>alan s</td>
<td>0.00</td>
</tr>
<tr>
<td>X</td>
<td>smith, alan b</td>
<td>0.00</td>
</tr>
<tr>
<td>Y</td>
<td>smith, alan bruce</td>
<td>0.00</td>
</tr>
<tr>
<td>Z</td>
<td>alan smith,</td>
<td>0.00</td>
</tr>
<tr>
<td>a</td>
<td>smitha</td>
<td>0.00</td>
</tr>
<tr>
<td>b</td>
<td>mr a smith</td>
<td>0.00</td>
</tr>
<tr>
<td>c</td>
<td>a s</td>
<td>0.00</td>
</tr>
<tr>
<td>d</td>
<td>smith_ab</td>
<td>0.14</td>
</tr>
<tr>
<td>e</td>
<td>smith, alanb</td>
<td>0.00</td>
</tr>
<tr>
<td>f</td>
<td>smith alan bruce</td>
<td>0.00</td>
</tr>
<tr>
<td>g</td>
<td>smith1</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Table 3.32: Detailed summary of person name query formats
in many cases this type of input corresponds to a computer uid; I was unable to verify this with
the data at my disposal.

While personal title are not uncommon in directory names, they are used very infrequently
in user queries; my sample data included only one case. While personal titles may help users to
recognise the correct entry, they offer little assistance in matching.

Just over 0.5% of all input was a uid, with the proportion nearer 1% for all three sets of UCL
queries. UFNs were used very little by UCL users, but were favoured by the PARA-US users.
More than half the UFNs did not use comma separated fields as described in [HK93], but used
the space separated format used by Netfind [ST91]. A UFN such as “a smith st andrews uk ”
is clearly harder to interpret than “a smith, st andrews, uk” although we will see in Section 3.8
that we can generally interpret these unstructured UFNs correctly.

Spelling mistakes

The huge variety of names means that it can be difficult to decide whether a name is correctly
spelled or not. The task is possible if one knows the name(s) that the user is trying to enter.
While I was unable to question users about what they intended, access to the database for UCL
has allowed reasonable guesses to be made about whether instances of UCL users’ input were
misspelled versions of names in the database.

I analysed the UCL input closely to detect spelling mistakes. Out of 669 cases where a
name was entered, 61 appeared to contain a spelling mistake. These 61 can be regarded as two
distinct groups. First, in 11 cases, the user’s input was a truncated version of the full name. In
some cases, these truncated names were probably cognitive errors, such as entering “johnston”
for “Johnstone”. In other cases, it appeared as though the user deliberately entered a truncated
name, possibly to reduce the amount of typing, possibly because the user was unsure about a
spelling. A clear example of this type of shortened input is where a user entered “I.e.will” for
“L.C. Willoughby”.

The remaining 50 errors were all conventional spelling mistakes. This is 7.47% of queries where
the user entered a person name. I would guess that most of these (about 40) were cognitive errors,
rather than typographical errors. Consequently, of these 50 spelling errors, 30 were Damerau
errors, a lower proportion of the total than for the other types of input.

Table 3.33 shows an analysis of which character position contained the first misspelled char­
acter in person name input and in individual tokens in person name input. The table shows a
rather different pattern for the position of spelling mistakes in user input of human names to
that seen for organisation and department names. Two-thirds of spelling mistakes occurred up
to the fourth character in any misspelled word, and half the spelling mistakes were one of the
first four characters entered in a person name. The reasons why spelling mistakes occur earlier
in person names than for other input types are probably two-fold. First, person name spelling
errors are more cognitive than typographical. Second, person name input tends to be shorter: for
3.7. PERSON NAMES

<table>
<thead>
<tr>
<th>Character position of spelling mistake</th>
<th>No. of misspellings by position in word</th>
<th>No. of misspellings by position in input</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>17</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11+</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>Total</td>
<td>50</td>
<td>50</td>
</tr>
</tbody>
</table>

Table 3.33: Character position of first incorrect character in misspelled person name input

the UCL data person name input was about two and a half characters shorter than department name input. Whatever the reasons, the tendency for person name spelling errors to occur nearer the start of input suggests there is less of a case for a matching technique for misspelled person names based on truncating input than there was for organisation and department names.

Foreign names clearly posed problems; basing this analysis on a university database probably over-emphasises this type of spelling problem. Even if we disregard foreign names, we must still acknowledge that human names are hard to spell; it is not for nothing that the phone directories contain lists of alternative spellings for frequently misspelled names. For example, my local directory annotates the entry for “Shepard” with ‘See also “Shephard”, “Shepherd”, “Sheppard” and “Shepperd”‘[Tel95].

Use of Dots with Initials

Overall 7.8% of all input had one or more initials. In almost half theses cases the initials were followed by dots. The breakdown by query set is shown in Table 3.34.

There is no clear pattern in the data; some groups of users prefer one format and some the other. For example, the UCL-CS users prefer the form with dots, almost always “a.smith” rather than “a. smith”; this might be because UCL CS users are familiar with email names using this format. Presumably other groups of users are subject to similar influences.
CHAPTER 3. USER INPUT

<table>
<thead>
<tr>
<th>Source of queries</th>
<th>No. of queries with initials</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>without dots</td>
<td>with dots</td>
<td></td>
</tr>
<tr>
<td>PARA</td>
<td>33</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>PARA-US</td>
<td>29</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>ULCC</td>
<td>70</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>UCL</td>
<td>51</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>UCL-CS-remote</td>
<td>25</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>UCL-CS-local</td>
<td>11</td>
<td>30</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.34: Initials with and without dots

Punctuation and Stop List Words

Although there are many subtly different formats of human names, there is less scope for including punctuation and stop list words that may inhibit matching. However, there are a few special cases of which the user interface designer should be aware. Table 3.35 shows how often these cases arise.

<table>
<thead>
<tr>
<th>Source of queries</th>
<th>Input includes ...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>comma</td>
</tr>
<tr>
<td>PARA</td>
<td>0</td>
</tr>
<tr>
<td>PARA-US</td>
<td>47</td>
</tr>
<tr>
<td>ULCC</td>
<td>24</td>
</tr>
<tr>
<td>UCL</td>
<td>12</td>
</tr>
<tr>
<td>UCL-CS-remote</td>
<td>2</td>
</tr>
<tr>
<td>UCL-CS-local</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3.35: Queries containing stop list words and punctuation

We noted earlier that reverse format queries constitute 5% of the queries in some sets of query data, and that usually these are indicated by a comma after the first token. There are also a few UFN queries in the samples, and these also have commas, in this case as field separating characters. We can, however, almost always distinguish a valid, reverse format person name from an invalid UFN as most UFNs have three or more components and thus two or more commas.

We cannot expect users to always correctly include or omit hyphens. It will probably help to remove hyphens from input if matching otherwise fails.

Apostrophes are of interest since some directory entries use the form “O’Neill, while others use “Oneill”. If matching failures occur against input including apostrophes it may be prudent to try the match on input without the apostrophe. Users favoured the form with apostrophes. Taking
3.7. PERSON NAMES

all the sets of input data together, there were 13 cases of the form “o’neill”, four of “oneill” and one of “o neill”.

In my sample data the “®” character always indicated that the input was an email address, usually along with a full DNS name. This type of input was not valid in DE.

Underscores occurred very rarely, but in my samples were always used as a separator character between a leading surname and one or more initials.

The substring “edu” at the end of the name input occurred a few times in the PARA-US set of queries; it indicated either a mail address or it was the last part of a UFN.

**Forenames and Their Familiar Forms**

In this section we examine how often users enter the standard form of a forename and how often they enter one of the familiar derivative names. While “steve” and “steven” are very similar, “elizabeth” and “betty” are less so, and user input of “betty” will not match “Elizabeth” without sophisticated matching techniques. Table 3.36 shows the number of occurrences of the ten most popular forenames or familiar names and their corresponding forms.

<table>
<thead>
<tr>
<th>Names(s)</th>
<th>Freq</th>
<th>Short names(s)</th>
<th>Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>david</td>
<td>34</td>
<td>dave</td>
<td>2</td>
</tr>
<tr>
<td>steven or stephen</td>
<td>15</td>
<td>steve</td>
<td>11</td>
</tr>
<tr>
<td>andrew</td>
<td>15</td>
<td>andy</td>
<td>2</td>
</tr>
<tr>
<td>richard</td>
<td>12</td>
<td>rick or dick</td>
<td>0</td>
</tr>
<tr>
<td>robert</td>
<td>11</td>
<td>bob or rob</td>
<td>2</td>
</tr>
<tr>
<td>james</td>
<td>10</td>
<td>jim</td>
<td>5</td>
</tr>
<tr>
<td>christopher, christine or christina</td>
<td>8</td>
<td>chris</td>
<td>9</td>
</tr>
<tr>
<td>elizabeth</td>
<td>8</td>
<td>beth or betty</td>
<td>3</td>
</tr>
<tr>
<td>susan</td>
<td>7</td>
<td>sue or susie</td>
<td>3</td>
</tr>
<tr>
<td>timothy</td>
<td>2</td>
<td>tim</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 3.36: The use of full and familiar name forms in person name queries

Overall, for the ten most popular names, the familiar form was used just over 25% of the time. There is clearly a potential matching problem here unless either directory administrators include familiar forms with entries or user interface developers implement sophisticated matching algorithms.
Any correspondence with RDNs?

The format of person names used as RDNs differs from site to site; thus how closely user input matches directory names is highly dependent on the form chosen by a site. However some insight into how often users are likely to enter RDNs can be gained by examining the UCL queries and database. Users entered a query in the form forename(s)-or-initial(s) followed by surname in 284 queries, whereas there were 385 surname only queries. We will see in Chapter 4 that surnames are unlikely to be used as RDNs. Of the 284 multiple token queries, only 71 matched the RDN. One factor depressing the matching rate is that not everyone at UCL has an entry in the directory; 13.5% of queries appeared not to have a corresponding directory entry. However, even allowing for this, it suggests that only 10-15% of name queries are equivalent to the corresponding RDN. This suggests that a querying strategy which prefers the use of read operations is unlikely to be satisfactory: read operations require the user-provided input to exactly match the directory name.

3.8 UFN input

When I started analysing UFN input, I was initially interested in seeing whether the type-it-all-in-one-go UFN format might induce significant differences in the style of names entered. For example, I wondered whether users would tend to omit some name parts when they were not prompted specifically for each field. I also wondered whether users might enter shorter name components with UFNs; ergonomic considerations when entering long input lines might encourage users to enter terser names.

Unfortunately, I was unable to gather UFN query data that is directly comparable to the data for the individual input fields. The problems stem from the variety of input formats that the free form nature of UFNs allows. If users are not shown examples of reasonable UFN queries, the result is that a large proportion of queries have no chance of being resolved. Usually the problem is that key components, such as the organisation name, are omitted. The format of UFN queries improved dramatically when I showed some example queries. However, the input style very closely mimicked that of the example queries. Thus it is not possible to draw conclusions on different styles of input based on the data available. However the study of UFN input provided some interesting insights into some of the problems with UFN querying. A fuller report of the experiments now follows. The work is based on input to the PARADISE and ULCC services; there was too little data from the UCL services to draw any conclusions.

3.8.1 PARADISE query data

The initial work was done using the PARADISE service. In order to classify the data, I defined the following format to be a reasonable query:
3.8. UFN INPUT

person, [ou,] organisation, [location,] country

This definition of reasonableness is a simplification for two reasons. First, in a few cases where the DIT has an unusual structure, a reasonable query will not find the required directory entry. Second, the UFN algorithm does not work if too many input fields are provided. For example, there is hardly any use of locality entries under the UK country entry, although localities are used in many other countries. One result of this simplification is that the percentage of reasonable queries reported below is a slight over-estimate. This generosity is justified on the grounds that some flexibility of query must be tolerated. We cannot expect users to be aware of, and make allowances for, arcane differences of detail in DIT structure; the onus must be on a DUA and the directory to offer powerful matching facilities. A summary of the analysis of the first set of PARADISE UFN data is presented in Table 3.37.

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reasonable format</td>
<td>37.8</td>
</tr>
<tr>
<td>Reasonable format, but no country name</td>
<td>12.2</td>
</tr>
<tr>
<td>Person name only</td>
<td>24.0</td>
</tr>
<tr>
<td>Country name entered</td>
<td>43.3</td>
</tr>
<tr>
<td>Organisation name entered</td>
<td>62.1</td>
</tr>
<tr>
<td>Domains and mail addresses</td>
<td>5.1</td>
</tr>
</tbody>
</table>

Table 3.37: Results from first set of UFN data

I analysed 1000 queries made in March 1994. The results showed that users were not very successful with UFN based querying; the free form of UFNs resulted in a majority of mal-formed queries. Only a little over a third (37.8%) of all queries conformed to a reasonable format. A further 12% of queries were formatted reasonably except for the omission of a country name; the omitted country was almost always "US". This is not surprising as the Internet continues to be US-centric, with very few US organisations having "US" as their top-level domain.

It is almost impossible to find a person's entry in the current X.500 directory, which does not support indexes which span multiple organisations, without specifying an organisation name.\(^2\) However, only 62% of users specified an organisation name in their UFN query.

Users entered a person name only in almost a quarter of all queries. Again this is not surprising, as this style of query is accepted by the Whois service [FHS85] with which many Internet users are familiar. There is also evidence that some users think the directory is based on an extension of the Internet domain name system, as 5% of queries contained either top-level domains such as

---

\(^2\)It is possible to do this in DE by brute force search techniques - the power search mode.
"edu" and "com", or were full mail names such as "bloggs@foo.bar.edu".

The big problem was that users often did not enter enough information to give their queries any chance of working. Although in general I was not concerned with user interface ergonomics and whether users improved their queries with experience, I did look at follow-up UFN queries to see if users got better with practice. They didn't. The average number of name parts entered was 2.29, 2.07 and 2.16 respectively for the first, second and third UFN queries of logged sessions.

In order to improve the effectiveness of UFN querying, I decided to guide DE's users more by implementing three small changes to the user interface. First, users are always shown example UFNs before they make a UFN query. The three examples shown are given in Figure 3.1:

- p barker, ucl, gb
- goodman, comp sci, univ coll, england
- john, univ los angeles, california, us

Figure 3.1: UFN examples shown to users

Second, if a UFN is entered with fewer than three name components, I now warn the user:

The name you have entered contains only n comma-separated component(s).
You have to enter at least 3 name parts to find people entries.
The suggested format is: person name, organisation name, country name.

The user is then asked if they wish to re-enter their query.

The third change is to look for dots or "@" signs in the input, indicating that the user may have entered domain names or a mail address. The user is warned:

You will do better if you avoid using domain names or userids as search keys.

As before the user is given the option of re-entering their query.

These simple enhancements resulted in much better queries. A summary of the results from a further sample of 1000 queries is given in Table 3.38.

The results show that users enter much improved UFNs given better guidance. The evidence is that a number of small improvements to the DUA help to get the percentage of reasonable queries nearer to 100%.

Some significant improvements in query format can be achieved by handling domain names, which occur as separate components in 6.9% of all input, and particularly "edu" which featured as a token in 3.7% of all queries. A simple rule that removes a domain component such as "edu" or "com" if it is not the last name component, and replaces it with "US" if it is the last component, produced a reasonable looking query for 78% of queries containing an "edu" component. Full email names were entered in a small number (0.5%) of cases.
3.8. UFN INPUT

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reasonable format</td>
<td>67.0</td>
</tr>
<tr>
<td>Reasonable format but no country name</td>
<td>9.1</td>
</tr>
<tr>
<td>Person name only</td>
<td>5.0</td>
</tr>
<tr>
<td>Country name entered</td>
<td>75.5</td>
</tr>
<tr>
<td>Organisation name entered</td>
<td>92.6</td>
</tr>
<tr>
<td>Domains and/or mail addresses</td>
<td>8.2</td>
</tr>
<tr>
<td>Domain as a separate component</td>
<td>6.9</td>
</tr>
<tr>
<td>UFN contained an &quot;edu&quot; component</td>
<td>3.7</td>
</tr>
</tbody>
</table>

Table 3.38: Results from second set of UFN data

Spaces were used as name component separators in 1.3% of the input; this is the format used by Netfind [ST91].

In 1.4% of all UFN input the person's name was entered as two comma-separated name components, in each case with the surname preceding either a forename or an initial. It is hard to detect this format if the second name component is not an initial, but that form occurred only twice in the sample. The UFN specification does allow commas to be included in name components by enclosing the name component within quotes. An example UFN using this facility is:

"bloggs, fred", foobar university, uk

There was no evidence of this form being used, and it is hard to imagine it being used by ordinary users of the directory.

Another possible solution is to use the alternative UFN component separator, the semi-colon. However, this form is not advertised by the user interface, nor any other UFN-based DUA I know of, and it was not used by any users.

The person's name field appeared to be a computer user's login name in 1.4% of all input cases.

In a few cases (1.2%), users entered some of the components in the wrong order. The most common problem was to put the country name before the organisation name, but there were several instances where users entered the organisation name before the department name. The only obvious remedy here is for the user interface to indicate clearly to the user how it is interpreting the query, so that the user can correct the mistake.
Although most X.500 service providers have been slow to attempt to provide multiple organisation searches (due to the cost of distributing searches to large numbers of DSAs), there is some evidence here that users want this type of service. In 2.5% of UFN queries, users attempted some sort of wild-carding for organisation name in order to do a multiple organisation search. The most common form was to enter a null component as in:

Fred Bloggs, , UK

Other similar forms included specifying ***, “univ”, “univ**” or “edu” as the organisation name.

Finally, I compared the overall number of characters entered for a UFN query with that of a simple DE query with four individual prompts. Counting all characters typed, including field separating carriage returns and commas, the UFN queries were on average 23.44 characters long while the simple queries were on average 29.94 characters in length. There are probably several reasons for this discrepancy.

First, even with the greater guidance as to what constitutes a reasonable query, a considerable number of UFN queries still lack important terms such as country and organisation names.

Second, less than 10% of UFN queries included a department name, whereas the equivalent figure is over 25% for the simple style of querying. This is not surprising as two of the three UFN examples (shown in figure 3.1) which users are shown omit a department name. Furthermore, department names are omitted in the suggested UFN format that is shown to users who are advised that their input is unlikely to find an entry.

Third, there is evidence that users type different names in UFN queries to those they use in simple queries. In this case, it seems clear that the examples shown to users have influenced their input, in some cases quite dramatically. Table 3.39 shows some of the most commonly used variants for country names.

The most startling difference between the figures for simple mode queries and UFN queries is that the country name “us”, which was used in only 12.4% of simple mode US queries, was used for 70% of UFN US queries, with an almost corresponding drop in the use of the form “usa”. The presumption must be that this is due to the example query. There is no similar dramatic change in usage of terms for UK queries, although the use of “gb” and “england”, terms which feature in the UFN examples, is significantly more for UFN queries than for simple queries. It seems clear that examples can be used to guide users to particular styles of input and even to encourage them to use particular terms, although the degree of influence does not appear to be consistent from the data used here.

3.8.2 ULCC query data

I also examined a set of 1384 UFN queries made using the ULCC service. This service was running the updated DE UFN software. The results were mostly similar. For example, 5.4% of queries were person name only (5% for the PARADISE service). 77.2% of queries included a
3.8. UFN INPUT

<table>
<thead>
<tr>
<th>Name form</th>
<th>Percentage of each form for UK and US queries for simple queries</th>
<th>UFN queries</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>US variants</td>
</tr>
<tr>
<td>usa</td>
<td>67.5</td>
<td>25.8</td>
</tr>
<tr>
<td>us</td>
<td>12.4</td>
<td>71.9</td>
</tr>
<tr>
<td>united states</td>
<td>14.0</td>
<td>1.3</td>
</tr>
<tr>
<td>other</td>
<td>6.1</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UK variants</td>
</tr>
<tr>
<td>uk</td>
<td>67.0</td>
<td>62.4</td>
</tr>
<tr>
<td>gb</td>
<td>10.3</td>
<td>18.8</td>
</tr>
<tr>
<td>england</td>
<td>10.7</td>
<td>14.5</td>
</tr>
<tr>
<td>other</td>
<td>12.0</td>
<td>4.3</td>
</tr>
</tbody>
</table>

Table 3.39: Contrasting UFN and simple mode input for country names

country name (75.5% for the PARADISE service). 6.9% of ULCC queries included one of the following domain names - edu, gov, mil, org, com, ac, co - as a separate component (6.9% for the PARADISE service).

I observed in the previous section how the examples seemed to influence what users entered. There was further strong evidence of that tendency with the ULCC data. When prompted for an organisation name, and not offered any example organisation names, 95.4% of users entered "university" rather than the abbreviated "uni(v)". The corresponding proportion for the UFN queries is 33.9%. My guess is that this discrepancy is due to the fact that two of the three UFN examples feature the abbreviation "univ", and that users naturally follow these examples.

The analysis of person names provides similar evidence. When prompted for a person name, but with no example formats, 10.0% of users of the ULCC service entered a name consisting of a single initial and a surname. With the UFN input, 53.8% of person names followed this format.

3.8.3 Conclusions on UFNs

DE's abstraction of the directory with its fairly rigid hierarchy of object classes has been rightly criticised as being inadequate for querying certain DIT structures, particularly that where locality entries are placed beneath country entries. UFN querying is often seen as a solution to this problem, since its flexibility means that any DIT structure, within some loose schema rules, can be queried. However, we noted that the benefits of the flexibility of UFNs have to be weighed against problems caused by the high proportion of mal-formed queries. We saw that users can be encouraged, by the use of examples, into entering the appropriate fields, in the correct order.
The evidence is that users followed the style of the example queries quite closely, although there were still a substantial proportion of mal-formed queries, usually due to the omitting of key components. As well as guiding users with the general form of queries, another advantage of using examples is that some of the potential user input problems, highlighted in sections 3.4 to 3.7, can be minimised; users can be shown recommended formats.

However, the use of examples does little to help UFN users with diverse DIT structures. Given examples that illustrate querying the most common structures, which is the most appropriate in any case?

Example queries are essential for UFN querying. However they are not enough alone and UFN search strategies need to:

- omit spurious components: e.g., domains such as “edu” and “ac”, or unnecessary localities
- offer users the opportunity of specifying additional name components such as country, organisation or locality names.

Much work needs to be done on improving the UFN algorithm for it to realise its potential.

### 3.9 Other Lessons

One of the most surprising lessons that I learned while doing the preliminary analysis of the PARADISE query data for this thesis was just how many queries could not work because the input strings contained control characters. For the PARADISE input, 6.8% of ordinary queries and 8.1% of UFN queries contained some control characters, which seriously jeopardised their chances of being matched. There appeared to be two principal problems, both associated with trying to correct erroneous input. First, terminals or terminal emulators often do not map backspace characters onto delete characters. Second, the input buffer often contained escape sequences generated by the arrow keys, which are not handled by the standard input routines. Note that this problem only occurs when DE is accessed using a remote terminal protocol.

Two simple changes to the user interface have solved this problem. First, the input string is purged of erroneous character, backspace character pairs. Second, any remaining control characters are detected and the user is asked to re-enter their input. Ironically, these simple changes are as important for improved name matching as many of the algorithm improvements suggested by the analysis described in Chapters 5 and 6 of this thesis.

### 3.10 Notes and Caveats on Methodology

The usefulness and generality of the analysis presented in this chapter is largely built on two things. First, I have analysed a substantial amount of query data. Furthermore this query data is taken from the logs of four separate services with different user populations. This use of
3.10. NOTES AND CAVEATS ON METHODOLOGY

several sources of query data is reassuring: the results from the analysis of the different services are broadly similar. Second, the queries analysed are all taken from the logs of well-established services. This allows us some confidence that the logs are of genuine "service" usage, rather than of the efforts of enthusiasts experimenting haphazardly with a new toy.

However, there are a number of issues which we need to consider to make a proper assessment of the generality of my findings: a discussion of these issues follows.

The work is all based on analysis of usage of DE. A study is needed of querying from other DUAs in order to better understand the extent to which the foibles of querying described in this chapter are due to the characteristics of DE. I did try to use data from other DUAs for this thesis, but could not obtain any logging from similarly well-established services. However, I am confident for several reasons that much of the analysis here will be applicable to other X.500 DUAs, and also to non-X.500 directory services. First, DE's style of querying where a user is shown a series of prompts is a general directory service abstraction: users are not expected to know anything about the X.500 directory hierarchy (which varies widely), nor expected to enter names in set formats. The input analysed here should be representative of any other directory service which offers a similarly general abstraction; clearly it will be less applicable to services which are more tied to existing naming schemes such as the DNS. Second, UFN format queries are, by definition, meant to follow a user-friendly format rather than X.500-specific conventions. UFN format queries are supported by many X.500 DUAs and are also being used by the SOLO service. Furthermore, the prompted style of input is essentially the same as UFNs, except that the name components are typed and carriage returns take the place of commas. Unfortunately I cannot quantify the extent to which the input I analysed was a reflection of users having learned by trial and error the sort of formats which the system found acceptable.

In some instances, users have a choice of DUAs. For example, most users in UCL's Computer Science department have a choice between DE and an X-Windows based DUA called pod. They also have access to other directory user interfaces such as gopher and WWW gateways to X.500. Furthermore, users may use other non-X.500 services for some directory needs: for example finger(1) or a paper directory for local look-ups. I have no way of knowing whether users turn to DE for specific types of queries, perhaps when other means have failed: this would mean that the DE usage I have measured may be atypical of directory service usage. Alternatively, it might be that users favour one particular tool, such as DE, Whois or Netfind, and use that for everything.

One consequence of only analysing input to DE is that there is no discussion of locality name input, since DE does not offer a locality name prompt.

A benefit of analysing usage of real services is that there is a lot of data, and that the data gathering is invisible to the users and thus can have had no influence on users' input. However, this method gives no opportunity to ask users about their intentions, when these are not clear from their queries. For example, when a user has entered a person's name of "james", did they intend that to be a forename or a surname? Similarly, with organisation names, is input of
"london" meant to be an abbreviation for the "University of London", or did the user intend it to be interpreted as the locality of London? Put simply, the analysis sometimes requires some guesswork regarding users' intentions.

Despite my efforts to gather querying data from diverse sources, the focus on both the users and the portion of the DIT to which their queries relate is narrower than I would have liked. First, most of the users are British or American. Second, most of the queries are for the UK portion of the DIT. Third, the users and the data relate almost exclusively to the academic and research community.

The reasons for the limited focus are pragmatic. First, I have used the data that I had available. Second, it is simplest for me, as an English speaker, to concentrate on English names; the analysis of spelling mistakes would not be possible (for me) for non-English names. Third, the analysis is highly labour-intensive; one has to draw the line somewhere on analysing additional sources of input.

One consequence of this limited perspective is that some of the analysis may not extrapolate well to other languages and cultures. For example, within the academic community, it is reasonable for users to abbreviate their organisation name input of a university name to the town name alone. Similar abbreviations are not available for non-academic organisations. Furthermore, that type of abbreviation will not be so reasonable when the directory consists of thousands of organisation entries. While acknowledging these limitations, I have worked with what was available. My hope is that the analysis here can act as a guide to others, suggesting different but related lines of enquiry.

We noted near the beginning of this chapter that an intention of this work was to assess as well as possible how users formulate directory service queries. It seems reasonable to assume that users approach a new directory service with preconceptions based on their experiences with other services. However, the section on UFN queries illustrates vividly the role of user interface ergonomics in improving the quality of user queries: e.g., the influence of example queries on screen at the time a query is made, any on-line help system, and any written documentation.

I have not attempted a detailed assessment of the effect of these factors on users' queries, or whether the feedback provided by the interface generally helped users to improve their queries. A brief analysis based on the PARADISE query data of the differences between person names entered with and without use of the help system suggests that the broad shape of queries was much the same in terms of number of name parts entered and how often listing was used. One major difference was that one of the formats suggested in the help screen (an initial followed by a surname) was used by nearly 10% of those using the help system, whereas those who did not use help only used this format in 1.1% of queries.

A study of directory user interface ergonomic issues is an important piece of work that remains to be tackled.
3.11 Summary and Conclusions

This chapter presents evidence on user input to a directory service. Understanding the various forms this input takes is a necessary first step towards improving DUA algorithms.

This section draws together the preceding discussion in this chapter, summarises the findings and draws some conclusions.

Basic form of queries

The evidence on country names is distorted by the large proportion of UK and US queries in the samples, as both these countries have popular short form names that are not RDNs; “uk” and “usa” respectively. If we consider all other countries apart from UK and US, the most popular form is a full name form, with the ISO 3166 two-letter country codes accounting for most of the remaining queries.

Just over half the sample of organisation names were full names. The next most popular form was a shortened form, such as “cambridge” for “Cambridge University”. Initials, such as “ucl”, accounted for 10% of all queries. 80% of all single token queries were DNS names. The organisation name queries are heavily biased to the academic and research communities, and the results need to be re-assessed when the directory has diversified to include more commercial organisations.

Whereas DE obliges a user to enter country and organisation names to find a person’s entry, the department name can be omitted. The result of this freedom is that, more often than not, users do not enter a department name. If users do enter a name it is usually a full department name form. There was some evidence that users enter initials for department names much more for local look-ups than for in other organisations. Listing department names was also a popular option, at over 15% of the sample.

The most striking feature of person name input is the huge variety of forms. However, three forms predominated: surname only, forename followed by surname, and initial followed by surname. Somewhat surprisingly, given the influence of other directory services, userids were not a popular form of input. Despite the variety of forms, it is possible to normalise almost all the name forms into a forename(s)-or-initial(s) followed by surname format, using clues such as token lengths (in particular initials), commas and underscores to determine the structure of the name entered.

There is a fundamental problem with UFNs, in that their flexibility gives users a lot of scope for mal-forming queries. The main problem is that users omit crucial components, such as the organisation name. Other problems are that users mis-order components, that they enter too many name components, that they are more prone to entering domain names such as “ac” and “edu”, and that they include commas within human names (which makes them appear to be two separate components). Another possible problem, not properly assessed here, is that the longer
CHAPTER 3. USER INPUT

UFN input strings may lead to more typing mistakes. These difficulties with UFN input mean that the UFN matching algorithms need to be commensurately more intelligent than algorithms such as DE's based on form-filling and with typed input fields.

However, the analysis of UFN input has been limited to national and international servers, where there is no "UFN environment". A UFN environment allows name components to be omitted if the user is querying within a pre-defined local environment; it is the UFN equivalent of DE's default values for certain input fields. A further study is required to assess whether users make proper use of this facility when querying, which allows abbreviated local queries, but still requires full details for remote queries.

Spelling Mistakes

Matching misspelled input has been seen as a key role of approximate matching, possibly the key role. However, experience with the Soundex algorithm used for approximate matching has been that it often gives too many obviously wrong matches. A consequence of this has been that DUA designers have been discouraged from using approximate matching, and the capability for matching misspelled input is lost. Therefore, it is important to understand how often mis-spelling occurs.

The evidence is that the error rates differ markedly for the different input types. For country names the error rate is below 1%; to some extent the error rate is depressed by the shortness of country names, in particular two letter codes; there is less opportunity for errors. For organisation and department names, the error rates are 3-3.5% on average, with the majority of the errors appearing to be typographic errors rather than cognitive errors. However, the proportion of spelling mistakes was over 7% for person names, with the majority of these errors probably being cognitive errors. The variety of human names makes them hard to spell. There is a detailed analysis of matching misspelled input in Chapter 5.

Stop-list Words and Punctuation

The problem of input containing words or punctuation that might inhibit matching mostly affects organisation and department name input. The problem is that input of "department of physics" does not match a directory name of "Physics", or that "king's college" does not match "Kings College", without using some sophisticated approximate matching algorithm. One solution is for DUAs to recognise some special cases and transform the user input into an alternative form, stripped of the special characters or stop-list words, if matching fails. We will explore this area further in Chapter 5.
3.11. SUMMARY AND CONCLUSIONS

User Input and RDNs

The ideal user input, from the point of view of optimising querying algorithms, is for users to enter name parts that exactly correspond to the RDNs of the directory entry. The required entry can then be found with a single read operation. Of course, users are not so obliging. Users tend to favour long country names over the ISO 3166 country codes, although defaulting to local values means that ISO codes are predominant. The defaulting of the local organisation name means that as many as three-quarters of organisation input can be RDNs; for organisation names entered by the user, about one-third are RDNs. When a department name was entered, it was the RDN nearly half the time. However, users entered a department name in less than half the queries. For person names, the large variety of input forms and person name forms in the directory militates against the user entering the right one. Illustrating this using the UCL data, less than 15% of query names were the same as the RDN of the target entry.

Although the majority of input is not equal to the RDN values, we can exploit the fact that queries tend to be clustered for a small number of countries, organisations and departments. This is the principle of locality. A DUA could hold tables that mapped the most common forms of non-RDN input onto the corresponding RDN equivalents. Mapping tables of 20 entries for each of country, organisation and department names could simulate RDN-equivalence in user input for between 75% and 99% of queries.

Normalising user input

The diversity of person name formats focuses attention on the need to do some transformation of person name input into some normalised form(s) in order to help with matching entries in the directory. However, similar transformations may be required for some organisation and department names. These transformations may require the re-ordering and/or dropping of components, so that, for example, "department of physics" is transformed to "physics", or "university of foo" is transformed to "foo university" or just "foo" if matching fails. We will explore these possibilities further in Chapter 5 on matching user input.
Chapter 4
Data in Directory

4.1 Introduction

This is a companion chapter to Chapter 3 on user input. The chapter contains an analysis of many aspects of data held in the X.500 directory. The motivation for this analysis is that once we understand both the nature of directory data and user input, then we are well qualified to determine what is required of querying algorithms.

The study seeks to do the following:

- To determine the basic form of names, and particularly RDNs, in the directory. For example, do administrators follow guidelines such as those set out in RFC 1617 [BKL94] and use full names or do they enter initials?

- To examine the possibility of and/or need for using non-naming attributes for matching. Attributes such as friendlyCountryName for country entries, and associatedDomain for organisation entries, may contain naming information that might be used in matching algorithms.

- To determine how often entries have alternative names.

- To look for name forms that could cause matching problems, such as names including punctuation characters, person names with surname-first common names, or organisation names such as "University of Anyplace" where users may favour the alternative form "Anyplace University".

The structure of this chapter is as follows. First, in Section 4.2 there is a description of the data that has been analysed in this chapter. The body of the chapter consists of the four sections analysing country names (Section 4.3), organisation names (Section 4.4), department names (Section 4.5) and person names (Section 4.6). A number of comments on and caveats concerning the methodology are discussed in Section 4.7. There is a summary of the results of the analysis and conclusions in Section 4.8.

4.2 The Directory Data

This section tries to summarise some of the main characteristics concerning naming entries in the DIT. The size of the DIT, estimated in [Goo94] to hold well over a million entries in May 1994, precludes an analysis of all directory data. Instead, we have to rely on drawing inferences from samples of data. In this section I describe briefly how the data analysed in this chapter was gathered.
There are fewer than 40 countries underneath the root node, and it was thus possible to analyse all the country names. The data was gathered using ISODE's *dish* DUI[Kil91].

The organisation name data is the full set of naming data registered beneath five sample country entries. The countries are, with the country codes in parentheses: the United Kingdom (GB); the United States (US); Spain (ES); Germany (DE); the Netherlands (NL). Each of these countries is well established in the DIT and has a non-trivial number of organisation entries. Furthermore, the data for these countries is available reliably. The data was gathered using *dish*. The sample comprises over 650 entries.

With some exceptions, which I will elaborate on shortly, the organisational unit data is the naming data from all organisational unit entries in the same five countries. The data was gathered using the *power search* mode of a slightly modified version of the DE DUI. This produced a sample of over 5,500 entries. However, the sample falls short of containing all organisational unit data for at least three reasons. First, the sample will be missing entries for any organisational DSAs that were down at the time the searches were made. Second, some DSAs may refuse to respond to queries of the type used to gather the data here; some organisations have configured their DSAs to refuse queries that will return many entries to prevent trawling of their data. Third, network congestion or routing problems may prevent connection to remote DSAs.

The person entry data was gathered by taking a sample of entries for each of the same five countries. For each country, I chose between one and three popular surnames, and used DE's power search mode to return all entries in the country with the given surnames. This produced a sample of over 4,000 entries. The caveats I listed regarding the collection of the organisational unit data also apply to the gathering of person name data.

### 4.3 Country names

The X.500 standard stipulates that the *countryName* attribute must be used for naming country entries; the value should be the country's ISO 3166 two-letter country code. While these values are well-suited to one function of directory RDNs, namely to ensure unique names amongst a set of sibling entries, they are not user-friendly names. With an oversight typical of CCITT/ISO standards, X.500 does not provide a standard attribute that can be used for naming countries with their full names. This means that the directory is unable to match input such as “United Kingdom” as the country name for a GB entry. This deficiency was remedied by the definition of the *friendlyCountryName* attribute in RFC1274 [BK91]. Systems supporting the schema extensions described in RFC1274 can now have “United Kingdom”, “England” or “UK” as *friendlyCountryName* attribute values of the GB entry.

At the time this analysis was undertaken (June 1995) there were 38 country entries under the root of the DIT. 37 of these entries, the exception being that for France, have at least one
4.4. ORGANISATION NAMES

The X.500 standard says that the organizationName attribute should be used for naming organisation entries. Whereas the choice of RDNs for country entries is mandated by the X.500
standard, the choice of organisation names is delegated to registration authorities within each country. Thus, there is scope for different naming policies within different countries. Recognising that the directory is best served by a consistent policy on naming, international recommendations on naming organisations have been set out, initially in RFC1384 [BK93], and updated in RFC1617 [BKL94]. These recommendations stem largely from early experience of managing registration in the UK part of the DIT, when the software developers (and de facto system managers) acted as an informal registration authority. The principal recommendation in the RFCs concerning organisation RDNs is that organisations should choose long name forms, but that these names should be familiar names rather than obscure names such as might be found on official charters.

Since these recommendations were first produced, a number of countries have developed their own guidelines, and sometimes their own registration authorities.

In the UK, an organisation called DISC operates a distinguished name registration scheme: this is described in [BSI94]. The emphasis is slightly different to the guidelines in the RFCs. Organisations are allowed to register short names or initials, providing DISC thinks the name is a reasonable choice. Thus, the Institute of Boils and Moles, if not widely known by its initials, would probably be prevented from registering as IBM. There is a challenge procedure to resolve clashes of interest. Organisations are also allowed to register more than one RDN; they can do this to prevent other organisations from registering a name that might easily be confused with their own name.

The Australian part of the DIT operates under guidelines that are documented in RFC 1562 [MP93]. These state that an organisation should use its officially registered name. This avoids the need for another naming authority and ensures that name clashes are avoided.

The North American Directory Forum (NADF) also recommend (in [NAD91] and [NAD93]) that naming is based on, what they term, civil authority. This authority is a mixture of national, regional and local standing, depending on the jurisdiction of the body that has registered or chartered a given organisation. The scheme guarantees uniqueness of names within the US part of the DIT for any organisation, however big or small. As for Australia, this approach obviates the need for a directory-specific registration authority.

The various recommendations on organisation naming are mainly concerned with the choice of RDN, although the various guidelines usually say that other forms of the organisation name should be included as alternative values of the organizationName attribute. The chief role of alternative values is to help a user when searching the directory; system administrators are advised to include any names by which users know the organisation familiarly. These names often include sets of initials: for example, "UCL" is an alternative value for "University College London".

Although the organizationName attribute holds the principal set of naming information for organisations, some other attributes contain information that may help users to find the entries they require. Another naming attribute used in many organisational entries is the associatedDomain attribute: this holds an organisation's fully qualified DNS name - the attribute is defined in
the COSINE/Internet schema extensions document, RFC 1274. If the attribute is widely used, and if users specify organisational domain names in queries, this attribute could be used as part of a search strategy.

Before moving on to examine the use of the various naming attributes, we must note an important qualification of the picture just described. The directory is still (in 1995) run largely by the research networks, rather than by public network operators. Although some naming registration schemes, such as that operated by DISC, are in place, there is often no obligation for organisations registering in the current directory to have followed the national or international guidelines. However, the research networks have tried to anticipate the introduction of "official" control of the directory, and have coerced most organisations into accepting the prevailing policies on naming. The advantage to an organisation of doing this is that it should not have to change its organisation's RDN when the directory is eventually subject to official naming authorities.

There is some basic analysis of the use of naming attributes in each of the five countries. A more detailed analysis of various aspects of UK organisation names follows. However, the restricted scope of the UK analysis means that these conclusions are specific to the UK data. In particular this data set contains a large proportion of academic institutions; this is typical of the current directory but lacks the diversity of organisations one would expect in a fully-fledged directory. Another characteristic is that much of the analysis is specific to the English language. However, it is anticipated that often there will be analogous problems with non-English data sets; the analysis here should act as a prompt for others to consider problems pertaining to their own language.

<table>
<thead>
<tr>
<th>Country</th>
<th>No. of orgs</th>
<th>% with alt. org name values</th>
<th>Ave no. of org names</th>
<th>% of organisations with attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>search-Guide</td>
<td>associated-Domain</td>
</tr>
<tr>
<td>UK</td>
<td>143</td>
<td>71.33</td>
<td>2.15</td>
<td>5.59</td>
</tr>
<tr>
<td>US</td>
<td>110</td>
<td>56.36</td>
<td>1.79</td>
<td>0.00</td>
</tr>
<tr>
<td>ES</td>
<td>97</td>
<td>81.44</td>
<td>2.05</td>
<td>0.00</td>
</tr>
<tr>
<td>DE</td>
<td>222</td>
<td>90.09</td>
<td>2.27</td>
<td>0.45</td>
</tr>
<tr>
<td>NL</td>
<td>94</td>
<td>51.06</td>
<td>2.07</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Table 4.1: Summary of organisation name data

Table 4.1 summarises the five sets of organisation name data. The majority of organisation entries (almost three-quarters of my sample) include alternative values of the organizationName attribute; the average for each of the sets of data is approximately two names per organisation. Another non-naming attribute that includes name information is associatedDomain. The proportion of entries with an associatedDomain attribute varies widely between the different countries; overall for my sample, a little over a third of all organisations included an associatedDomain
attribute.

We noted in Section 2.5.3 that the searchGuide attribute could be used to help DUAs formulate their queries: in practice the attribute is rarely used. There are some other attributes that could be used to support more generalised, yellow pages querying. The majority of entries include a localityName attribute. If the attribute is included it is generally the name of the town or city of the organisation, although in the US the attribute value is almost always a city and state name. Most entries contain a postalAddress attribute. A smaller number, but still the majority, of entries have a postalCode attribute.

**RDNs and Number of Name Tokens**

One measure that tends to confirm that the advice on choosing long name forms as RDNs has been followed is the proportion of RDNs that consist of a single word. The analysis is presented in Table 4.2.

<table>
<thead>
<tr>
<th>Country code</th>
<th>single word</th>
</tr>
</thead>
<tbody>
<tr>
<td>UK</td>
<td>3.50</td>
</tr>
<tr>
<td>US</td>
<td>10.00</td>
</tr>
<tr>
<td>ES</td>
<td>12.37</td>
</tr>
<tr>
<td>DE</td>
<td>11.71</td>
</tr>
<tr>
<td>NL</td>
<td>10.64</td>
</tr>
</tbody>
</table>

Table 4.2: Percentage of RDNs consisting of a single token

On average, about one in ten names in the directory consists of a single word, and no more than one in eight for any of the countries sampled. This is useful knowledge as it means that user input of a single token is unlikely to be a directory RDN, and that such a name almost certainly needs to be resolved using a directory search operation. Alternatively, if the user input is an organisation RDN, we can omit this stage of name resolution.

**Alternative Names**

Table 4.3 shows that directory administrators have mostly followed the advice of specifying longer name forms for an organisation's RDN, but specified shorter names as alternative values. A typical example is the University of Brighton, which has chosen “University of Brighton” as its RDN, and included “Brighton University” and “Brighton” as alternative values. Another reason why alternative values tend to be shorter can be seen in the next section on domain names.

Some other cases where alternative names are appropriate are described in RFC 1617. The RFC suggests that for non-English-speaking parts of the DIT, administrators would improve the directory service by including English language variants whenever possible. The samples I have
examined show that a fifth of the Dutch entries have English language variants, as did a few of
the German entries. On the whole, this recommendation is not widely implemented.

Another use for alternative names is when non-ASCII characters are required, such as for
characters with accents and diacritics. An example of such a character is the German umlaut.
The recommendation is to include a transliterated form of the name, as many display devices
may not be able to, or may not be set up to, render accented characters correctly.

### 4.4.1 DNS Domain names

Many system administrators have guessed correctly that users often enter organisation names that
are also DNS domain names: see Chapter 3 for evidence on this. These directory administrators
have included one or more domain names as alternative values of the organisation name. For
example, the DNS name for the University of Kent is "ukc.ac.uk". The University of Kent has
included the value "UKC" as an alternative value of its organisation name. However, not all
system administrators have done this. Table 4.4 shows how many organisations in each country
have associatedDomain attribute values that are not included either as alternative organisation
names or appear as substrings of organisation names.

Table 4.4 shows that the majority of organisations with associatedDomain attributes do in-
clude these values as organisation names. For example, in the UK only 8.39% of organisation
entries include an associatedDomain attribute value that is not also included as an organizationName value. However a sizeable minority of domain names are not included as organisation
names; there is thus a case for DUIs searching on the associatedDomain attribute as well as the
organizationName attribute.

### 4.4.2 UK-specific analysis

The UK part of the DIT contained 143 organisation entries when it was sampled in June 1995.
Of these, 95 had been added by organisational administrators; the remaining 48 were bulk-loaded
as part of an attempt to get full coverage in the UK DIT of all universities and similar level
CHAPTER 4. DATA IN DIRECTORY

<table>
<thead>
<tr>
<th>Country</th>
<th>Orgs where associatedDomain attribute contains additional name(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>number of orgs</td>
</tr>
<tr>
<td>UK</td>
<td>12</td>
</tr>
<tr>
<td>US</td>
<td>15</td>
</tr>
<tr>
<td>ES</td>
<td>6</td>
</tr>
<tr>
<td>DE</td>
<td>4</td>
</tr>
<tr>
<td>NL</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4.4: Entries where an associatedDomain attribute contains additional name forms

Educational organisations.

Names as Initials

36 of the 95 names added by the organisations themselves include a name value consisting solely of initials; none of the bulk-loaded organisations has a name of solely initials. 25 of this 36 are DNS names, although only 18 of them are included as associatedDomain names. 35 are of the form without dots, e.g. "UCL", while only one includes dots, as in "U.C.L.".

Punctuation and Stop List Words

We noted in Chapter 3 that punctuation and stop list words can inhibit matching. For example, user input of "Foobar Ltd." with the trailing dot character does not exactly or substring match a directory name of "Foobar Ltd" without the dot. However, the problem of forms including stop list words and punctuation can be mitigated by the directory containing alternative organizationName values. How often do these stop list words and punctuation characters occur in organisation names?

Three entries include organisation names with commas; the two that were added by the organisations themselves include alternative names without the commas.

Five entries contain an apostrophe. Two of the five entries have alternative names excluding the apostrophe. One, Goldsmiths' College, even includes a further alternative name with the apostrophe before the "s".

There are three entries with the abbreviation "St" for Saint. Two take the form "St" without the dot, while the third includes the dot as in "St.".

Eight organisations that are limited liability companies have a name that ends with "Ltd", while only one ends with "Ltd.".

Four organisations have hyphenated names, and none of these four includes a version without the hyphen.
4.5. DEPARTMENT NAMES

The most common stop list words are "the" (22 names for 21 organisations), "of" (91 names for 79 organisations) and "and" (13 names for 9 organisations).

The high frequency of the word "of" is due to the academic bias of the data; all but one of the 91 occurrences is "University of", "Polytechnic of" or similar. In most cases, administrators have included an alternative form without the word "of". Only 5 of the organisation entries added by the organisations themselves do not include an alternative name without the word "of".

We saw in Chapter 3 that directory users favoured the forms "University of Foo" and "Foo University" almost equally in their queries. Universities, however, show a distinct preference for the form "University of Foo". Of the 97 entries with the word "university" in one or more of the organisation names, 44 had an RDN of the form "University of Foo" while 19 had an RDN as in "Foo University". Offering some help for users unsure about which form to use, 25 had both forms.

### 4.5 Department names

The standard says that the `organizationalUnitName` attribute should be used for naming organisational units or departments. The choice of an organisation's department names is the responsibility of that organisation. As a result, one would expect more variety of naming styles than for organisation entries.

The standard suggests a long name form should be used as the distinguished name value and that abbreviations should be included, where appropriate, as alternative values. RFC 1617 gives the same advice.

There are several books on X.500 (e.g. [Cha94] [Ste93] [Rad94]) but these offer system administrators little help with naming entries, as the books concern themselves more with the capabilities of the standard rather than details of using it to good effect.

<table>
<thead>
<tr>
<th>Country</th>
<th>No. of orgs</th>
<th>No. of depts</th>
<th>% depts with alt. values</th>
<th>Average names per dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>UK</td>
<td>41</td>
<td>2391</td>
<td>26.71</td>
<td>1.51</td>
</tr>
<tr>
<td>US</td>
<td>28</td>
<td>1028</td>
<td>44.84</td>
<td>1.51</td>
</tr>
<tr>
<td>ES</td>
<td>24</td>
<td>1073</td>
<td>41.94</td>
<td>1.74</td>
</tr>
<tr>
<td>DE</td>
<td>56</td>
<td>966</td>
<td>36.02</td>
<td>1.42</td>
</tr>
<tr>
<td>NL</td>
<td>8</td>
<td>274</td>
<td>93.07</td>
<td>2.09</td>
</tr>
</tbody>
</table>

Table 4.5: Summary of organisational unit name data

A summary of the five sets of department name data is given in Table 4.5. The analysis is based on names for over 5000 departments. The analysis shows that departments tend to have
fewer names than was the case for organisations; an average of just over one and a half as opposed to just over two. A little over a third of the departments in the sample had one or more alternative names.

<table>
<thead>
<tr>
<th>Country</th>
<th>No. of orgs</th>
<th>&lt;10</th>
<th>10-50</th>
<th>50-90</th>
<th>&gt;90</th>
</tr>
</thead>
<tbody>
<tr>
<td>UK</td>
<td>41</td>
<td>23</td>
<td>5</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>US</td>
<td>28</td>
<td>20</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>ES</td>
<td>24</td>
<td>11</td>
<td>2</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>DE</td>
<td>56</td>
<td>26</td>
<td>6</td>
<td>9</td>
<td>15</td>
</tr>
<tr>
<td>NL</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 4.6: No. of organisations with varying percentages of departments with alternative names

Table 4.6 shows the number of organisations in each country that have certain percentages of departments with alternative names. For example, 23 (out of 41) UK organisations had alternative values for less than 10% of their department entries. Overall the table shows that just over half of all organisations created alternative values for less than 10% of their department entries; just over a fifth could be regarded as systematically creating alternative department name values, with more than 90% of their departments having alternative names.

<table>
<thead>
<tr>
<th>Country</th>
<th>Ave. length of org unit names</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RDNs</td>
</tr>
<tr>
<td>UK</td>
<td>22.90</td>
</tr>
<tr>
<td>US</td>
<td>18.51</td>
</tr>
<tr>
<td>ES</td>
<td>28.65</td>
</tr>
<tr>
<td>DE</td>
<td>23.17</td>
</tr>
<tr>
<td>NL</td>
<td>21.86</td>
</tr>
</tbody>
</table>

Table 4.7: Average length of organisational unit names

There is no consistent picture of longer RDNs and shorter alternative names as there was for organisation names: see Table 4.7 for details. The UK and NL data follows the pattern of longer RDNs and shorter alternative values, but the data for the US and DE shows longer alternative names. The very short average for NL alternative names is due to the prevalence of department codes or systematic abbreviations: e.g., "Biomedisch Technologisch Instituut" has an alternative value of "BMTI".
As for organisation names, there is little following of the recommendation in RFC 1617 that non-English-language organisations should include English-language versions of department names. One Spanish organisation in the sample did include English names for almost all its departments, but no other Spanish organisations included any English language names. A few German and Dutch organisations included a smattering of English language names, but none did so systematically.

There is more use of alternative names, comprising ASCII characters only, when non-ASCII characters have to be used to represent department names. Almost all Spanish organisations using non-ASCII characters to represent department names including letters with diacritics also used a transliterated name using only ASCII characters.

The issue of alternative names did not arise with German organisations, as none of them included non-ASCII characters in their department names. Instead they all used transliteration in their names: e.g. "für" is represented as "fuer".

Similarly, although the Dutch language has accented characters, there were none in my sample of department names.

4.5.1 UK-specific analysis

The UK part of the DIT contained 41 organisations with department entries when the sample was taken in July 1995. The sample ranges from two organisations with a single department with a single name, to one organisation with 61 departments that averages almost nine names per department.

As for the organisation data, I have studied the UK department names in more detail to gain a better understanding of the characteristics of the data.

Inconsistent Naming of Objects

The lack of a centralised naming authority for organisation's department entries creates the problem of inconsistent naming. The following example demonstrates the problem well. 28 UK organisations have department names that include one or more of the four strings shown in Table 4.8.

<table>
<thead>
<tr>
<th>Name includes ...</th>
<th>as an RDN</th>
<th>as an alt. name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computing Service</td>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>Computer Service</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>Computer Centre</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Computing Centre</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 4.8: Different names for computer centres
The analysis shows that "Computing Service" is the most popular choice but that there almost as many names including the string "Computer" as there are including "Computing". The only common substring in these name forms is "Comput" and this does not match those departments with names such as "IT services".

A similar problem occurs with Mathematics departments. Of 30 organisations with such departments, 24 had a department name including the word "Mathematics", 10 included the abbreviation "Maths" and there were three cases of "Mathematical Science". The difficulty for the user can be seen if we consider that neither of the shorter forms is a substring of the longer forms. Thus, a user entering "Maths" has to rely on approximate matching to match his/her input with an entry with one of the longer name forms. Unfortunately, and we are anticipating discussion to come in Chapters 5 and 6, the widely used Soundex algorithm does not approximately match any of the three forms.

Not all department names present as many problems. For example, of nine organisations with a French department, all included one name value of "French". Similarly for ten German departments, eight had a name of "German" and the other two included "German" as a substring.

Another problem for the user is that the same name is often used for different objects. There are four cases of the abbreviation "CS", and they all relate to different departments: "Counselling Service"; "Curator's Section"; "Computing Service"; and "Computer Science".

In most cases directory administrators have not included abbreviated forms of names. However the directory user should be aware that they are used in some cases. For example, there are 146 cases of the word "office" but 10 cases of "off". There are 110 cases of the word "engineering", but 20 of the word "eng" and two of "eng.". There are 47 cases of the word "development" and seven different abbreviations of the word, all occurring just once. While some are substrings of the word "development", e.g. "deve" and "dev", others such as "develp" are not.

Given the naming problems, the evidence is that directory administrators need to be assiduous in creating department entries with a number of alternative names, particularly for departments that tend to be named inconsistently in different organisations.

**Punctuation and Stop List Words**

There are several stop list words in department names that occur frequently. These words are devoid (or almost devoid) of any semantics. The main ones, each occurring in a number of organisations, are "and", "of", "centre", "office", "school", "unit", "for", "department", "institute", "faculty" and "division". Since users tend to be terse in their input, these extra words in department names mean that exact matching will be less likely, and users will have to rely on substring matching. Including terse alternative values, such as "Mathematics" or "Maths" for "Faculty of Mathematics", would allow more exact matching.

Some punctuation marks occur frequently. For example, 371 department names out of a total sample of 3618 (10.25%) include some text between a pair of parentheses. Although the word
"and" is used most often as a conjunction (476 times), "+" (46 times) and "&" (22 times) are also used.

The inclusion or not of apostrophes in possessives varies from case to case. If we consider the term "Registrar's" as in "Registrar's Office", five times it occurs with the apostrophe and six times without it. There are also five instances of "Bursar's" and one of "Bursars", and four "Chancellor's" and three "Chancellors". In two cases, both forms were included, but generally the user has no way of knowing which form has been chosen in any instance. It would clearly help users match their required entries if alternative forms, with and without apostrophes, were included in the directory.

The Diversity of Naming

The variety of different names for essentially the same thing coupled with the use of stop list words means that department names differ greatly from organisation to organisation. Table 4.9 shows this diversity and the need for different matching strategies to find the required department entries. The table gives details for the ten most common department names found in the sample; I excluded words such as "department", "office" and "systems" that did not indicate specific departments. For each department name, I have shown the number of matches that would be achieved by four different types of matching: exact matching; leading substring matching, trailing substring matching and any substring matching.

<table>
<thead>
<tr>
<th>Department</th>
<th>Exact</th>
<th>Leading substring</th>
<th>Trailing substring</th>
<th>Any substring</th>
</tr>
</thead>
<tbody>
<tr>
<td>Medicine</td>
<td>2</td>
<td>15</td>
<td>16</td>
<td>46</td>
</tr>
<tr>
<td>Biology</td>
<td>5</td>
<td>9</td>
<td>23</td>
<td>44</td>
</tr>
<tr>
<td>Computing</td>
<td>6</td>
<td>27</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>History</td>
<td>10</td>
<td>15</td>
<td>21</td>
<td>34</td>
</tr>
<tr>
<td>Physics</td>
<td>14</td>
<td>19</td>
<td>22</td>
<td>31</td>
</tr>
<tr>
<td>Biochemistry</td>
<td>7</td>
<td>11</td>
<td>9</td>
<td>25</td>
</tr>
<tr>
<td>Mathematics</td>
<td>13</td>
<td>15</td>
<td>20</td>
<td>24</td>
</tr>
<tr>
<td>Chemistry</td>
<td>16</td>
<td>19</td>
<td>21</td>
<td>23</td>
</tr>
<tr>
<td>English</td>
<td>5</td>
<td>21</td>
<td>5</td>
<td>23</td>
</tr>
<tr>
<td>Law</td>
<td>6</td>
<td>10</td>
<td>10</td>
<td>18</td>
</tr>
</tbody>
</table>

Table 4.9: Matching common department names

The results show that there is little pattern. In only two cases (Mathematics and Chemistry) are more than half the departments exactly matched by the single word department name. In the majority of cases a department is more likely to have a name such as, for example, "Computing
Service" than simply "Computing". This lack of simple names, combined with inconsistency of naming, places an onus either on the user to guess name forms correctly, or on the DUA to use sophisticated matching techniques.

4.6 Person names

The standard says that the commonName attribute should be used for naming person entries. The standard also mandates that every person entry should include a surname; this is useful for matching purposes as the most common type of person name query is surname-only.

As for department names, the choice of an organisation's person names in the directory is the responsibility of that organisation. There is conflicting guidance on recommended name forms. The standard itself shows examples of human names consisting of: surname only; forename followed by surname; a form with a personal title, forenames, surname, and letters indicating academic awards and membership of professional bodies. The books on X.500 tend to follow the standard inasmuch as there is no clear line taken on preferred forms.

RFC1617 tries to step into the breach and advises that the RDN should usually be of the form familiar-first-name surname. It deprecates cluttering the name with bits of information that help disambiguating names, but that are not human name information. The RFC also advises that other forenames, or sets of initials, should be included in alternative common names. It suggests that personal title information should be held, if required, in a separate personalTitle attribute.

In practice, a huge variety of person name formats are used by different organisations. It is not unusual for several formats to be used within one organisation. There are several reasons for this. First, data is often drawn from multiple sources: for example, from electronic mail lists, telephone lists, and personnel records. Furthermore, an organisation may choose to devolve directory maintenance to its departments, each running different databases. These sources may not represent names the same way: some may only offer initials while others have full forenames. The way people are named in a database may evolve over time: for example, new entries in the UCL telephone directory database are created with full forename, but old entries are mostly initials and surname only.

A summary of the person name data is given in Table 4.10. The sample consists of 4011 entries from 102 organisations in five countries. Given that organisations have free rein over how their person entries are named, the 102 organisations is the most significant measure of the diversity of the sample.

Table 4.11 gives a detailed breakdown of the most prevalent RDN forms used in each of the five countries. There is a great variety of name formats. The square brackets used in formats J and O indicate that the enclosed dot character is sometimes included, sometimes not.

There is little consistency either within or between countries. Only one format, Forename Surname (code A), is use in all countries. Some formats are widely used in one country, but little
4.6. PERSON NAMES

<table>
<thead>
<tr>
<th>Country</th>
<th>No of person entries</th>
<th>Ave names per entry</th>
<th>No. of orgs</th>
</tr>
</thead>
<tbody>
<tr>
<td>UK</td>
<td>2565</td>
<td>1.61</td>
<td>44</td>
</tr>
<tr>
<td>US</td>
<td>949</td>
<td>1.34</td>
<td>20</td>
</tr>
<tr>
<td>DE</td>
<td>109</td>
<td>1.08</td>
<td>13</td>
</tr>
<tr>
<td>ES</td>
<td>344</td>
<td>1.83</td>
<td>15</td>
</tr>
<tr>
<td>NL</td>
<td>44</td>
<td>1.11</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 4.10: Summary of person entry data

<table>
<thead>
<tr>
<th>Code letter</th>
<th>Example name</th>
<th>Each category as % of name input</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>UK</td>
</tr>
<tr>
<td>A</td>
<td>Alan Smith</td>
<td>21.64</td>
</tr>
<tr>
<td>B</td>
<td>Alan B Smith</td>
<td>10.80</td>
</tr>
<tr>
<td>C</td>
<td>Alan B. Smith</td>
<td>0.04</td>
</tr>
<tr>
<td>D</td>
<td>A B Smith</td>
<td>21.75</td>
</tr>
<tr>
<td>E</td>
<td>A Smith</td>
<td>10.45</td>
</tr>
<tr>
<td>F</td>
<td>Alan Bruce Smith</td>
<td>4.58</td>
</tr>
<tr>
<td>G</td>
<td>Smith, A...</td>
<td>0.00</td>
</tr>
<tr>
<td>H</td>
<td>A.B. Smith</td>
<td>7.56</td>
</tr>
<tr>
<td>I</td>
<td>A.B.Smith</td>
<td>5.15</td>
</tr>
<tr>
<td>J</td>
<td>Mr[,] A...Smith</td>
<td>4.44</td>
</tr>
<tr>
<td>K</td>
<td>A. Smith</td>
<td>1.64</td>
</tr>
<tr>
<td>L</td>
<td>A.Smith</td>
<td>3.24</td>
</tr>
<tr>
<td>M</td>
<td>uid</td>
<td>3.82</td>
</tr>
<tr>
<td>N</td>
<td>Alan (A.B.) Smith</td>
<td>2.53</td>
</tr>
<tr>
<td>O</td>
<td>A...Smith jr[,]</td>
<td>0.00</td>
</tr>
<tr>
<td>P</td>
<td>A...Smith (uid)</td>
<td>0.00</td>
</tr>
<tr>
<td>Q</td>
<td>Smith ...</td>
<td>0.58</td>
</tr>
</tbody>
</table>

Table 4.11: Summary of person name query formats
used in other countries: e.g. formats B and C together are used over for over 60% of RDNs in the US but for less than 3% of RDNs in Germany, Spain and The Netherlands.

Most of the names conform to a general format of starting with a forename or initial and ending with a surname. There are a number of exceptions. One organisation used userids for most of its person name RDNs. There is some use of surname-first names (codes G and Q). One organisation included a forename, but repeated it as an initial in parentheses (code N). There is some use in the US of names ending in “Jr”, “Jr.” or a generational qualifier as in “Alan Smith III”, or “Alan Smith,III”. Personal titles are used in 4.44% of UK person entry RDNs, but little elsewhere.

One reason for strange RDN formats is that a key function of RDNs is to disambiguate an entry from all its sibling entries. The need to do this sometimes conflicts with the desire to choose "natural" formats such as forename(s) and surname, or initial(s) and surname: for example, there might be two “Alan Smith”s in a department. RFC 1617 recommends that directory administrators should use multi-attribute RDNs in such cases, so that the RDN consists of both a commonName attribute (with a natural value such as “Alan Smith”) and an attribute such as userid or uniqueIdentifier.

However a number of organisations have solved this problem of name clashes by adding extra characters to the commonName RDN value, thus enforcing uniqueness. The following are examples of how administrators have achieved unique names. Unless stated to the contrary, the extra distinguishing information is added only to those entries that need to be disambiguated from like-named entries.

- One organisation used computer userids for almost all person entry RDNs.
- Three organisations have appended a user identifier in parentheses to the person’s name: e.g. “Alan Smith (u12345)”. 
- Two organisations have appended an integer in parentheses after the person’s name: e.g. “Alan Smith (2)”. 
- One organisation appended an integer directly to the person’s name: e.g. “Alan Smith2”.
- One organisation appended an eight-letter code after the name: e.g. “Alan Smith CNT12345”.
- One organisation used surnames as RDNs and appended an integer in parentheses to distinguish between the inevitable collisions: e.g. “Smith (3)”. 
- One organisation appended telephone number and site information to all its person entry RDNs: e.g. “Smith, Alan, 609-123-4567, c-site g127a”
4.6. PERSON NAMES

There are many possible name formats when including initials, arising from the inclusion or omission of spaces and dots. One possible source of confusion arises when administrators concatenate initials into a single token: e.g. "Alan Bruce Smith" is rendered as "AB Smith". RFC 1617 recommends that this form is avoided because of the difficulty of interpreting whether this token is a short forename, such as "Ed", or concatenated initials. However, in practice there appears to be little confusion. First, concatenated initials were always upper case in my sample, whereas names are generally just capitalised. Second, leading two letter tokens (other than personal titles) are predominantly concatenated initials (8 out of 10 cases in the UK), while leading tokens of three or more letters are almost always names (555 out of 556 cases in the UK).

Alternative names

One solution to the problem of matching person names caused by there being so many different name formats is for administrators to include additional name formats as alternative names. For example, an entry named "A B Smith" might have alternatives of "Alan Smith" and "Smith, Alan B".

<table>
<thead>
<tr>
<th>Country</th>
<th>Percentage entries including 1+ alternative names</th>
<th>Percentage orgs with some entries including an alt. name</th>
</tr>
</thead>
<tbody>
<tr>
<td>UK</td>
<td>47.76</td>
<td>56.82</td>
</tr>
<tr>
<td>US</td>
<td>33.30</td>
<td>45.00</td>
</tr>
<tr>
<td>DE</td>
<td>8.26</td>
<td>15.38</td>
</tr>
<tr>
<td>ES</td>
<td>94.77</td>
<td>73.33</td>
</tr>
<tr>
<td>NL</td>
<td>11.36</td>
<td>20.00</td>
</tr>
</tbody>
</table>

Table 4.12: Person entries including alternative common name attributes

However, we can see from Table 4.12 that providing alternative person names appears from the sample to be far from systematic. Spain provided the only sample where more than half the entries had alternative names, although over half the UK organisations had some person entries with alternative names.

Table 4.13 presents more evidence on the use of alternative names. It shows that over half the organisations (58 out of 102) had alternative values for less than 10% of their person entries, while just over a quarter of the organisations gave alternative names for over 90% of their entries. Evidently few organisations systematically give multiple names to their person entries.

When we look at matching algorithms we will see that one way of matching human names is to try to match the leading initial and surname only, by using a filter of the form (if we assume the Alan Smith example):
### Table 4.13: No. of organisations with varying percentages of persons with alternative names

<table>
<thead>
<tr>
<th>Country</th>
<th>No. of orgs</th>
<th>&lt;10</th>
<th>10-50</th>
<th>50-90</th>
<th>&gt;90</th>
</tr>
</thead>
<tbody>
<tr>
<td>UK</td>
<td>44</td>
<td>22</td>
<td>4</td>
<td>7</td>
<td>11</td>
</tr>
<tr>
<td>US</td>
<td>20</td>
<td>12</td>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>DE</td>
<td>13</td>
<td>11</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>ES</td>
<td>15</td>
<td>5</td>
<td>0</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>NL</td>
<td>10</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 4.13: No. of organisations with varying percentages of persons with alternative names

### Table 4.14: Percentage entries including 1+ common names in "A...Smith" format

<table>
<thead>
<tr>
<th>Country</th>
<th>Percentage entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>UK</td>
<td>94.66</td>
</tr>
<tr>
<td>US</td>
<td>96.73</td>
</tr>
<tr>
<td>DE</td>
<td>89.91</td>
</tr>
<tr>
<td>ES</td>
<td>94.77</td>
</tr>
<tr>
<td>NL</td>
<td>59.09</td>
</tr>
</tbody>
</table>

Table 4.14: Percentage with at least one common name conforming to “A...Smith” format
4.6. PERSON NAMES

cn=a\*smith

It then becomes irrelevant whether the target entry has forenames or initials, and if initials are used, whether they are followed by dots and/or spaces. This strategy works so long as at least one of the person name forms in the directory starts with the first forename or initial and ends with the surname. Table 4.14 shows that three of the five countries (and the three with the biggest samples) have more than 94% of entries conforming to this format; the average for the sample as a whole is 94.64%. My view is that matching names in the directory would be greatly facilitated if this figure was 100%.

Userids

As we can see from Chapter 3, a small amount of user input for person names is userids. It is not surprising that some input is of this form given that directory services such as Whois and Finger can search on userids.

Furthermore, the UFN querying algorithm defined in RFC 1781 [Kil95b] uses the userid attribute in its filters for finding persons entries. However, do many directory entries for people contain a userid attribute? Some evidence on this is presented in Table 4.15.

<table>
<thead>
<tr>
<th>Country</th>
<th>% entries including a userid</th>
<th>Percentage orgs with some entries including a userid</th>
</tr>
</thead>
<tbody>
<tr>
<td>UK</td>
<td>30.02</td>
<td>38.64</td>
</tr>
<tr>
<td>US</td>
<td>32.88</td>
<td>25.00</td>
</tr>
<tr>
<td>DE</td>
<td>22.02</td>
<td>30.77</td>
</tr>
<tr>
<td>ES</td>
<td>2.33</td>
<td>13.3</td>
</tr>
<tr>
<td>NL</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Table 4.15: Person entries including a userid attribute

In the UK, the US and Germany, between a fifth and a third of entries have a userid attribute, with very few of these attributes in Spain or The Netherlands. This sparse coverage limits the effectiveness of search strategies using the userid attribute.

O Apostrophe

An area where directory naming is inconsistent is for names such as O'Neill and O'Connell; some organisations include the apostrophe in directory names of this type, others omit it, while some include both forms.

I repeated the sampling exercise to find out how often these forms are used. I used DE's power search mechanism to search for all the UK's O'Neill and Oneill entries. Overall 16 organisations
had one or more entries matching either or both of these names. Eight organisations used the form without the apostrophe, seven using it for RDNs; there were 43 names in this format. Thirteen organisations used the form with the apostrophe, 12 using it for RDNs; there were 52 names in this format.

Only one organisation included both formats in their entries. Given that users enter both formats, it would help matching if more administrators included both forms.

Forenames and Their Familiar Forms

Many forenames have popular familiar forms. If directory administrators create entries with forenames rather than initials, they can either choose to use full forenames or their equivalent familiar forms. There is a potential matching problem if, say, administrators predominantly use formal names, while users enter familiar names in their queries. Table 4.16 shows how often directory administrators used the full and familiar forename forms as the first name of UK directory entries. The table is the ten most frequent names in the sample that have popular familiar forms.

<table>
<thead>
<tr>
<th>Full name(s)</th>
<th>Short names(s)</th>
<th>Freq</th>
<th>Name(s)</th>
<th>Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>David</td>
<td>Dave</td>
<td>46</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Steven or Stephen</td>
<td>Steve</td>
<td>25</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Richard</td>
<td>Rick or Dick</td>
<td>25</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Christopher,</td>
<td>Chris</td>
<td>22</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Christine or Christina</td>
<td>Andy</td>
<td>22</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Robert</td>
<td>Bob or Rob</td>
<td>20</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Michael</td>
<td>Mike</td>
<td>14</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Nicholas</td>
<td>Nick</td>
<td>9</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>James</td>
<td>Jim</td>
<td>9</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Philip</td>
<td>Phil</td>
<td>7</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.16: Use of full and familiar name forms for person entries

The table shows that most administrators eschew the use of familiar name forms; full names out-number familiar forms by nearly seven to one. Since users quite often enter familiar forms (about one time in four - see Chapter 3) it would help matching if administrators entered familiar forms in all cases where appropriate.
4.7 Notes and Caveats on Methodology

In this section I note some possible limitations of the analysis that has been described in this chapter. There are essentially two types of problem. First, there may be biases in the data analysed, causing misleading results. Second, important areas have not been studied at all, and some that have been mentioned have not been studied thoroughly; I note the issues that would benefit from further work.

An obvious possible cause of bias is that the five countries used for much of the analysis are atypical of the directory as a whole. The reasons for selecting the five countries were that:

- Each of the countries contained a non-trivial amount of data; in fact the US and the UK have the largest amount of data of any countries in the DIT.
- The countries contained data that was generally accessible; reliability in some parts of the DIT is still patchy.
- The countries were managed to some degree by national managers, and were still expanding.
- The countries were not all English-speaking. Furthermore, some use non-ASCII characters to represent some letters: for example, the German umlaut.

A more particular bias may be due to the use of UK-specific data for some more detailed analysis. The reasons for restricting the scope of some of the analysis to solely the UK were pragmatic: firstly, it limited the amount of analysis that was required; secondly, I am syntactically and semantically familiar with English names, and thus the analysis is more likely to be accurate and also can be more detailed.

Ultimately I do not believe that small biases in the data affect the arguments very much. The main aim of the chapter is to get an overall feel for the naming data in the directory, and to discern the main trends. In some cases, such as the format of person names, the main point of note is the variety of formats, rather than an overall trend common to all countries. I believe that the UK-specific analysis that I have undertaken should suggest to others the sort of problem areas that need to be considered in other countries. A brief examination of the data reveals that this is certainly the case. For example, the Spanish department data has four different forms for the Spanish word for department, the short forms of which are not simple abbreviations of the longer forms. US incorporated companies often have names of the form “Ace Computer, Inc.”. However, sometimes the comma is omitted, sometimes the dot is omitted, sometimes the word “Incorporated” is used instead of the abbreviation, but more often than not the term is omitted. There are numerous other examples of these subtleties in the data.

The sins of omission are probably more serious than those of commission. I have included nothing on locality entries. There are two reasons for this. First, since DE does not allow locality name input, I have no user input to compare with locality names in the directory. Second, the use of localities in the directory is still experimental. Their final role will only emerge as national
CHAPTER 4. DATA IN DIRECTORY

Public network operators take over the running of the directory and adopt naming hierarchies, including locality entries, designed to cater for millions of organisation entries.

Another important area that I have scarcely touched upon is the issue of the use of national languages in an international directory. How do users fluent only in Swahili query the Polish DIT, if the DIT has entries in local language only? A solution of sorts is in place for country entries, as the standard mandates the use of ISO country codes, which are, by definition, language independent. The rider "of sorts" is necessary as in Chapter 3 we see that many users prefer to query the directory using full country names rather than country codes. One possibility is for organisations to include alternative names in other languages, and particularly in English, for their organisation and its departments. The brief analysis I have included in this chapter shows that this is not done very much and rarely done systematically. Even if alternative values were added, there is currently no way of tagging information so that the user is only shown attributes in the correct language. The topic of language support for matching names is being considered within the X.500 standards group: some facilities should be available in the 1997 version of X.500.

4.8 Summary and Conclusions

Since the form of country names is mandated by the standard, the only issue with country names is whether the friendlyCountryName attribute is used to provide full country names as well as the official country codes. In all countries except one, this attribute is being used, often to provide country names in several different languages.

Overall, for the other types of entry, the guidance on naming in RFC 1617 is mostly being followed. Full name forms are mostly being used to name organisations and departments, with shorter name forms used as alternative names. There is a huge variety of name formats for person names, and the most popular name form differs from country to country. As we move down the DIT from the root to the leaves the average number of alternative values per entry drops from over five friendlyCountryName values to about one and a half commonName values for person entries. There is clearly scope for many more alternative values, to reflect better the variety of input forms that we noted in Chapter 3. However, despite the large variety of name forms, we can still often match the majority of entries using intelligent matching techniques. For example, over 94% of all person entries have at least one name form that would be matched by a filter of the form

\[ cn=initial*surname \]

We have also noted a number of potential problems with name forms. These include the inconsistencies of form: different department names being used for departments performing essentially the same function; the terms "and", "&" and "+" all being used as a conjunction in department names; the use or omission of apostrophe characters; the presence of various stop list words and punctuation characters; the use of abbreviated names that are not always substrings
of the longer name forms; and several others.

There are sometimes other attributes that contain name information, and these can also be used for searching the directory. For example, some organisation entries had `associatedDomain` attributes that included values that were not present as alternative values of the `organizationName` attribute. Similarly, many person entries include a `userid` attribute that can be searched as well as the `commonName` and `surname` attributes.
Chapter 5

Experiments with Name Matching

5.1 Introduction

This chapter contains an analysis of some possible name matching algorithms. We are solely concerned here with the matching of input against directory names: i.e., can we find the correct entry X from a set of names given user input of Y, even when the user input does not correspond closely with the name(s) in the directory? We are not concerned here with DIT structure issues which also have a bearing on whether we can find entries: for example, we do not consider how to find organisation entries that may be three or four levels down the DIT beneath state or locality entries. We implicitly assume, unless stated otherwise, that we are comparing a user's input with the full set of directory names. The implied DIT has country entries beneath the root, organisation entries beneath country entries, optional organisational units beneath organisation entries and person entries below either organisation or department entries. These assumptions are currently true for the majority of the DIT.

5.1.1 Outline of the Chapter

We examine the best way of matching user input to directory entries on an input category by category basis. While the general approach in each category is similar, each input category poses subtly different matching problems and the best approach varies from category to category.

The general outline of each of the sections analysing matching algorithms is as follows. Each section begins with a brief description of the query data and the database against which these queries are tested. This is followed by an analysis of some basic search filters applied to the query data. The filters analysed may be considered the basic building blocks of a directory service, and include basic exact, substring and approximate matching filters. In most cases, these filters are applied to the unmodified user input. The following characteristics of the filters are analysed:

- whether a filter finds any results;
- whether the filter finds the correct result(s);
- whether the filter finds a single result;
- how many results a filter finds on average.

There is then a discussion of any special considerations that we should take into account when designing more realistic search filters. These issues include whether we should consider trying...
to match the input against other attributes, and whether short input strings need to be treated
differently from input in general.

We then examine the effectiveness of some search strategies that might realistically be used
when building DUAs. These centre on the search filters proposed in RFC 1781, "Using the OSI
Directory to Achieve User Friendly Naming", and those used by the author's DE. Alternative
versions of these basic strategies are tried, showing how we can improve look-up performance
by tuning our algorithms to take more note of the characteristics of the data. We also examine
whether the "read then search" strategy is likely to be more efficient than a pure search-based
strategy.

The focus then turns to the hard-to-match queries. We look at the reasons why some queries
can not be matched using exact and substring matching techniques; sometimes it is simply that
information is not in the directory.

One problem is misspelled input. Many directory implementations, including Quipu, use
Soundex for approximate matching. Soundex is designed as a phonetic matching algorithm: how
well does it cope with the misspelled input in my samples. Another problem within the scope of
approximate matching is how to match correctly spelled user input which differs in form from the
name in the directory. Quipu offers some assistance with this problem, but it is likely that different
implementations will tackle this problem with diverse approaches. The problem of approximate
matching is discussed in Chapter 6.

As approximate matching is not standardised, particularly with respect to how DSAs handle
mismatches of name form, there are some advantages for DUAs to do some work themselves
to match hard-to-find entries. Three techniques are of particular interest. First, we look at the
effect of stripping stop-list words from the input. Second, although person names do not generally
contain stop-list words, person name input can usefully be normalised into a form consisting of
a leading initial and a surname in order to facilitate matching. Third, we examine whether
matching truncated input words is a viable approach to matching inexact input.

We then look at any issues that are specific to the input category. Finally there is a summary
of the main findings of the section.

5.1.2 Some Notes on Methodology

The tests were all made using the dish(1) program, the generic DUA provided with the Quipu
system. dish, which stands for DIrectory SHell, was an ideal tool for my testing for two reasons.

- dish allows the user to specify any valid X.500 operation. This contrasts with a DUA such
  as DE which only uses a small subset of X.500 operations.

- dish allows operation parameters to be specified as command line arguments. Its results
  are written crudely formatted to standard output, which can then be processed further by
  other UNIX tools such as awk, sed, grep and wc.
5.2. COUNTRY NAMES

Unfortunately, I discovered during the course of my experiments that *disp* does not always correctly translate a string formatted search filter into the correct X.500 search filter; the problems occur because Quipu is over zealous about removing spaces from strings in filter items. I tried to correct these problems, but was advised by Quipu's principal code developers that the errors were very difficult to solve. As a consequence, I was unable to undertake a few tests that I wanted to do.

A considerable number of search filters are presented in the following sections. However, the filters presented are simplified versions of what is needed in practice. In each case, the filters should have an additional ANDed component, specifying the object class of the required entries. For example, an exact match filter on the *commonName* attribute for the name "Fred Bloggs" would be in full:

\[(cn=Fred Bloggs) \text{ AND } (objectclass=person)\]

This would prevent matching any other type of object named by a *commonName* attribute when searching for a person.

Another feature of the tests is that none of them, except where explicitly stated otherwise, have their results restricted by any size or time limits. This is true both for DSA administrative limits and for DUA service controls.

Approximate matching is mentioned frequently throughout the chapter. The algorithm used is Knuth’s Soundex [Knu73], which is supplied as part of the Quipu implementation. The Soundex algorithm, and several alternative algorithms, are described fully in Chapter 6. Quipu, however, uses some non-standard settings of Soundex. These settings, and the reasoning behind them, are discussed in Appendix D.

5.2 Country Names

The analysis of country name matching is based on experiments using the input from the four sources of query data described in Chapter 3. These queries are matched against a copy of the country data held in the DIT; this is described in Chapter 4.

For simplicity, the sets of input data are treated as a single data set. The query data consists of all the country name input that was recognisably a country name, excluding the following categories:

- “Input” that was merely an accepted default value.
- Queries with asterisks; the user has indicated the type of substring match they require.
- Queries with superfluous punctuation characters, such as backslashes and quotation marks. These characters could be detected and removed from user input; their presence is likely to inhibit matching. Note that input with dots after initials is NOT excluded.
CHAPTER 5. EXPERIMENTS WITH NAME MATCHING

The resulting sample of 7481 queries forms the basis for the experiments described in this section. 7390 (98.78%) of the queries were for countries with entries in the directory.

5.2.1 The effectiveness of some basic filters

I compared five basic filters to test their effectiveness at finding the required entries. These filters could be considered the raw facilities provided by X.500; each filter is applied individually to the user input. The five filters are as follows (each filter type is tagged with an abbreviated name in parentheses and these tag names are used frequently in the subsequent text and tables):

Country Name - exact match (cExact): This matches input that is a country name RDN.

Friendly Country Name - exact match (coExact): This matches input that is exactly equivalent to one of the friendlyCountryName values.

Friendly Country Name - leading substring (coLeadSub): This filter matches queries where the input is abbreviated: e.g. “can” for Canada, and “united states” for United States of America.

Friendly Country Name - any substring (coAnySub): This filter is useful for matching input that corresponds to part of a longer, formal name: e.g. “Britain” matches Great Britain and “States” matches United States.

Friendly Country Name - approximate match (coApprox): This filter is useful for matching misspelled input, and several other cases where a user’s input does not precisely match the form in the directory.

The analysis of these filters is shown in Table 5.1.

<table>
<thead>
<tr>
<th>Filter type</th>
<th>% matched by filter</th>
<th>Mean no. res.</th>
<th>Med. no. res.</th>
<th>Max. no. res.</th>
<th>% queries with single res.</th>
<th>% queries with correct res.</th>
<th>% results correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>cExact</td>
<td>42.44</td>
<td>0.42</td>
<td>0</td>
<td>1</td>
<td>42.44</td>
<td>42.44</td>
<td>100.00</td>
</tr>
<tr>
<td>coExact</td>
<td>85.98</td>
<td>0.86</td>
<td>1</td>
<td>1</td>
<td>85.98</td>
<td>85.98</td>
<td>100.00</td>
</tr>
<tr>
<td>coLeadSub</td>
<td>93.60</td>
<td>0.96</td>
<td>1</td>
<td>2</td>
<td>90.88</td>
<td>92.61</td>
<td>98.94</td>
</tr>
<tr>
<td>coAnySub</td>
<td>96.28</td>
<td>1.59</td>
<td>1</td>
<td>10</td>
<td>77.49</td>
<td>94.49</td>
<td>98.14</td>
</tr>
<tr>
<td>coApprox</td>
<td>97.41</td>
<td>1.68</td>
<td>2</td>
<td>11</td>
<td>43.15</td>
<td>96.15</td>
<td>98.71</td>
</tr>
</tbody>
</table>

Table 5.1: Country name matching using basic filters

When interpreting these results, the reader should remember that this analysis is solely for user entered names; in practice, as is shown for the UCL services in Chapter 3, one would expect country names to be defaulted more often than entered.
5.2. COUNTRY NAMES

Having registered that caveat, we see, as one would expect, that as we loosen the filter from exact to approximate matching that each filter finds progressively more matches. 85% of user input exactly matches either a countryName or friendlyCountryName attribute value in a country entry. Substring matching finds matches for a further 10% of queries, and approximate matching finds matches for another 1% of queries. 2.45% of queries found no matches: about half of these were for entries not in the directory and half were for countries that had directory entries but were not found. We investigate the reasons for entries not being found in Section 5.2.3.

The relatively small number of country entries in the directory (when compared, for example, with the number of people in an organisation) means that the number of results returned is usually very low. The two exact matching filters never return more than one result. Only the approximate matching filter returns a median of more than one result; its median result set is two results. The maximum number of results returned with any filter is ten and eleven for coAnySub and coApprox matching respectively; however result sets of this size are only achieved when matching short input strings such as "IN" and "SE" that could be matched exactly. While the coAnySub and coApprox filters are the most effective at finding results, their tendency to return more results means that they return a single result less often; approximate matching finds a single result less half the time.

All the filter types usually return the correct result, if they return results at all. Even the worst filter in this respect, coAnySub, got the correct result in 98.14% cases when a result was returned.

It is fair to conclude that there are no real surprises in these results. The more effective a filter is at finding the correct entry, the more likely it is to find other entries as well. However, we can achieve better results by doing two things. First, DUAs can use different filters according to various characteristics of the input string: e.g., if the input is two characters, it is reasonable to guess that the input is a country code and can thus be matched against the country RDN. Second, DUAs can use two or more filter items in combination; the various combinatorial options are described in Chapter 2.

5.2.2 More Sophisticated Search Strategies

In this section we examine six further search strategies to explore the possibilities described at the end of the previous section. Some of the examples are included to illustrate problems that have to be solved. One of the strategies is that currently implemented in DE. Another is that described in RFC 1781, which describes the UFN search algorithm. Finally, having learned from this analysis, we include some filter combinations that are demonstrably better than those specified or implemented.

The six filters are as follows:

**Single filter based on length (SFL):** This filter uses exact matching on the country code if the input is two letters, and uses any substring matching on all other input. The pseudo
CHAPTER 5. EXPERIMENTS WITH NAME MATCHING

code for this filter is as follows:

```c
if (input length == 2)
    cExact
else
    coAnySub
```

**UFN:** This is the filter suggested in RFC1781, which describes the UFN matching algorithm. The pseudo code for this filter is as follows:

```c
if (input length == 2)
    cExact OR coExact
else
    coAnySub OR coApprox
```

**DE:** This is the search strategy as implemented in DE. It uses a sequence of searches which continues until a result is found or the filters have all been tried. The pseudo code for the DE search strategy is as follows:

```c
if (input length == 2)
    cExact, coExact, coLeadSub, coAnySub, coApprox
else
    coExact, coLeadSub, coAnySub, coApprox
```

**More efficient DE (MEDE):** From the analysis it is clear that some filters are unhelpful: for example, approximate matching on two letter input. This strategy omits some of the usual filters and should be more efficient. The pseudo code for this MEDE search strategy is as follows:

```c
if (input length == 2)
    cExact, coExact
else
    coExact, coAnySub, coApprox
```

**SFL filter with name-to-RDN mapping table:** The input is pre-processed using a ten entry name-to-RDN mapping table of the form described in Chapter 3, and then used in conjunction with the SFL filter.

**MEDE with mapping table:** A ten entry mapping table is used in conjunction with the MEDE search strategy.

The analysis of these search strategies is shown in Table 5.2.

The SFL filter is included mainly for illustrative purposes. This filter has two principal deficiencies. First, and the main problem with the data sets used here, is that it treats user input
5.2. COUNTRY NAMES

Table 5.2: Country name matching using more sophisticated search filters

<table>
<thead>
<tr>
<th>Filter type</th>
<th>% matched by filter</th>
<th>Mean no. res.</th>
<th>Med. no. res.</th>
<th>Max. no. res.</th>
<th>% queries with single res.</th>
<th>% queries with correct res.</th>
<th>% results correct</th>
<th>Mean no. of searches</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFL</td>
<td>81.06</td>
<td>0.81</td>
<td>1</td>
<td>3</td>
<td>80.68</td>
<td>81.02</td>
<td>99.95</td>
<td>1</td>
</tr>
<tr>
<td>UFN</td>
<td>97.53</td>
<td>1.18</td>
<td>1</td>
<td>6</td>
<td>77.69</td>
<td>97.41</td>
<td>99.88</td>
<td>1</td>
</tr>
<tr>
<td>DE</td>
<td>97.91</td>
<td>0.99</td>
<td>1</td>
<td>6</td>
<td>97.50</td>
<td>97.57</td>
<td>99.65</td>
<td>1.35</td>
</tr>
<tr>
<td>MEDE</td>
<td>97.53</td>
<td>0.98</td>
<td>1</td>
<td>6</td>
<td>97.15</td>
<td>97.38</td>
<td>99.85</td>
<td>1.29</td>
</tr>
<tr>
<td>SFL + map table</td>
<td>97.02</td>
<td>0.97</td>
<td>1</td>
<td>3</td>
<td>96.64</td>
<td>96.98</td>
<td>99.96</td>
<td>1</td>
</tr>
<tr>
<td>MEDE + map table</td>
<td>98.37</td>
<td>0.99</td>
<td>1</td>
<td>6</td>
<td>97.99</td>
<td>98.22</td>
<td>99.85</td>
<td>1.06</td>
</tr>
</tbody>
</table>

of "uk" as a two-letter country code; since it does not try any alternative filters, input of "uk" fails to match. Second, there is no provision for mis-spelled input; since the rate of misspelling with country names is quite low (see Chapter 3), this may not be regarded as a serious problem.

All the remaining filters give good results, with the following characteristics:

- all find results in more than 97% queries
- none return a lot of results;
- all are better than 99.5% accurate in the results they do return.

However, there are some notable differences between the filters. There are two possible drawbacks with the UFN filter. It always uses a compound filter, which may be more expensive for DSAs to evaluate than filters with a single filter item. Second, the UFN filter is relatively poor at returning a single result; 77% of queries against 97% for the best filters. The reason for this is that the UFN algorithm always uses approximate matching for any input of other than two characters, and, as we saw, approximate matching tends to return more results. A consequence of this, foreseen in RFC 1781, is that the user (or probably the user's DUA) has to filter out approximate matches if exact matches exist in the result set.

The existing DE implementation gives similar results, except that it finds a single match (almost always correct) in 97.50% cases. The cost of this greater accuracy is that DE can use up to five filters when evaluating a query, although the average is only 1.35 filters.

Close inspection of the existing DE implementation reveals some possible inefficiencies. For example, it makes little sense to try approx matching on two letter codes if exact matching has failed; the failure almost always indicates that the country does not have a directory entry, and thus the consequence of approximate matching is spurious results. The MEDE algorithm tries to trim some of the less useful filters used by the original DE. However, the analysis shows that
the MEDE algorithm gives very similar results to the existing DE. MEDE gives slightly fewer matches, is slightly more accurate, and uses fewer filters (average of 1.29) per search than DE.

In Chapter 3 we noted that by using small name-to-RDN mapping tables we could transform a substantial proportion of non-RDN input into RDN form. The potential usefulness of this is that if we know an entry's RDN we can read the entry, rather than having to search for it, and this is presumed to be more efficient. Even if we continue to search for entries, the mapping table has done part of the query resolution work.

I tried using mapping tables with algorithms SFL and MEDE. The main benefit using the SFL algorithm with a mapping table is that more entries are found; the UK is now no longer mis-treated as an RDN. There are two benefits to the MEDE algorithm. First, more entries are found: it finds those with input of "France" - an explanation is given in Section 5.2.3. Second, the average number of filters required is reduced from 1.29 to 1.06.

In fact we can be much more economical with our directory operations than this. Unless the country entry has attributes that are useful in their own right, the sole reason for resolving the country name input to the corresponding entry is to get the country name RDN. This RDN can then be used as the base object of operations to find, usually, an organisation entry. If the user input is the two-letter country code, no country name resolution is required, and we can start the name resolution process by trying to match the organisation name input. The problem is knowing exactly when the user's country name input is an RDN.

The "read and search" technique, described by Afifi and Huitema in [AH92], and refined by Woermann and Pacchioni as the AFRO algorithm [WP94], omits the country name resolution phase when possible. The effect of omitting this resolution phase on the number of operations required is shown in Table 5.3 for three of the strategies described above.

<table>
<thead>
<tr>
<th>Filter type</th>
<th>Mean no. of ops.</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>always try to try to avoid resolve name name resolution</td>
<td></td>
</tr>
<tr>
<td>DE</td>
<td>1.35</td>
<td>0.91</td>
</tr>
<tr>
<td>SFL + map</td>
<td>1.00</td>
<td>0.09</td>
</tr>
<tr>
<td>MEDE + map</td>
<td>1.06</td>
<td>0.18</td>
</tr>
</tbody>
</table>

Table 5.3: The effect of trying to omit country name resolution by using the read and search strategy on the number of operations required to resolve a country name

One problem with this approach arises with two-letter country names that are not country RDNs: the vast majority of such cases in my sample were the name "UK". The following example should help to explain this. If we are trying to resolve the query

S Kille, ISODE, UK
5.2. COUNTRY NAMES

using the AFRO algorithm we would first try a read operation using a purported name of:

cn=S Kille, o=ISODE, c=UK

This would fail and indicate that no part of the purported name matched any directory components. We would then have to resolve the “UK” part of the name using a search operation. Since we could have done this in the first place, the wrong assumption that “UK” is an RDN means that name resolution requires an additional operation - a failed read - over the always search to resolve the name strategy.

We can solve this problem and increase the effectiveness of this strategy of not explicitly resolving country names by also using a mapping table. The SFL algorithm used with a mapping table on average required one search operation per eleven queries to resolve country names; the MEDE strategy, the most successful at finding the correct results, only required an average of 0.18 searches per country name.

The conclusion is that by using even a small name-to-RDN mapping table, we can eliminate the majority of country name resolution operations.

5.2.3 Reasons for Failing to Find an Entry

2.45% of users’ queries failed to match any countries in the directory, using any of the filters described earlier. The principal reasons for matching failure were:

Country not in directory: The user entered a name for a country that did not have a corresponding directory entry; only 38 countries had entries at the time the directory data was analysed.

No friendlyCountryName attribute: The French country entry did not have a friendlyCountryName attribute, and thus a query of “france” did not match the entry for c=FR.

Typographical errors: While approximate matching found most typographical errors or other misspellings, some were not matched. See Section 5.2.4

Format not in directory: The user’s input corresponded to a country in the directory, but in a form not held in the directory. For example, the user entered “swedish” for Sweden, or “w.germany” for Germany (or, as it is in full, the Federal Republic of Germany).

The relative frequency of these matching failures for the sample data is shown in Table 5.4. The query “france” resulted in over half those that queries that were for a country with an entry in the directory but that could not be found. While the c=FR entry should ideally include a friendlyCountryName attribute with a value “France” (and did in the past), its absence does not prevent the c=FR entry being found if a mapping table is used. The inclusion of “France” in the mapping table is the main reason why the MEDE algorithm with a mapping table is more successful than the MEDE algorithm without a mapping table.
CHAPTER 5. EXPERIMENTS WITH NAME MATCHING

<table>
<thead>
<tr>
<th>Reason</th>
<th>per cent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Country not in directory</td>
<td>49.46</td>
</tr>
<tr>
<td>No friendly Country Name</td>
<td>34.24</td>
</tr>
<tr>
<td>Typographical errors</td>
<td>15.22</td>
</tr>
<tr>
<td>Format not in directory</td>
<td>1.09</td>
</tr>
</tbody>
</table>

Table 5.4: Reasons for country name matching failures

Almost all the other entries that were not found were due to misspellings, and these appeared to be predominantly, if not all, typographical errors. We can now look how successful approximate matching is at finding misspelled input.

5.2.4 How Useful is Approximate Matching of Country Names?

We noted in Chapter 3 that misspelling of country name input is less of a problem than for other types of input. Overall, there were 50 cases in my sample queries where a user made a spelling mistake for a country, where the country had an entry in the directory; this is 0.67% of the total sample. The success of approximate matching with misspelled input is shown in Table 5.5.

<table>
<thead>
<tr>
<th>Outcome of approximate match</th>
<th>Freq. as %age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Found correct entry</td>
<td>58.0</td>
</tr>
<tr>
<td>Found incorrect entry</td>
<td>12.0</td>
</tr>
<tr>
<td>Failed to find anything</td>
<td>30.0</td>
</tr>
</tbody>
</table>

Table 5.5: Country name spelling mistakes and approximate matching

Approximate matching found the correct entry in well over half the cases where the input was misspelled.

5.2.5 Summary for Country Name Matching

The small set of country names in the directory means that none of the matching techniques produced a lot of entries. The main points of this section are:

- Several techniques have been demonstrated that return small result sets and give accurate results.

- Input of two characters should only be exactly matched; substring and approximate matching on short input produces spurious matches.
5.3. ORGANISATION NAMES

- Even small name-to-RDN mapping tables can reduce the number of operations required to resolve country names.
- The read and search technique can reduce the number of operations on the directory.
- The read and search technique, if used with mapping tables, can almost eliminate the need for country name matching.

5.3 Organisation Names

The analysis of organisation name matching is based on experiments using the input from the four sources of query data described in Chapter 3. These queries are matched against a copy of the UK organisation data held in the DIT; this is described in Chapter 4.

For simplicity, the sets of input data are treated as a single data set. The query data consists of all the UK organisation name input that was recognisably an organisation name, although the following categories are excluded:

- "Input" that was merely an accepted default value.
- Queries with asterisks: the user has indicated the type of substring match they require.
- Queries with superfluous punctuation characters, such as backslashes and quotation marks. These characters could be detected and removed from user input; their presence is likely to inhibit matching. Note that input with dots after initials is NOT excluded.

The resulting sample of 4158 queries forms the basis for the experiments described in this section. 3573 (85.93%) of the queries were for organisations with entries in the directory immediately beneath the GB node.

5.3.1 The effectiveness of some basic organisation matching filters

I compared four basic filters to test their effectiveness at finding the required entries. These filters could be considered the raw facilities provided by X.500; each filter is applied individually to the user input. I have also included an analysis of how often user input exactly matched the corresponding organisation RDN. This case and the four filters are as follows (each filter type is tagged with an abbreviated name in parentheses and these tag names are used frequently in the subsequent text and tables):

**Organisation Name input is RDN** \((oRDN:)\): The user input exactly matches the entry's RDN.

**Organisation Name - exact match** \((oExact):\) This matches input that is exactly equivalent to one of the *organizationName* values.
CHAPTER 5. EXPERIMENTS WITH NAME MATCHING

**Organisation Name - leading substring (oLeadSub)**: This is useful for queries where the input is abbreviated: e.g. "Cambridge" for Cambridge University, or "BT" for BT Plc.

**Organisation Name - any substring (oAnySub)**: This filter is useful for matching input that corresponds to part of a longer, formal name: e.g. "John Moores" matches Liverpool John Moores University.

**Organisation Name - approximate match (oApprox)**: This filter is useful for matching misspelled input, and several other cases where a user's input does not precisely match the form in the directory.

The analysis of these filters is shown in Table 5.6.

<table>
<thead>
<tr>
<th>Filter type</th>
<th>% matched by filter</th>
<th>Mean no. res.</th>
<th>Med. no. res.</th>
<th>Max. no. res.</th>
<th>% queries with single res.</th>
<th>% queries with correct res.</th>
<th>% results correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>oRDN</td>
<td>27.34</td>
<td>0.27</td>
<td>0</td>
<td>1</td>
<td>27.34</td>
<td>27.34</td>
<td>100.00</td>
</tr>
<tr>
<td>oExact</td>
<td>57.05</td>
<td>0.57</td>
<td>1</td>
<td>2</td>
<td>56.90</td>
<td>57.05</td>
<td>100.00</td>
</tr>
<tr>
<td>oLeadSub</td>
<td>69.89</td>
<td>0.83</td>
<td>1</td>
<td>4</td>
<td>59.96</td>
<td>69.65</td>
<td>99.66</td>
</tr>
<tr>
<td>oAnySub</td>
<td>75.69</td>
<td>1.02</td>
<td>1</td>
<td>47</td>
<td>67.72</td>
<td>75.40</td>
<td>99.62</td>
</tr>
<tr>
<td>oApprox</td>
<td>83.59</td>
<td>1.80</td>
<td>1</td>
<td>99</td>
<td>55.99</td>
<td>80.18</td>
<td>95.91</td>
</tr>
</tbody>
</table>

Table 5.6: Breakdown of organisation name matching by filter

When interpreting these results, the reader should remember that this analysis is solely for user entered names; in practice, as is shown for the UCL services in Chapter 3, one would expect organisation name input to be defaulted to the users' organisation more often than entered. There is an analysis of some of the better search strategies applied to a set of queries (including the acceptance of the default value) made by UCL users in Section 5.3.5. It is also important to note that since only 85% of the queries are for organisations which have an entry in the directory, a completely successful matching algorithm can only find the correct entry 85% of the time. Another requirement of a good matching algorithm is that it should not match the wrong entries: there is some discussion of this issue in Section 5.3.7.

The analysis in Table 5.6 shows that while just over a quarter of the user input was an organisation RDN, 57% of the queries were matched by a oExact filter. Clearly the use of alternative names for organisations provides very effective assistance for name matching. Each of the remaining three filters finds progressively more matches, with approximate matching finding at least one matching entry in over 83% of all queries. We examine the reasons why the remaining 17% of queries did not match any entries in Section 5.3.6: many of these queries are for organisations lacking a directory entry, but some queries fail for other reasons.

Although one would anticipate there to be thousands of organisation entries beneath the country node in a fully populated DIT, the DIT had only 143 organisation entries beneath the
5.3. ORGANISATION NAMES

GB country entry at the time of writing (June 1995). This means that the number of results returned is usually very low, irrespective of the filter. However the \textit{oAnySub} and \textit{oApprox} filters can both return large result sets given certain input; the characteristics of input that is likely to produce large result sets is discussed in Section 5.3.2. While the \textit{oApprox} filter is the most effective at finding results, its tendency to return more results (the correct one plus one or more spurious matches) means that it returns a single result less often; \textit{oApprox} matching found a match in almost 50\% more queries than did \textit{oExact} matching, but found a single result in fewer cases.

All the filter types usually return the correct result, although the \textit{oApprox} filter was worse than the other filters.

With these basic matching characteristics in mind, we can go on to look at some more sophisticated filters and search strategies. Before we do this, we will examine two possible refinements to matching strategies. First, we will look at the case for treating short input differently; this is suggested in the description of the UFN algorithm in RFC 1781. Second, we will look at the possibility of matching using the \textit{associatedDomain} attribute, as we noted in Chapter 3 that approximately 80\% of single token queries were organisation domain names.

5.3.2 Very Short Organisation Name Input

It should be obvious to the reader that the shorter the input string, the greater the likelihood that a substring match (particularly an \textit{any substring} match) will be made. Depending on the characteristics of the approximate matching algorithm used, it is also likely that short strings will mean relatively large result sets for approximate matching. With this in mind, the suggestion is made in RFC 1781 that:

There may be some improvements [to the filters] with respect to very short keys. Not making approximate or substring matches in these cases seems sensible.

Does the data analysed in this chapter support this idea? The answer is unequivocally that it does.

Of the 12 queries that produced more than 10 matches when using \textit{oAnySub} matching, seven had three characters or fewer. Of the 49 queries that produced more than 10 matches when using \textit{oApprox} matching, 42 had three characters or fewer. In general, input of three or fewer characters resulted in appreciably more matches. For example, input of three characters or fewer on average produced 5.88 approximate matches, while the average for the sample as a whole was 1.80 matches.

Not only does \textit{oAnySub} and \textit{oApprox} matching produce more results, it often produces the wrong results. The figures in Table 5.7 show how many queries of three or fewer characters, that failed to find any entries using \textit{oExact} matching, found entries using looser types of matching. The figures are based on the DE approach of sequential filters, where a looser match is only tried
if the previous filter failed to find any results.

<table>
<thead>
<tr>
<th>Filter type</th>
<th>Correct results</th>
<th>Incorrect results</th>
</tr>
</thead>
<tbody>
<tr>
<td>oLeadSub</td>
<td>23</td>
<td>0</td>
</tr>
<tr>
<td>oAnySub</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>oApprox</td>
<td>10</td>
<td>70</td>
</tr>
</tbody>
</table>

Table 5.7: The success of non-exact matching filters at matching short organisation name queries

23 matches were made using oLeadSub matching and all were correct. Ten more matches were found using oAnySub matching of which nine failed to produce the correct results. A further 80 matches were found using oApprox matching of which 70 did not produce the correct results. For the 10 queries that did find the correct entry using oApprox matching, the average result set size was a hefty 99.0 entries, 69% of all UK organisation entries.

We should bear in mind the nature of short input when we come to devise matching algorithms that treat short input differently. We noted in Chapter 3 that if an input string is no more than three characters, there is more than a 90% probability that the query is a set of initials. Furthermore three-quarters of names that are initials have two or three letters.

5.3.3 Matching Organisation Entries using the AssociatedDomain Attribute

In this section we consider the use of the associatedDomain attribute for matching, since many single token queries are equivalent to the organisation name prefix of DNS domain names. Furthermore, we noted in Chapter 4 that while some organisations include DNS names as alternative organizationName values, some do not.

The filter I used for the experiments with matching on domain names was as follows:

```
associatedDomain=widget.*
```

where the input string is “widget”. Initial experiments showed that this filter, with the dot character after the input, was more selective than the filter without the dot. For example, it meant that input of “sheffield” matched sheffield.ac.uk but not sheffield-hallam.ac.uk. This filter is referred to as assocDomainLeadSubDot in the rest of this chapter.

Of the 1797 single token queries, 975 were matched by the above filter, while 890 were matched using oExact matching. In effect, this reflects the strength of the relative tendencies for administrators to include an associatedDomain attribute, or to include short forms of the organizationName attribute. The survey in Chapter 4 shows that while at least half of the organisations in
5.3. ORGANISATION NAMES

Each country provides alternative name values, usually short forms, the percentage of organisations including an associatedDomain attribute varies widely. While matching on associatedDomain names may be a useful and efficient technique in some parts of the directory, in others it may be ineffective. This suggests that DUAs should make searching on the associatedDomain attribute a configurable option, its use depending on which parts of the directory are being queried most frequently.

Another aspect of matching on associatedDomain names is whether any entries are found that cannot be found by other techniques. In fact, for the UK query data, matching on domain names found only four entries that were not found by exact or substring matching, and only one additional entry if approximate matching on organisation names is tried.

5.3.4 Organisation search strategies based on more sophisticated filters

We now examine seven further search strategies to explore some possible improvements. The experiments are based on queries for which there are corresponding entries in the directory. Two types of strategy are examined: two strategies are based on the UFN search algorithm, which is described in RFC 1781; the other five strategies are based on DE's search strategy of using a sequence of search filters. Some of the strategies incorporate the techniques suggested in Sections 5.3.2 and 5.3.3. The seven strategies are as follows:

A: UFN. This is the filter suggested in RFC 1781, which describes the UFN matching algorithm. This filter is as follows:

oAnySub OR oApprox

B: UFN - length 3. This is a modified form of the UFN filter, which takes into account that short input tends to produce a lot of spurious matches using oAnySub and oApprox filters, and that short input is often a DNS domain name. The pseudo code for this modified UFN filter is as follows:

if (single token AND (input length <= 3))
    oExact OR assocDomainLeadSubDot
else
    oAnySub OR oApprox

C: DE. This is the search strategy as implemented in DE. It uses a sequence of searches which continues until a result is found or the filters have all been tried. The pseudo code for the DE search strategy is as follows:

oExact, oLeadSub, oAnySub, oApprox

D: DE without oLeadSub. This search strategy is the normal DE strategy without the oLeadSub filter. Dropping this filter is feasible since the oAnySub filter will find any leading substring
matches. The interest here is to see how many fewer filters such a version of DE would use on average, and whether accuracy of matching is impaired at all. The pseudo code for this search strategy is as follows:

\[ \text{oExact, oAnySub, oApprox} \]

E: **DE - length 3.** This is the original DE strategy for input of more than three characters, but not using \(\text{oAnySub} \) and \(\text{oApprox} \) filters for short input. The pseudo code for this search strategy is as follows:

\[
\begin{align*}
\text{if (single token AND (input length \leq 3))} \\
\text{oExact, oLeadSub} \\
\text{else} \\
\text{oExact, oLeadSub, oAnySub, oApprox}
\end{align*}
\]

F: **DE + associatedDomain after oExact.** This is the same as case E, but also doing an \(\text{assocDomainLeadSubDot} \) match after trying an \(\text{oExact} \) match for single token input strings. The pseudo code for this filter is:

\[
\begin{align*}
\text{if (single token)} \\
\text{if (input length \leq 3)} \\
\text{oExact, assocDomainLeadSubDot, oLeadSub} \\
\text{else} \\
\text{oExact, assocDomainLeadSubDot, oLeadSub, oAnySub, oApprox} \\
\text{else} \\
\text{oExact, oLeadSub, oAnySub, oApprox}
\end{align*}
\]

G: **DE + associatedDomain before oExact.** As for case F, except that the associatedDomain match is tried before trying an \(\text{oExact} \) match.

The analysis of these filters is shown in Table 5.8.

There is a remarkable uniformity about many characteristics of all the seven search strategies. For example, all return a median of a single result. The percentage of queries returning the correct result (sometimes along with spurious results) hardly varies: the range is 79.32% to 80.18%. However, closer inspection of the results shows some definite trends.

First, the DE strategy of a sequence of filters (cases C to G) produces a single result significantly more often; the best UFN case (B) is 58.02% of all queries, compared with 78.07% for the best DE strategy (case D).

The maximum and average number of results returned is much reduced by not using \(\text{oAnySub} \) and \(\text{oApprox} \) matching for short (three characters or fewer) input strings, without a large drop in the number of correct results achieved.
### 5.3. ORGANISATION NAMES

<table>
<thead>
<tr>
<th>Filter code</th>
<th>% matched by filter</th>
<th>Mean no. res.</th>
<th>Med. no. res.</th>
<th>Max. no. res.</th>
<th>% queries with single res.</th>
<th>% queries with correct res.</th>
<th>% results correct</th>
<th>Mean no. of searches</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>94.07</td>
<td>2.00</td>
<td>1</td>
<td>99</td>
<td>64.43</td>
<td>93.31</td>
<td>99.20</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>92.44</td>
<td>1.36</td>
<td>1</td>
<td>12</td>
<td>66.95</td>
<td>92.30</td>
<td>99.85</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>94.07</td>
<td>1.35</td>
<td>1</td>
<td>99</td>
<td>87.18</td>
<td>92.98</td>
<td>98.84</td>
<td>1.64</td>
</tr>
<tr>
<td>D</td>
<td>94.07</td>
<td>1.39</td>
<td>1</td>
<td>99</td>
<td>85.84</td>
<td>93.25</td>
<td>99.14</td>
<td>1.46</td>
</tr>
<tr>
<td>E</td>
<td>93.09</td>
<td>1.03</td>
<td>1</td>
<td>7</td>
<td>87.10</td>
<td>92.67</td>
<td>99.55</td>
<td>1.62</td>
</tr>
<tr>
<td>F</td>
<td>93.14</td>
<td>1.01</td>
<td>1</td>
<td>7</td>
<td>89.36</td>
<td>92.72</td>
<td>99.55</td>
<td>1.74</td>
</tr>
<tr>
<td>G</td>
<td>93.14</td>
<td>1.01</td>
<td>1</td>
<td>7</td>
<td>89.53</td>
<td>92.72</td>
<td>99.55</td>
<td>1.71</td>
</tr>
</tbody>
</table>

Table 5.8: More sophisticated organisation name matching filters

The use of the `assocDomainLeadSubDot` filter for single token input led to a slightly higher proportion of queries returning a single result, a slightly greater number of filters, but little change otherwise.

Overall the results in Table 5.8 show that the adjustments to the filters to allow for short queries, and the searching on the `associatedDomain` attribute for single token strings, are basically fine tuning. The effect is not to get any more correct results – sometimes the “improved” strategies get slightly fewer correct results – but to get fewer incorrect results. The more sophisticated search strategies are more selective, at a cost of using marginally more filters to evaluate a query. We try to get a feel for the impact of simple and more complex search strategies on response times in the NameFLOW-Paradise directory in Appendix G.

#### 5.3.5 Read and Search Technique and the Impact of a Mapping Table on the UCL Query Set

While using the data set combined from several sets of queries gives us more data of actual user input to analyse, it is unrepresentative of a normal directory service where a local organisation name is defaulted, and there is a tendency for a few organisations to be queried far more heavily than others. For example, with the UCL service, Oxford, Cambridge and other London universities and colleges predominate in the queries. Some optimisations of the querying algorithm can only be assessed with a typical query load, as the essence of some optimisation is to exploit the skewed distribution of the queries.

The analysis in this section is based on the queries made by users of the UCL service. The query data comprises all queries within the UK, including those where the organisation name input is merely the default value of “University College London”. We compare six strategies which are all based on the version of DE described as case F in the previous section. It is referred to as simply DE here for brevity of exposition, and is included chiefly as a benchmark for the
further refinements of the search strategy. Each of the remaining five alternatives embellishes the first strategy in some way.

**H:** *case F in the previous section.*

**I:** *DE + read.* The organisation name input is not resolved by a search operation but is used as the base object of read, list or search operations for entries nearer the leaves of the DIT, such as person or department entries. If the organisation input is the RDN of that organisation's entry, this is an optimisation; if the input is not the RDN, then a search has to made to resolve the organisation name, and the initial read is an overhead.

**J:** *DE + read if query includes a space.* The same as case I, except that the read and search strategy is only used if the user input contains a space. Few organisation name RDNs consist of single tokens - 5 out of 143 in the UK DIT - and so a single token is an indicator that a query name is unlikely to be an RDN.

**K:** *DE + read + mapping table.* The same as case I, except that the query data is pre-processed so that the ten most common instances of non-RDN input are converted to the appropriate RDNs before matching is attempted.

**L:** *DE + read if default value.* An initial read is done only if the organisation name is the default value, otherwise the DE search strategy is used.

**M:** *DE + read if default and mapping table.* An initial read is done only if the organisation name is the default or one of the values in a ten entry mapping table. The mapping table contains the ten most common query names other than the default, irrespective of whether they are RDNs or not. For all other queries the DE search strategy is used.

<table>
<thead>
<tr>
<th>Filter code</th>
<th>Mean no. of filters / ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>1.37</td>
</tr>
<tr>
<td>I</td>
<td>0.96</td>
</tr>
<tr>
<td>J</td>
<td>0.77</td>
</tr>
<tr>
<td>K</td>
<td>0.76</td>
</tr>
<tr>
<td>L</td>
<td>0.74</td>
</tr>
<tr>
<td>M</td>
<td>0.60</td>
</tr>
</tbody>
</table>

Table 5.9: The impact of read and search and mapping tables on the number of filters used matching organisation names

An important first point is that the number of filters used to evaluate the basic DE algorithm (case H) is only 1.37 for the UCL data, whereas it was 1.74 filters for the combined set of query
5.3. ORGANISATION NAMES

The figures in Tables 5.8 and 5.9 should be used primarily for comparing techniques, rather than as absolute values.

In principle, the six search strategies behave identically except for the number of filters or other operations used to resolve the query. In practice, one of the most popular queries ("birkbeck" for Birkbeck College) did not normally find the entry for Birkbeck College, which is under the entry for the University of London. The mapping table allows us to map onto the DN for Birkbeck and find this entry.

Since the query set is predominantly the default value, which is an RDN, the "read and search" technique (I) is significantly more efficient than the normal DE strategy in terms of average number of filters used. The simple optimisation of only trying the "read and search" approach if the query contains a space (J) is quite effective. It is almost as effective as using a ten entry name-to-RDN mapping table (K). With this technique, the average number of operations to resolve an organisation name is 0.76 operations per query.

However, the simpler strategy of using a "read and search" if the query is the default value, and the normal DE strategy otherwise (L), is more efficient still. One can conclude that the cost of the read failures for the non-default queries outweighs the benefit of any read successes. We can further optimise this strategy by using a mapping table for the ten most popular queries (M), and using the "read and search" strategy for these queries as well as the default value. This strategy uses only 44% of the operations for resolving organisation names required by the raw DE strategy.

It should be evident that there are other combinations of the techniques described that could be tried. However it is clear from the results above that even simple techniques are effective at reducing the load on the directory when resolving organisation names.

5.3.6 Reasons for Failing to Match an Organisation Name Query

682 queries (16.40%) failed to match any entries using any of the various filters on the organizationName attribute, while 821 queries (19.74%) failed to find the correct entry. The principal reasons for matching failure were:

**Organisation not in directory:** The user entered a name for an organisation that did not have a corresponding directory entry.

**Organisation entry not immediately under c=GB:** The organisation queried has an entry in the directory but it was not immediately beneath the c=GB entry, but held one or more levels further down the DIT. In the UK all organisation entries in this category are held under the "University of London" entry. In other countries the problem is mainly that organisation entries are held beneath state or locality entries.

**In the directory:** The user's input corresponded to an organisation in the directory, but was misspelled, or was in a form not held in the directory. The various problems are discussed
The relative frequency of these matching failures for the sample data is shown in Table 5.10.

<table>
<thead>
<tr>
<th>Reason</th>
<th>per cent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organisation not in directory</td>
<td>65.29</td>
</tr>
<tr>
<td>Org entry not under c=GB</td>
<td>5.97</td>
</tr>
<tr>
<td>Org in directory, but not found</td>
<td>28.75</td>
</tr>
</tbody>
</table>

Table 5.10: Reasons for organisation name matching failures

Almost two-thirds of the queries where the entry was not found failed simply because the entry was not in the directory; coverage of the UK directory is predominantly higher education establishments with a few computing and research organisations. Just over 5% of query failures were due to the target entry not being under the c=GB entry. See Chapter 2 for a discussion of this issue.

However, nearly 30% of the failures (and almost 6% of all queries) failed to match even though the entry was in the directory, and in the expected place. Misspelling is a significant problem for organisation names, although we will see in Section 5.3.7 that the approximate matching algorithm finds the correct entry in the majority of cases of misspelled input. The main cause of matching failures in my sample is that the form of the user's query did not match the form of the name or names in the directory. Essentially this is an approximate matching problem, but one not solved by the algorithm used in the Quipu system used for these tests. However, we can solve this problem in many cases by removing stop list words and punctuation from the user input if matching on the full input fails. We will examine a number of techniques in Section 5.3.8.

5.3.7 How Useful is Approximate Matching for Misspelled Organisation Names?

We saw in Chapter 3 that misspelling of organisation name input was a significant problem; 3.29% of all organisation name input in my sample was misspelled. Overall, there were 120 misspellings of organisation names where the organisation has an entry in the directory. The successfulness of approximate matching with misspelled organisation name input is shown in Table 5.11.

Approximate matching found the correct entry in exactly two-thirds the cases where the input was misspelled. If the correct entry was not found, usually no results were returned rather than incorrect ones.

In the next section we will look at whether there is anything else we can do to find entries that cannot be found even by using approximate matching.
5.3. ORGANISATION NAMES

<table>
<thead>
<tr>
<th>Outcome of approximate match</th>
<th>Freq. as %age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Found correct entry</td>
<td>66.67</td>
</tr>
<tr>
<td>Found incorrect entry</td>
<td>0.83</td>
</tr>
<tr>
<td>Failed to find anything</td>
<td>32.50</td>
</tr>
</tbody>
</table>

Table 5.11: Misspelled organisation names and approximate matching

5.3.8 Finding Organisation Entries Not Found By Normal Matching Techniques

In this section we review some steps that can be taken to help with the 236 queries that did not yield any results using `oExact`, `oLeadSub`, `oAnySub` or `oApprox` matching, even though the correct entry was in the directory.

The most common problem in the sample query data was that users had entered a form such as “university of foo” when the directory entry name was of the form “Foo University”, or vice-versa. A very simple first step is to remove a leading “university of ” or a trailing “ university” and to try substring or approximate matching on the name tokens that remain. 137 of the queries were in this format. The results of this experiment are shown in Table 5.12.

<table>
<thead>
<tr>
<th>Filter type</th>
<th>No of correct matches</th>
<th>No of correct single matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>oAnySub</td>
<td>115</td>
<td>106</td>
</tr>
<tr>
<td>oApprox</td>
<td>118</td>
<td>98</td>
</tr>
</tbody>
</table>

Table 5.12: Matching organisation names stripped of “University (of)”

The technique is clearly very successful. Approximate matching found 118 correct matches using this technique, exactly half the queries previously not matched. The `oAnySub` filter is more selective than the `oApprox` filter – a higher proportion of single correct results – although there were no incorrect matches using either filter.

Having dealt with the easy cases, the remaining 118 cases require a little more work. We noted in Chapter 3 that one or more of a small set of stop list words or punctuation occurred in 2.67% of all queries. We noted some cases showing how these words and characters could prevent matching. In the next three subsections I show how removing these stop list words can help matching. First, we look at some words and characters that should be removed from users’ input. Then we look in turn at matching techniques for single and multiple token input.
The Words and Characters Removed

Some modifications need to be made to all hard-to-match input, irrespective of whether single or multiple token. The steps are these:

- Remove any apostrophe characters.
- Replace any commas with spaces.
- Replace any hyphens with spaces.
- Remove the trailing ‘s’ character from any word ending in s, so long as the query is at least four characters.

The reasoning behind the first three steps should be self-evident; however, the fourth deserves some explanation. Even using Quipu's Soundex approximate matching algorithm, user input of “queens” does not match a directory name of “Queen’s”: see Appendix D for an explanation. A simple way of circumventing this problem is to remove the trailing ‘s’ character. Another reason is that users occasionally append an ‘s’ where none is required. An example in the data is input of “robert gordons” for “The Robert Gordon University”. A third reason is that abbreviations sometimes have an 's' appended, as in “notts” for “Nottinghamshire, or “herts” for “Hertfordshire”: the trailing ‘s’ prevents the abbreviated form being a substring of the long form. The reason for not making this change to short queries is that three letter names in queries are likely to be sets of initials.

We will assume in the next two subsections that the transformations described above have already been applied to the input data.

Finding single token entries

There are two useful approaches to the matching of single token queries. One approach works well for input of four or more characters, the other for input of three characters or less.

First we will examine the technique for single token queries of four or more characters: this is a sample of 29 queries. Experimentation showed that a useful technique is to reduce this type of query to four characters and then to try oAnySub or oApprox matching on the shortened string. Both matching techniques found the correct match in 17 cases out of the 29 in the sample.

The decision to truncate to four characters is somewhat arbitrary, but represents an attempt to strike a balance between finding as many correct entries as possible without finding too many spurious ones. The decision is made in the knowledge that over two-thirds of all organisation name spelling mistakes occur after the fourth character: see Chapter 3. We will review the optimum length for truncating strings in Section 5.3.9.

There were 23 other single token queries: those with input of fewer than four characters. All 23 queries were sets of initials. An obvious filter to handle this type of input is to use substring matching on the initial letters. For example, if the input is UEA, the filter would be:
5.3. ORGANISATION NAMES

This approach was successful with all 23 queries in the sample.

Finding multiple token entries

There were 66 multiple token queries that were not matched using the various organizationName filters or by stripping "university (of)" from the input.

Several further modifications should be made to multiple token input before further matching is tried. These are:

- Remove the following words: 'the', 'of', 'and'.
- Remove the character '&, replacing it with a space if necessary as a token separator.
- Remove the following words at the end of a query: plc, ltd, limited.
- In an academic environment, remove the following words: polytechnic, poly, college, school, institute and any word beginning with the letters "uni".
- Change "st." to "st".

The rationale for most of the above modifications to user input should be clear to readers. The words that are removed contribute little to the semantics of matching, and can cause matches to fail if present when not required, or if the words are in the wrong order.

Some of the modifications are clearly dependent on the academic querying environment which I used for my experiments. It might not, for example, be prudent to remove all words beginning with "uni" in a predominantly commercial environment. However, it seems reasonable to speculate that there may be some similar sorts of tuning that would work in other querying environments. We should also note these modifications are language dependent.

I tried five matching strategies on the 66 queries: the experiments are described in detail in Appendix C.1. It is impossible to identify a single best technique, as some matching strategies offer better precision while others offer good recall. However, two of the best techniques both rely on oAnySub matching on the first two words of the input that remain after the stop list words have been removed. The words are also truncated to a maximum of four characters. These two filters are:

Substring matching with ANDed components

\[(o=*truncatedFirstWord*) \text{AND} (o=*truncatedSecondWord*)\]

Substring matching with ORed components

\[(o=*truncatedFirstWord*) \text{OR} (o=*truncatedSecondWord*)\]

The AND filter matched 44 of the 66 queries, with no incorrect matches, and with good precision: it had an average result set of only 1.15 entries. The OR filter found a correct match
in 62 cases, found an incorrect match in the other four cases, but was much less precise in that the average result set was 6.38 entries.

The AND filter looks to be a valuable technique in that it found two-thirds of these hard to match queries. The OR filter is too profligate to be used other than as a technique of last resort, but in these circumstances it should be effective if large result sets do not deter the directory user.

5.3.9 Use of Approximate and/or Substring Matching For Hard-To-Find Organisation Queries

We saw in the previous section how a combination of pruning stop list words from input, truncating words to four letters and using substring matching techniques was effective at finding the majority of entries not found by approximate matching. In this section we evaluate whether we should consider replacing all uses of approximate matching by the prune/truncate/substring methods, or whether we should use these methods in tandem with approximate matching.

These ideas are evaluated on the set of queries that have corresponding entries in the directory, but that are not found by the following DE strategy. This is the version of DE described as case F in Section 5.3.4, except that it does not use approximate matching. The strategy is:

```plaintext
if (single token)
  if (input length <= 3)
    oExact, assocDomainLeadSubDot, oLeadSub
  else /* longer single token input */
    oExact, assocDomainLeadSubDot, oLeadSub, oAnySub
else /* multiple token */
  oExact, oLeadSub, oAnySub
```

This produced a set of 434 unmatched queries, 12.15% of all queries for which there is an entry in the directory.

I evaluated fifteen strategies which use various combinations of the techniques described in earlier sections. In particular, the strategies variously include one or more of the following techniques:

- Separating the input into words and matching the words individually;
- Only considering the first two words in the input;
- Stripping the input of stop list words;
- Truncating the tokens to various shorter lengths;
- Treating short input as sets of initials;
- Using approximate matching on pre-processed input;
5.3. ORGANISATION NAMES

- Using substring matching on pre-processed input;
- Using approximate matching and substring matching in tandem.

A detailed comparison of the fifteen techniques is given in Appendix C.2. The main findings are as follows.

Approximate matching (as implemented in Quipu) on this set of hard-to-match queries produces the correct result less than half the time. The effect on the overall sample of queries is to leave 6.6% of queries (which had directory entries) unmatched. In contrast, the best strategy I devised failed to find the correct match for less than 1% of queries and, furthermore, produced smaller result sets. Ten of the fifteen strategies matched at least 98% of all queries.

The best approximate matching and substring matching techniques produce very similar results. My view is that the performance of the best substring matching algorithm is the more interesting as substring matching is precisely defined, and thus a DU A can get guaranteed behaviour, whereas approximate matching is implementation dependent, and the matching characteristics are determined by the DSA answering the query.

The most important features of an algorithm trying to find hard-to-match queries are:

- The input should be stripped of stop list words and characters.
- Multiple word input should be reduced to the two leading words.
- Substring matching should be on the first four characters of input words; truncating to four characters gives the best balance between precision and recall.
- Input of three or fewer characters should be matched as a set of initials.
- The very best matching performance was obtained by using either approximate matching followed by substring matching, or substring matching followed by approximate matching.

5.3.10 How Much do These Enhancements Improve DE’s Organisation Name Matching?

We have discussed a variety of techniques for improving matching. We can now evaluate how much these techniques improve the original DE algorithm when applied to the full query set for which there are entries in the directory. The results in Table 5.13 provide a comparison between the original DE algorithm, and a revised DE algorithm that includes some of the most effective enhancements. The revised algorithm is that described as case O in Appendix C.2, which is based on algorithm F in Section 5.3.4.

I believe that the results show that the revised algorithm is markedly superior. For the query sample as a whole, the improved algorithm correctly matches almost 6% more queries, with only 1.3% of all queries now not being correctly matched. This improvement is highly significant (using
CHAPTER 5. EXPERIMENTS WITH NAME MATCHING

<table>
<thead>
<tr>
<th>Filter code</th>
<th>% matched by filter</th>
<th>Mean no. res.</th>
<th>Med. no. res.</th>
<th>Max. no. res.</th>
<th>% queries with correct res.</th>
<th>% queries with single res.</th>
<th>% results correct</th>
<th>Mean no. of searches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orig DE</td>
<td>94.07</td>
<td>1.35</td>
<td>1</td>
<td>99</td>
<td>92.98</td>
<td>87.18</td>
<td>98.84</td>
<td>1.64</td>
</tr>
<tr>
<td>New DE</td>
<td>99.13</td>
<td>1.18</td>
<td>1</td>
<td>16</td>
<td>98.71</td>
<td>92.58</td>
<td>99.58</td>
<td>1.76</td>
</tr>
</tbody>
</table>

Table 5.13: A comparison of the original DE algorithm with an algorithm including the most effective enhancements

McNemar's test for comparing paired binary outcomes for large samples) with a \( z \) value of 14.11: this corresponds to a \( P \) value of less than 0.001.

The revised algorithm is more precise with a lower mean number of results, and more single entry result sets. The cost of these improvements is that the revised algorithm now requires an average of 1.76 search operations compared with 1.64 for the original strategy. In the author's view this cost is more than recompensed by the enhanced matching capabilities.

5.3.11 Summary on Organisation Name Matching

There is a wealth of detail in this section. The following points stand out.

- The usefulness of alternative names is demonstrated by the fact that 27% of the queries matched entry RDNs, while 57% exactly matched an organisation name.

- Input of two or three characters should be treated separately: it should not be matched using \( \text{oAnySub} \) or \( \text{oApprox} \) matching; short input should be treated either as a set of organisation name initials or as an abbreviated organisation name.

- The DE strategies were better than the UFN techniques at getting single entry result sets: DE got 35% more single entry result sets. This accuracy is gained at the cost of the DE technique on average using between 1.46 and 1.74 searches per query.

- The average number of search operations can be reduced to 0.6 per query by using the 'read and search' technique coupled with a mapping table.

- We can do a lot better than merely using approximate matching for finding hard-to-match entries. The most important steps are to remove stop-list words from the user input, and to use approximate or substring matching on the remaining input. It helps to truncate the tokens to four characters before trying substring matching.

5.3.12 Conclusions on Organisation Name Matching

While we have examined the DE and UFN algorithms in some detail, and have evaluated a number of possible improvements, it is not possible to select any individual technique as "the
best". There are trade-offs between all the various options and a DUA implementor must decide which matching characteristics he/she requires. The UFN algorithms are very simple and only require a single search operation. The DE strategy of a sequence of searches typically requires more querying operations, but is better at finding the correct entry with no other spurious entries. The cost of the DE search strategy can be considerably reduced by the use of a mapping table, which obviates the need to evaluate the organisation name for the most frequent queries.

If either a DE or UFN strategy is applied to the input as it is, over 6% of correct entries were not found. This occurred because of misspellings, and mismatches between the form of a user's query and the name in the directory. The most important step in finding these hard to match entries is to strip the input of stop list words and punctuation. Having done this, approximate and substring matching techniques can be used to match the majority of these difficult queries, particularly if only the first two words of the stripped input are considered. If substring matching is used, the input words should be truncated to four characters.

I have demonstrated some modifications to the DE strategy to work better with misspelled or mal-formed queries. The enhancements are effective with little impact on the average number of operations required to resolve a query.

5.4 Organisational Unit Names

The analysis of organisational unit name matching is based on experiments using the input from the UCL query source described in Chapter 3. All the queries analysed in these matching experiments are for departments within UCL, and are matched against a copy of the UCL department data held in the DIT; this is described in Chapter 4.

The query data consists of all the UCL query data department name input, that was recognisably a department name, excluding the following categories:

- Queries with asterisks; the user has indicated the type of substring match they require.
- Queries with superfluous punctuation characters, such as backslashes and quotation marks. These characters could be detected and removed from user input; their presence is likely to inhibit matching. Note that input with dots after initials is NOT excluded.
- Queries where I was unable to determine which department name was intended.

The resulting sample of 784 queries forms the basis for the experiments described in this section.

5.4.1 The Effectiveness of some Basic Department Matching Filters

I compared four basic filters to test their effectiveness at finding the required entries. I have also included an analysis of how often user input exactly matched the corresponding organisational unit RDN. This case and the four filters are as follows:
CHAPTER 5. EXPERIMENTS WITH NAME MATCHING

Organisational Unit Name input is RDN (ouRDN): The user input exactly matches the entry’s RDN.

Organisational Unit Name - exact match (ouExact): This matches input that is exactly equivalent to one of the organizationalUnitName values.

Organisational Unit Name - leading substring (ouLeadSub): This filter matches queries where the input is abbreviated: e.g. this filter finds “Physics and Astronomy” but not “Meta-physics” if a user’s query is ‘Physics”; it finds “Computer Science” if a user types “Comp”.

Organisational Unit Name - any substring (ouAnySub): This filter is useful for matching input that corresponds to part of a longer, formal name: e.g. it finds “Physics and Astronomy” if a user’s query is “Astronomy”.

Organisational Unit Name - approximate match (ouApprox): This filter is useful for matching misspelled input, and other cases where a user’s input does not precisely match the form in the directory.

<table>
<thead>
<tr>
<th>Filter type</th>
<th>% matched by filter</th>
<th>Mean no. res.</th>
<th>Med. no. res.</th>
<th>Max. no. res.</th>
<th>% queries with single res.</th>
<th>% queries with correct res.</th>
<th>% results correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>ouRDN</td>
<td>50.26</td>
<td>0.50</td>
<td>1</td>
<td>1</td>
<td>50.26</td>
<td>50.26</td>
<td>100.00</td>
</tr>
<tr>
<td>ouExact</td>
<td>51.66</td>
<td>0.52</td>
<td>1</td>
<td>1</td>
<td>51.66</td>
<td>51.66</td>
<td>100.00</td>
</tr>
<tr>
<td>ouLeadSub</td>
<td>75.77</td>
<td>0.86</td>
<td>1</td>
<td>3</td>
<td>66.58</td>
<td>74.49</td>
<td>98.32</td>
</tr>
<tr>
<td>ouAnySub</td>
<td>80.87</td>
<td>1.45</td>
<td>1</td>
<td>12</td>
<td>58.04</td>
<td>79.72</td>
<td>98.58</td>
</tr>
<tr>
<td>ouApprox</td>
<td>89.67</td>
<td>2.58</td>
<td>1</td>
<td>28</td>
<td>58.55</td>
<td>88.39</td>
<td>98.58</td>
</tr>
</tbody>
</table>

Table 5.14: Breakdown of matching department names by filter

The analysis of the matching performance of these filters is shown in Table 5.14. It shows that just over a half of the user input in this sample was a department RDN. A further one and a half per cent of the sample were matched by an ouExact filter; very few of the departments have alternative names. Each of the remaining three filters finds progressively more matches, with approximate matching finding at least one matching entry in almost 90% of all queries. We examine in Section 5.4.4 the reasons why the remaining 10% of queries did not match any entries (in fact over 11% did not match correctly), and whether we can do anything about it.

In contrast with organisation entries, where one would anticipate there to be thousands of organisation entries beneath the country node in a fully populated DIT, the number of organisational units within an organisation is likely to be much smaller. The UCL database has 137 departments under the organisation node. This means that the number of results returned is usually low, irrespective of the filter. However, as with organisation name matching, substring
and approximate match filters can both return large result sets given certain input; the characteristics of input that is likely to produce large result sets is discussed in Section 5.4.2. As with matching organisation names, approximate matching finds more correct entries, but with a lower proportion of single result sets.

All the filter types usually return the correct result; unlike the organisation filters, approximate matching was slightly better in this respect than the substring filters.

With these basic matching characteristics in mind, we can go on to look at some more sophisticated filters and search strategies. Before we do this, we will examine one possible refinement to matching strategies, and will look at the case for treating short input differently; this is suggested in the description of the UFN algorithm in RFC 1781.

### 5.4.2 Very Short Input

As for organisation names, we will examine if there is case for short department names being treated specially. Specifically is there a case for avoiding substring and approximate matching?

Whereas for organisation names I regarded short input as being three characters or fewer, for department names I have looked at queries of up to four characters. Whereas organisation acronyms are almost all three letters, this seemed to be less true for UCL’s department names, which were anything up to five characters in length.

<table>
<thead>
<tr>
<th>Type of Input</th>
<th>Fewer than or equal to 3 chars</th>
<th>Fewer than or equal to 4 chars</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initials</td>
<td>36</td>
<td>41</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>23</td>
<td>35</td>
</tr>
<tr>
<td>Name</td>
<td>0</td>
<td>18</td>
</tr>
<tr>
<td>Total</td>
<td>59</td>
<td>94</td>
</tr>
</tbody>
</table>

Table 5.15: Type of department name input for short queries

All the input of fewer than or equal to three characters was a set of initials or an abbreviated department name. Input of up to four characters includes some queries for a department with a short name, namely “Laws”.

The full department name and the abbreviated names can be matched by ouExact and ouLeadSub matching. The initials need to be treated specially, as this type of input is not matched straightforwardly by any of the basic filters described in Section 5.4.1, unless the initials are stored in the directory as one of the values of the department name. There are two possible filters which could be used in addition to the ouExact and ouLeadSub filters suggested above. These are:

\[
\text{ou=+firstLetter* secondLetter* [ thirdLetter* fourthLetter*]}
\]
CHAPTER 5. EXPERIMENTS WITH NAME MATCHING

\[ \text{ou}^* = \text{firstLetter secondLetter} \ [ \text{thirdLetter fourthLetter} ] \]

where the third and fourth letter components are included as appropriate. In the experiments that are described shortly, I have used the second of these techniques because the substring matching method was affected by the software bugs in Quipu described in Section 5.1.2. This technique is referred to as \textit{ouApproxOnInitials}.

There was only one query in my sample that suggested a case for using \textit{ouAnySub}, and none for \textit{ouApprox} matching, on this type of input.

The figures in Table 5.16 show how many queries of four or fewer characters, that failed to find any entries using \textit{ouExact} matching, found entries using \textit{ouLeadSub} or \textit{ouApproxOnInitials} matching. The figures are based on the DE approach of sequential filters, where the \textit{ouApproxOnInitials} filter is only tried if the \textit{ouLeadSub} filter fails to find any results.

<table>
<thead>
<tr>
<th>Filter type</th>
<th>Correct results</th>
<th>Incorrect results</th>
<th>No match found</th>
</tr>
</thead>
<tbody>
<tr>
<td>ouLeadSub</td>
<td>36</td>
<td>2</td>
<td>30</td>
</tr>
<tr>
<td>ouApproxOnInitials</td>
<td>27</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 5.16: Short department name queries not matched using exact matching

This sequence of filters found matching entries for all but one query, and almost always found the correct entry. The maximum result set was only three entries whereas using approximate matching on the same queries produced bigger result sets including one of 28 entries.

5.4.3 Search strategies based on more sophisticated filters

We now examine seven further search strategies to explore some possible improvements. Some of the examples are included to illustrate problems that have to be solved. One of the strategies is that currently implemented in DE. Another is that described in RFC 1781, which describes the UFN search algorithm. Having learned from this analysis, we include some filter combinations that are demonstrably better than those specified or currently implemented. There are also three strategies that use the "read then search" technique; there are examples both with and without name-to-RDN mapping tables.

The seven strategies are as follows:

A: \textit{UFN}. This is the filter suggested in RFC 1781, which describes the UFN matching algorithm.

This filter is as follows:

\[ \text{ouAnySub OR ouApprox} \]
5.4. ORGANISATIONAL UNIT NAMES

B: UFN - length 4. This is a modified form of the UFN filter, which takes into account that short input tends to produce a lot of spurious matches using ouAnySub and ouApprox filters, and that short input is either a set of initials or an abbreviated name. The pseudo code for this modified UFN filter is as follows:

\[
\text{if (single token AND (input length <= 4))} \\
\text{ouLeadSub OR ouApproxOnInits} \\
\text{else} \\
\text{ouAnySub OR ouApprox}
\]

C: DE. This is the search strategy as implemented in DE. It uses a sequence of searches which continues until a result is found or the filters have all been tried. The pseudo code for the DE search strategy is as follows:

\[
\text{ouExact, ouLeadSub, ouAnySub, ouApprox}
\]

D: DE - length 4. This is the original DE strategy for input of more than four characters. Shorter input does not use ouAnySub or ouApprox filters, but treats short input as a set of initials. The pseudo code for this search strategy is as follows:

\[
\text{if (single token AND (input length <= 4))} \\
\text{ouExact, ouLeadSub, ouApproxOnInits} \\
\text{else} \\
\text{ouExact, ouLeadSub, ouAnySub, ouApprox}
\]

E: DE + read. This is the basic DE strategy, except that the department name is initially assumed to be the RDN, and thus does not need resolving by a search operation.

F: DE + read + mapping table. The same as strategy F, except that the ten most common non-RDN queries are mapped to their RDNs before trying the read first strategy.

G: DE + read (only if in mapping table). The mapping table contains the ten most common queries, with mappings to their corresponding RDNs if appropriate. The read first strategy is only tried if the query is in the mapping table.

The analysis of these filters is shown in Table 5.17.

As with the organisation name matching, there are more similarities between the different techniques than differences. They all find matches for between 89.67% and 91.96% of the queries; they all have a median of a single match; when they deliver results, they are correct between 97.30% and 98.89% of the time. What are the main differences?

First, the DE strategy of a sequence of filters (cases C to G) matches fewer entries on average (1.10 for the best DE strategy and 1.81 for the best UFN strategy). The DE strategy also gets a single result significantly more often; the best UFN strategy (case B) for 64.92% of all queries, compared with 86.48% for the best DE strategy (case D).
CHAPTER 5. EXPERIMENTS WITH NAME MATCHING

<table>
<thead>
<tr>
<th>Filter code</th>
<th>% matched by filter</th>
<th>Mean no. res.</th>
<th>Med. no. res.</th>
<th>Max. no. res.</th>
<th>% queries with single res.</th>
<th>% queries with correct res.</th>
<th>% results correct</th>
<th>Mean no. of filters</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>89.67</td>
<td>2.58</td>
<td>1</td>
<td>28</td>
<td>58.55</td>
<td>88.39</td>
<td>98.58</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>91.96</td>
<td>1.81</td>
<td>1</td>
<td>11</td>
<td>64.92</td>
<td>90.94</td>
<td>98.89</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>89.67</td>
<td>1.28</td>
<td>1</td>
<td>28</td>
<td>83.04</td>
<td>87.24</td>
<td>97.30</td>
<td>1.92</td>
</tr>
<tr>
<td>D</td>
<td>91.96</td>
<td>1.10</td>
<td>1</td>
<td>9</td>
<td>86.48</td>
<td>89.80</td>
<td>97.64</td>
<td>1.88</td>
</tr>
<tr>
<td>E</td>
<td>89.67</td>
<td>1.28</td>
<td>1</td>
<td>28</td>
<td>83.04</td>
<td>87.24</td>
<td>97.30</td>
<td>1.91</td>
</tr>
<tr>
<td>F</td>
<td>91.96</td>
<td>1.29</td>
<td>1</td>
<td>28</td>
<td>83.80</td>
<td>90.56</td>
<td>98.47</td>
<td>1.27</td>
</tr>
<tr>
<td>G</td>
<td>89.67</td>
<td>1.28</td>
<td>1</td>
<td>28</td>
<td>83.04</td>
<td>87.24</td>
<td>97.30</td>
<td>1.42</td>
</tr>
</tbody>
</table>

Table 5.17: Breakdown of department name matching by filter

The two strategies that treat short input specially (cases B and D) have much lower maximum result set sizes than the other techniques, and get single results more often than other techniques.

A slightly higher proportion of UFN queries returned the correct match; the DE strategy of sequential filters sometimes gets a wrong match, and thus stops searching before the correct match is found.

There was virtually no benefit from using the “read then search” technique on the raw query data. Doing this with the existing DE strategy reduced the average number of directory operations from 1.92 to 1.91. However, both mapping table approaches used with the “read then search” technique were effective at reducing the load on the directory by between a quarter and a third the number of operations.

5.4.4 Reasons for Failing to Find an Entry

81 queries (10.33%) failed to match any entries using any of the following filters: ouExact, ouLeadSub, ouAnySub or ouApprox. 91 queries (11.61%) failed to find the correct entry with these filters. The principal reasons for matching failure were:

**Initials:** The user entered a name comprising a set of initials, and this form was not in the directory. We have examined filters to solve this problem in Section 5.4.2.

**Abbreviation:** The user entered an abbreviated form of a name, and this did not match the full name. An example is “Maths” not matching Mathematics.

**Punctuation:** The user input included or omitted punctuation characters which prevented matching. These were all either wrongly included ampersands or wrongly omitted apostrophe ‘s’ characters.

**Department:** The user’s input included the word “department (of)”.
5.4. ORGANISATIONAL UNIT NAMES

Wrong form: The user entered a name that was of the wrong form in some other way. Examples include "General Administration" for "General Administrative Services Division", and "Astronomy and Physics" for "Physics and Astronomy".

Spelling mistake: The user's input was misspelled.

The relative frequency of these matching failures for the sample data is shown in Table 5.18.

<table>
<thead>
<tr>
<th>Reason</th>
<th>per cent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initials</td>
<td>30.8</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>8.8</td>
</tr>
<tr>
<td>Punctuation</td>
<td>18.7</td>
</tr>
<tr>
<td>Department</td>
<td>5.5</td>
</tr>
<tr>
<td>Wrong form</td>
<td>27.5</td>
</tr>
<tr>
<td>Spelling mistake</td>
<td>8.8</td>
</tr>
</tbody>
</table>

Table 5.18: Reasons for failing to find the correct department entry

Whereas for organisation names where the majority of the queries not matched successfully did not have directory entries, almost all department queries have corresponding directory entries. There were only 11 for which I could not determine the intended department with reasonable certainty, and these have been excluded from the sample.

Nearly a third of queries where the correct entry is not found are where the user has entered initials and this form is not in the directory. Almost as many queries suffer from errors of form; often in such cases, a match would have been made if the user had typed less. Punctuation mismatches account for nearly one in five failures to find an entry, and misspelling nearly one failure in ten.

5.4.5 How Useful is Approximate Matching for Misspelled Department Names?

We saw in Chapter 3 that misspelling of department name input was a significant problem; 3.19% of UCL department name input in my sample was misspelled. Overall, there were 25 misspellings of department names where the department has an entry in the directory. The successfulness of approximate matching with misspelled department name input is shown in Table 5.19.

Approximate matching found the correct entry in just over two-thirds the cases where the input was misspelled. If the correct entry was not found, no results were returned rather than incorrect ones. These results are almost identical to those for misspelled organisation names.

In the next section we will examine what we can do to find entries that cannot be found even by using approximate matching.
### Table 5.19: Misspelled department names and approximate matching

<table>
<thead>
<tr>
<th>Outcome of approximate match</th>
<th>Freq. as %age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Found correct entry</td>
<td>68.00</td>
</tr>
<tr>
<td>Found incorrect entry</td>
<td>0.00</td>
</tr>
<tr>
<td>Failed to find anything</td>
<td>32.00</td>
</tr>
</tbody>
</table>

#### 5.4.6 Finding Department Entries Not Found By Basic Matching Techniques

Since many of the problems that inhibited organisation name matching also occur with department queries, we should anticipate that similar techniques will help us to find the entries not found with the exact, substring and approximate match filters on the full input. This was true for 81 (10.33%) of my department name queries. In this section we will test the techniques of removing stop list words and punctuation, truncating words to a few characters and using substring matching, and treating short input as initials.

We noted in Chapter 3 that one or more of a small set of stop list words or punctuation occurred in 6.75% of all queries, and saw in Section 5.4.4 that almost a quarter of the queries that failed to match failed because of punctuation characters or because they wrongly included the word “department”. Some modifications were made to all hard-to-match input, irrespective of whether single or multiple token. The steps were these:

- Remove any apostrophe characters.
- Replace any commas with spaces.
- Replace any hyphens with spaces.
- Remove the trailing ‘s’ character from any word ending in s, so long as the query is at least four characters.

In fact, the first three of these steps were not necessary with my problem queries. A number of trailing ‘s’es were removed: usefully, for example, from Bursars, Provosts and Registrars, as in each of these cases the directory name has an apostrophe ‘s’; needlessly, but harmlessly for matching, from words such as “Physics” and “Genetics”.

Several further modifications were made to multiple token input before matching was tried. These were:

- Remove the following words: the, of, and.
- Remove the character ‘&’, replacing it with a space if necessary as a token separator.
5.4. ORGANISATIONAL UNIT NAMES

- Remove the following words: department, office, school.
- Remove any tokens in parentheses.

The rationale for most of the above modifications to user input follows from the analysis in Chapter 3, which showed that these words and characters often occurred in user input, but did little to assist matching. The last of the above cases follows from an examination of the queries that failed. Input tokens in parentheses were usually subtle qualifiers of the department name, and tended to impede matching. The list of problem words could obviously be extended or modified to suit the needs of a particular organisation.

An indication of the effectiveness of the steps described above are that simply by using approximate matching on the modified input, 33 of the 81 missing matches were found.

We saw in Section 5.4.2 that input of four characters or fewer needs to be treated specially, and we will use the approximate match filter suggested in that section. That is:

\[ \text{ou}^* = \text{firstLetter secondLetter [thirdLetter fourthLetter]} \]

Of the 19 cases (only two names) of input of fewer than five characters, 18 cases (one name) matched using the approximate matching technique described above.

We noted in Section 5.3.8 that truncating input to four characters was a possible alternative to approximate matching for finding organisation entries: we test the effectiveness of the technique with department queries. Single token input is handled by an ou\text{AnySub} filter. There were 27 hard-to-match single token queries.

Two multiple input token techniques are tried. Both use ou\text{AnySub} filter items for the first two words; one technique ANDs these filter items while the other technique ORs the two filter items. The two filters can thus be written:

\[(\text{ou}=*\text{truncatedFirstWord*}) \text{ AND } (\text{ou}=*\text{truncatedSecondWord*})\]

\[(\text{ou}=*\text{truncatedFirstWord*}) \text{ OR } (\text{ou}=*\text{truncatedSecondWord*})\]

There were 35 hard-to-match multiple token queries. The results for the various matching experiments are shown in Table 5.20.

Approximately two-thirds of the queries for the single token and the ANDed multi token input achieved matches, and these were correct in over 95% of these cases. The ORed multiple token filter found the correct entry in over three-quarters of all cases, but at a cost of a higher average number of matches and very few single entry result sets.

5.4.7 Use of Approximate and/or Substring Matching For Hard-To-Find Department Queries

As we did for organisation name matching, we will now review whether we can use the techniques of pruning stop list words from input, truncating words to four letters and using substring
CHAPTER 5. EXPERIMENTS WITH NAME MATCHING

<table>
<thead>
<tr>
<th>Input/filter type</th>
<th>% matched by filter</th>
<th>Mean no. res.</th>
<th>Med. no. res.</th>
<th>Max. no. res.</th>
<th>% queries with single res.</th>
<th>% queries with correct res.</th>
<th>% results correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single token</td>
<td>62.96</td>
<td>1.41</td>
<td>1</td>
<td>7</td>
<td>37.04</td>
<td>62.96</td>
<td>100.00</td>
</tr>
<tr>
<td>Multi token /</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- AND</td>
<td>68.57</td>
<td>0.89</td>
<td>1</td>
<td>2</td>
<td>48.57</td>
<td>65.71</td>
<td>95.83</td>
</tr>
<tr>
<td>- OR</td>
<td>85.71</td>
<td>5.60</td>
<td>4</td>
<td>38</td>
<td>7.14</td>
<td>76.19</td>
<td>88.89</td>
</tr>
</tbody>
</table>

Table 5.20: Matching hard-to-match department name queries using truncated input

matching instead of or as well as approximate matching.

These ideas are evaluated on the set of 150 queries which had corresponding entries in the directory, but that were not found by the following DE strategy. This is the normal DE strategy, except omitting the ouApprox filter.

ouExact, ouLeadSub, ouAnySub

This set of 150 queries constitutes 19.12% of all queries for which there is an entry in the directory.

I evaluated fourteen different search strategies which use various combinations of the techniques discussed earlier. These techniques include:

- Splitting the input into separate tokens and approximately matching the words individually.
- Only considering the first two words in the input.
- Stripping the input of stop list words.
- Truncating the tokens to various shorter lengths.
- Treating short input as initials.
- Using substring matching first, and then trying approximate matching if substring matching failed; and vice-versa.

A detailed comparison of the fourteen techniques is given in Appendix C.3. The main findings are as follows. Note that when a full sample matching rate is given, it assumes that exact and substring matching have been used to match the simpler queries.

Quipu’s approximate matching matched less than half (46%) of this set of hard-to-match queries. The effect on the whole sample of using this matching technique is to leave over 10% queries unmatched. In contrast, the most successful matching strategy, based on approximate matching of modified user input, found the correct entry for 98% of these hard-to-match queries: an overall matching rate for the all query data of 99.6%. The best substring matching techniques found almost 90% of the hard-to-match queries, a full sample success rate of 98%.
One very simple technique is worth mentioning. Simply reducing the input, however many words, to a string of the first four characters in the input (or the first word if that has fewer than four characters) produced full sample matching rates of 97% or 98% for \textit{ouLeadSub} and \textit{ouAnySub} matching respectively.

The majority of the fourteen techniques also offered good precision, with over half the queries producing single entry result sets. All the experimental techniques found less incorrect results than the benchmark technique of approximate matching on unmodified data.

The findings largely corroborate those for the hard-to-match organisation name queries.

- The input should be stripped of stop list words and characters.
- Multiple word input should be reduced to the two leading words.
- Substring matching should be on the first four characters of input words; truncating multiple word input to the leading four characters only provides very effective recall, although the technique lacks precision.
- Input of four or fewer characters should be matched as a set of initials.
- A good blend of recall and precision can be achieved by using either approximate matching followed by substring matching, or substring matching followed by approximate matching.

5.4.8 How Much do These Enhancements Improve DE's Department Name Matching?

In this section, we evaluate how much these techniques improve the original DE algorithm when applied to the full query set for which there are entries in the directory. The results in Table 5.21 provide a comparison between the original DE and two possible revisions of the algorithm.

Revision A: This is based on the original DE algorithm, except that \textit{ouApprox} matching is not tried, and the approximate matching filter is replaced by technique N described in Appendix C.3.

\begin{verbatim}
ouExact, ouLeadSub, ouAnySub,
if (single token AND (input length < 5))
    ouApproxOnInitials
else if (single token)
    ouApprox, ou=*truncatedFirstWord*
else
    (ou"=firstWord) AND (ou"=secondWord), ou=*truncatedFirstWord*
\end{verbatim}

Revision B: As for revision A except that \textit{ouLeadSub} is not tried.
CHAPTER 5. EXPERIMENTS WITH NAME MATCHING

<table>
<thead>
<tr>
<th>Filter code</th>
<th>% matched by filter</th>
<th>Mean no. res.</th>
<th>Med. no. res.</th>
<th>Max. no. res.</th>
<th>% queries with single res.</th>
<th>% queries with correct res.</th>
<th>% results correct</th>
<th>Mean no. of searches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orig DE</td>
<td>89.67</td>
<td>1.28</td>
<td>1</td>
<td>28</td>
<td>83.04</td>
<td>87.24</td>
<td>97.30</td>
<td>1.92</td>
</tr>
<tr>
<td>Revision A</td>
<td>98.98</td>
<td>1.33</td>
<td>1</td>
<td>17</td>
<td>90.43</td>
<td>97.32</td>
<td>98.32</td>
<td>1.86</td>
</tr>
<tr>
<td>Revision B</td>
<td>98.98</td>
<td>1.90</td>
<td>1</td>
<td>17</td>
<td>71.17</td>
<td>97.45</td>
<td>98.45</td>
<td>1.22</td>
</tr>
</tbody>
</table>

Table 5.21: A comparison of the original DE department matching algorithm with two algorithms including effective enhancements

Both revised algorithms are superior to the original DE in that they find matches for most of the queries that were not previously matched. Whereas the original DE failed to match over 10% of all queries, the failure rate is only 1% for both revised algorithms, and these were mostly for cases where the user had entered a name form not held in the directory such as “accommodation office” not matching “student residences office”. The improvement in the ability of revised algorithm A over the original algorithm to find correct results is statistically highly significant: McNemar’s test gives a $z$ value of 8.78, with a corresponding $P$ value of less than 0.001.

The revised algorithms are also marginally more accurate than the original algorithm - fewer queries match the wrong entries.

The revised algorithms differ in their precision and number of filters required. Revision A would be preferred if the ability to uniquely entries is important, while revision B is better if it is important to minimise the number of search filters used. It may be possible to further reduce the number of filters required by the use of a mapping table.

5.4.9 Summary on Department Name Matching

- Department name matching is helped by there being a relatively small number of departments in a typical organisation.

- The DE sequence of searches strategy is better than the UFN single search for getting fewer results on average and more single entry result sets; the cost is an average of nearly two searches per query.

- Short input needs special treatment; it should either be regarded as an abbreviated department name or a set of initials.

- The read and search strategy resulted in no reduction of operations on raw query data.

- Read and search does offer an improvement in terms of fewer operations if used with a name-to-RDN mapping table.
5.5. MATCHING PEOPLE NAMES

- Removing stop list words, and only considering first two words in input are both useful techniques.

- Some techniques are presented that find over twice as many correct results as are found by approximate matching for those queries not found by exact or substring matching.

5.4.10 Conclusions on Department Name Matching

The conclusions on department name matching are similar to those on organisation name matching. If a simple algorithm is required, a modified version of the UFN algorithm is effective. If greater precision is required, particularly in the ability to get single entry result sets, then a revised version of the DE strategy is best.

However, there are some differences between department name and organisation name matching. There were fewer stop list words in the department name input: removing such words is less critical to successful matching than it was for organisation name matching. Department name matching is potentially a simpler problem than organisation name matching: the number of departments is constrained by organisation size, whereas the directory could conceivably have tens or hundreds of thousands of organisation entries. This suggests that a more profligate matching algorithm is feasible for department name matching, as there will in general be less entries to mis-match against.

Another big difference, if we recall the discussion of search scope in Section 2.6, is that usually we do not need to match department names anyway. We examine this issue further in Appendix 5.6.

5.5 Matching People Names

The analysis of person name matching is based on experiments using the UCL query data described in Chapter 3. The experiments were conducted against a set of 573 queries, which were a mixture of single token and multiple token queries. A further 108 queries were not used in the experiments as the directory did not appear to contain corresponding entries: the UCL database contains entries for most, but not all, staff and students. Another 36 queries were excluded from the original query set: these queries were omitted from the main analysis only because their inclusion would tend to obscure other findings. The queries omitted from the main analysis fall into the following categories:

- The query contained an asterisk; the user has thus indicated the type of substring matching required.

- The query contained extraneous punctuation characters, probably due to typing errors. These errors could be detected before matching is attempted.
The query was not for a person, but for a room or a role. These queries are supported by DE, but are beyond the scope of this analysis. They formed almost 2% of the total queries where a name was entered.

Although the person name input was reasonable, a department name was specified and I could not deduce the intended department.

The user input was unambiguously a userid, not related in any way to a person's name: the UCL database includes userids for only 3% of the entries.

The query was a comma-separated UFN.

Other queries, all short, where it was not clear if the user's input really was a person name: e.g. it may have been a department name, or perhaps a typing error.

The query data was modified for some queries where a department was specified. In such cases, if a user specified, for example, “Computer Centre” when the correct department was “Computer Science”, the query data was modified to contain the correct department name. The reason for this was to focus on person name matching, rather than dwell on department name matching problems that have been discussed in Section 5.4. We examine the problem of users specifying the wrong department name in Appendix 5.6.

The query set was matched against a copy of the UCL directory database. This comprises 16,174 entries for people: the name formats used in the UCL database are outlined in the next section.

A significant difference between the work described in this section and the matching analysis for other categories of input is that the database of person names is much bigger. The issue of large result sets affects person name matching algorithms whereas it has not been a major issue for the other data types. I have included an analysis of the effect of user-imposed result set size limits since a number of DUAs have tried to deal with large result sets by the use of size limits.

### 5.5.1 Person name forms used in UCL database

The UCL X.500 database is collated from several sources, and several name forms are widely used in entries in the database. Most of 16174 entries have a single commonName attribute. However, 564 entries (basically those in the Computer Science department) have two or more commonNames. It is important that the reader knows something about the name formats used as this is the context for the matching experiments described in the remainder of this chapter.

Table 5.22 shows the most popular formats: the percentages sum to more than 100% as some entries have more than one format. Over three quarters of the entries include a commonName attribute with the first forename. Over a third of the entries also include a second forename. Just over a quarter include formats with initials instead of forenames. Although the proportion of entries which include alternative commonNames is lower than the average for the directory as
5.5. MATCHING PEOPLE NAMES

Table 5.22: Name formats used for person names in UCL database

<table>
<thead>
<tr>
<th>Format as an example name</th>
<th>% age entries including this format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alan Smith</td>
<td>34.6</td>
</tr>
<tr>
<td>Alan Bruce Smith</td>
<td>30.5</td>
</tr>
<tr>
<td>A Smith</td>
<td>21.0</td>
</tr>
<tr>
<td>Alan B Smith</td>
<td>6.4</td>
</tr>
<tr>
<td>Alan Bruce Charles Smith</td>
<td>5.2</td>
</tr>
<tr>
<td>A B Smith</td>
<td>5.0</td>
</tr>
<tr>
<td>Alan B C Smith</td>
<td>0.8</td>
</tr>
<tr>
<td>A B C Smith</td>
<td>0.6</td>
</tr>
</tbody>
</table>

5.5.2 The Effectiveness of Some Basic Filters for Matching Person Names

In this section we examine the effectiveness of some basic filters. The aims are threefold. First, we examine how successful the filters are at finding results. Second, we look at how many results are returned for each type of search, anticipating, for example, that approximate matching returns more results than exact matching. Third, we examine what proportion of queries that return results return the correct entry.

One difference with the analysis of people compared with that of other categories is that there is no analysis of how often each search strategy yields single entry result sets. The reason for this is that the size of the database, coupled with the fact that many user queries are surname-only, means that result sets with multiple entries are to be expected. In fact, single entry result sets are less important at leaf entries than higher up the DIT, where a search resolving to a single organisation or department name means that query resolution can proceed automatically to the next level lower down the DIT without reference to a user. In Appendix C.5 we examine the impact of database size on some aspects of the expected number of results.

Another difference is that the queries are broken into categories according to whether:

- The queries are single or multiple token. Multiple token queries – for example, an initial or forename and a surname – contain more information and tend to get smaller result sets. Furthermore, single and multiple token queries are best served by different matching strategies. For example, it does not usually make sense to match on the surname attribute if the query is multiple token.
If a department is specified, the scope of the query is narrowed to a smaller portion of the organisational database; if none is specified all entries within the organisation are in scope and result sets will inevitably be larger.

These two factors together make four combinations, and each is analysed separately.

We compare four filters for both single token queries and multiple token queries: the choice of filters is strongly influenced by those used in DE. The filter sets are slightly different for the single and multiple token queries. For the single token queries, the four filters are as follows (each filter type is tagged with an abbreviated name in parentheses and these tag names are used frequently in subsequent text and tables):

Surname - exact match \((snExact)\): Most single tokens are surnames.

CommonName - any substring match \((cnAnySub)\): This filter finds both forenames and surnames.

Surname - approximate match \((snApprox)\): This filter is useful for matching misspelled surnames.

CommonName - approximate match \((cnApprox)\): This filter is useful for matching either misspelled forenames or surnames.

For multiple token queries, the filters are:

CommonName - exact match \((cnExact)\): The filter matches if the user input and one of the forms in the directory match exactly.

CommonName - any substring \((cnAnySub)\): This filter is useful for multiple forename queries, and those where the user has specified a second forename and surname.

CommonName - initial-star-lastname \((cnInitStarLastname)\): Since not all directory entries include forenames, stripping the first name component down to a single initial often helps matching. For example, queries of the form "paul barker" or "p f barker" are both transformed into a filter of the form \(cn=p*barker\). The last component is separated by either a dot or a space character from other components.

CommonName - approximate match \((cnApprox)\): This filter helps with misspelled forenames and surnames. It also gives some support for matching initials and forenames.

The Number of Person Name Queries Returning Results

The sample queries were replayed on the UCL database using each of the filters described above. Table 5.23 shows the percentage of searches using each filter that yielded any results. The results are best interpreted separately for single and multiple token queries. For single token queries, over 83% of all queries exactly matched a surname. A further 10% of queries matched
5.5. MATCHING PEOPLE NAMES

<table>
<thead>
<tr>
<th>Filter</th>
<th>Single token</th>
<th>Multiple token</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>no dept</td>
<td>dept</td>
</tr>
<tr>
<td>snExact</td>
<td>85.4</td>
<td>80.3</td>
</tr>
<tr>
<td>cnExact</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>cnAnySub</td>
<td>95.1</td>
<td>91.2</td>
</tr>
<tr>
<td>snApprox</td>
<td>98.9</td>
<td>95.2</td>
</tr>
<tr>
<td>cnInitStarLastname</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>cnApprox</td>
<td>98.9</td>
<td>98.6</td>
</tr>
<tr>
<td>Not matched by any filter</td>
<td>1.1</td>
<td>1.4</td>
</tr>
<tr>
<td>Total in category</td>
<td>185</td>
<td>147</td>
</tr>
</tbody>
</table>

Table 5.23: Percentage of person name queries by filter where at least one entry is matched

using the \textit{cnAnySub} filter; there is analysis of what these extra matches were in Section 5.5.9. \textit{snApprox} and \textit{cnApprox} matching were more successful still at finding some matching entries, with only just over 1\% of queries unmatched.

The picture is quite different for multiple token queries. Only 30\% of these queries matched any entries using a \textit{cnExact} filter, with only 3\% more queries matching the input as a substring of an entry's commonName. In contrast, over 78\% of queries matched using a \textit{cnApprox} filter and 83\% matched using a \textit{cnInitStarLastname} filter. 8.7\% of multiple name queries did not match any directory entries for any of the four filters tried; this is considerably worse than the failure rate for single token queries. As one would expect, the more that a user enters, the less likely he/she is to get any results.

However, it also evident that the two most successful methods of matching multiple token queries match different entries to a considerable extent; we can see this as the most successful matching method, \textit{cnInitStarLastname}, finds results for 83\% of queries, while only 8.7\% queries failed to produce results for any matching method. We will examine the reasons for this in Section 5.5.7.

Only 71 user queries (12.4\%) were equivalent to person entry RDNs, and only 44 queries where a department name was specified. There appears to be little case for a read-then-search strategy for person entries, as the read is likely to fail far more often than it succeeds, and thus be a cost on look-ups rather than an efficiency gain.

**Number of Person Entries Returned**

Table 5.24 shows the mean and median number of results for each search filter. Note that the queries are not restricted by user imposed or DSA administrative size limits. While with a large database and surname-only queries, we cannot always expect to identify a single entry, we would
like the result sets to be as small as possible to help users identify the entries they require.

<table>
<thead>
<tr>
<th>Filter</th>
<th>Single token</th>
<th>Multiple token</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>no dept</td>
<td>dept</td>
</tr>
<tr>
<td></td>
<td>mean</td>
<td>med</td>
</tr>
<tr>
<td>snExact</td>
<td>4.7</td>
<td>2</td>
</tr>
<tr>
<td>cnExact</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>cnAnySub</td>
<td>21.0</td>
<td>3</td>
</tr>
<tr>
<td>snApprox</td>
<td>81.6</td>
<td>17</td>
</tr>
<tr>
<td>cnInitStarLastname</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>cnApprox</td>
<td>211.5</td>
<td>34</td>
</tr>
</tbody>
</table>

Table 5.24: Average number of results returned per filter type

The figures in Table 5.24 show three key influences on the number of results returned. These are:

**Department specified** If no department name is specified, the scope of the search is the whole organisation, and on average more results are returned.

**Number of tokens** The extra information provided by users in multiple token queries focus queries more tightly than surname or forename only queries.

**Approximate matching** produces far more results than exact or substring matching.

The means are distorted by a few queries that return very large numbers of results. The medians are more indicative of the typical number of results returned per query. These show that only the combinations of **snApprox or cnApprox** matching, for single token queries, when no department is specified, are likely to return a lot of results.

Table 5.25 shows how often this combination is likely to produce a lot of matches. More than a quarter of **cnApprox** matches produced more than 100 matches and more than one in ten produced over 500 matches (about 3% of the database) for my sample queries. This suggests that approximate matching is a blunt instrument for querying large databases, and that we should use it as sparingly as possible for single token queries when no department name is specified.

If we accept the argument about using approximate matching sparingly, the next question to be answered is: how many matches are we likely to get if we only use approximate matching when other exact matching or substring matching filters have failed? Table 5.26 gives some answers.

The average number of responses is still quite high for single token, no department queries, but much lower than the number of results returned for all queries. For example, the average number of results for **cnApprox** matching for the ‘single token, no department specified’ cohort was 211.5 for the entire cohort but only 27.9 for those queries where exact and substring matching
5.5. MATCHING PEOPLE NAMES

<table>
<thead>
<tr>
<th>Search filter</th>
<th>% queries returning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100+ matches</td>
</tr>
<tr>
<td>snApprox</td>
<td>19.5</td>
</tr>
<tr>
<td>cnApprox</td>
<td>27.6</td>
</tr>
</tbody>
</table>

Table 5.25: Percentage of 'single token, no dept' queries producing a lot of matches

<table>
<thead>
<tr>
<th>Search filter</th>
<th>Single token - no dept</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mean</td>
</tr>
<tr>
<td>snApprox</td>
<td>9.0</td>
</tr>
<tr>
<td>cnApprox</td>
<td>27.9</td>
</tr>
</tbody>
</table>

Table 5.26: The number of matches achieved by approximate matching if it is used only after exact and substring matching have failed for single token, no department person name queries

The Accuracy of Person Name Search Filters

It was often a difficult task deciding which entry a query was intended to match: the database was quite large; users did not always specify department names correctly; the database did not always include forenames; and there were many misspelled queries. Nevertheless, the figures in Table 5.27 should give a reasonable guide to the accuracy of the various filters. If I have erred in my judgement, it is probably to slightly underestimate the number of correct matches.

<table>
<thead>
<tr>
<th>Filter</th>
<th>Single token</th>
<th>Multiple token</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>no dept</td>
<td>dept</td>
</tr>
<tr>
<td>snExact</td>
<td>100.0</td>
<td>100.0</td>
</tr>
<tr>
<td>cnExact</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>cnAnySub</td>
<td>99.4</td>
<td>100.0</td>
</tr>
<tr>
<td>snApprox</td>
<td>92.9</td>
<td>91.4</td>
</tr>
<tr>
<td>cnInitStarLastName</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>cnApprox</td>
<td>98.9</td>
<td>98.6</td>
</tr>
</tbody>
</table>

Table 5.27: Percentage of queries returning results that return the correct result

I assumed that any single token input matched by a snExact filter and any multiple token filter matched by a cnExact filter were always correct. While this seems an obvious assumption,
it is not necessarily correct. For example, was the query of “d knight” meant to match the “D Knight” in the “Medical Research Council” department, or was it really intended to match “David Knight” in the “Information Systems Division”? \textit{cnExact} matching only returns the “D Knight” entry.

Most of the types of matching were very accurate. The worst performer was \textit{snApprox} matching on single input tokens with an accuracy of 92%. The main cause of the inaccuracy was returning spurious results when the query was in fact a forename.

\section*{5.5.3 Special Characteristics of Person Name Input}

In this section, we examine some of the characteristics of user queries that have an impact on querying strategies. This will help us to devise a querying strategy that gets the right results as often as possible with as few spurious results as possible.

\subsection*{Single Token Person Name Input}

We saw in Chapter 3 that single token input was mostly surnames, with forenames and userids comprising the bulk of the remainder. This suggests the following filters are necessary to find entries.

- \textit{snExact} to match surname queries
- \textit{cnLeadSub} to match forename queries
- \textit{uidExact} to match userid queries

There are no infallible indicators to which type of query is which. While the best initial assumption is that a single token query is a surname, forenames are used quite often in queries where the directory service user is looking for someone within their own organisation. However, we should remember from Chapter 4 that not all person entries include forenames in their \textit{commonName} attributes. One helpful indicator of name format is that userids very often contain numeric characters: see Chapter 3.

As misspelling is a common problem with person names, some \textit{approximate} matching filters may also be needed.

Several questions arise concerning the use of the \textit{userid} attribute in search filters. First, on the evidence of Chapter 4 \textit{userid} attributes are included in at most one third of all entries. Indeed, system administrators are increasingly wary of making userids public knowledge. A person attempting to break into a computer system has a head start if he/she knows one or more valid userids. Thus, while searching on userids may provide useful results, the lack of use of \textit{userid} attributes means that their use in search filters can only be seen as “topping up” a more general strategy.
Second, despite the widespread use of other directory services that make use of searching on userids, not all directory administrators allow searching on the userid attribute, even if they are included in the directory.

Third, there may be legal impediments. French law prevents searches of public directories using certain identifiers: a French colleague reports that it is not clear whether computer userids fall into this category [Lan96].

Fourth, the case for approximate matching on userid is questionable, although this is specified in RFC 1781. Often userids occupy algorithmically determined name-spaces, and it makes little sense to use an approximate matching algorithm designed to correct phonetic or typographical misspellings for matching names such as "ucjtsjd". In general, approximate matching algorithms are unlikely to work well where a name space is tightly clustered according to some artificial naming convention.

Multiple token person name input

In Chapters 3 and 4, we saw the variety of multiple token personal names used both in user queries and in the directory. This poses a matching problem. Input of "alan smith" or "smith, a" does not match a directory entry with name "A Smith" using exact, substring or Quipu's approximate matching. There are many more permutations involving multiple forenames or initials, full stops, etc.

We noted in Chapter 4 that nearly 95% of directory entries have at least one commonName form which starts with the first letter of the first forename and ends with the surname. In Section 5.5.2, we saw the usefulness of the cnInitStarLastname filter, which is based on a very simple transformation of multiple token input. We will examine some alternative transformations in Section 5.5.8.

While multiple token input poses matching problems, it has the advantage that we can discount matching on userids, which rarely if ever include spaces: UNIX allows it, but I have never seen a userid including a space. For English names, multiple tokens are rarely surnames. However, a few do occur and it may be prudent to try to match multiple token input against the surname attribute if other types of matching have failed.

Length of Person Name User Input

We have already seen, when considering matching on other input categories, that short input may need special treatment, either to reduce the number of matches or to match the correct entries at all.

Unlike organisation and department names, there was no obvious evidence of short input being sets of initials. However, we saw in Section 5.5.2 that some filters produce large result sets; to what extent is this caused by using inappropriate filters on short input?

I examined the worst query cohort - the 185 single token, no department queries - to ascertain
CHAPTER 5. EXPERIMENTS WITH NAME MATCHING

the strength of this effect. The findings are shown in Table 5.28.

<table>
<thead>
<tr>
<th>Query Length</th>
<th>Sample Size</th>
<th>snExact Ave</th>
<th>snExact Med</th>
<th>cnAnySubstring Ave</th>
<th>cnAnySubstring Med</th>
<th>snApprox Ave</th>
<th>snApprox Med</th>
<th>cnApprox Ave</th>
<th>cnApprox Med</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>1.0</td>
<td>1</td>
<td>1327.0</td>
<td>1327</td>
<td>750.0</td>
<td>750</td>
<td>2359.0</td>
<td>2359</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>4.9</td>
<td>2</td>
<td>77.3</td>
<td>20</td>
<td>619.1</td>
<td>239</td>
<td>1704.0</td>
<td>1244</td>
</tr>
<tr>
<td>4</td>
<td>23</td>
<td>2.6</td>
<td>3</td>
<td>16.3</td>
<td>5</td>
<td>159.9</td>
<td>128</td>
<td>467.9</td>
<td>293</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>2.4</td>
<td>1</td>
<td>26.0</td>
<td>4</td>
<td>115.7</td>
<td>57</td>
<td>292.0</td>
<td>109</td>
</tr>
<tr>
<td>6</td>
<td>44</td>
<td>5.3</td>
<td>2</td>
<td>6.5</td>
<td>3</td>
<td>35.6</td>
<td>10</td>
<td>68.7</td>
<td>33</td>
</tr>
<tr>
<td>7</td>
<td>35</td>
<td>4.3</td>
<td>1</td>
<td>4.8</td>
<td>2</td>
<td>14.7</td>
<td>11</td>
<td>22.9</td>
<td>13</td>
</tr>
<tr>
<td>8</td>
<td>24</td>
<td>10.7</td>
<td>2</td>
<td>12.2</td>
<td>2</td>
<td>18.0</td>
<td>5</td>
<td>32.7</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>13</td>
<td>4.5</td>
<td>1</td>
<td>4.6</td>
<td>1</td>
<td>8.0</td>
<td>4</td>
<td>8.9</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>2.5</td>
<td>0</td>
<td>3.0</td>
<td>0</td>
<td>3.0</td>
<td>0</td>
<td>3.5</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>1.5</td>
<td>1</td>
<td>1.5</td>
<td>1</td>
<td>1.5</td>
<td>1</td>
<td>1.5</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>1.0</td>
<td>1</td>
<td>1.0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 5.28: Person Name query length and the number of matches

Short query strings do not appear to be a problem for exact match filters. However, the one query of two characters - the string "de" - produced large numbers of matches when using the other three filters. It seems reasonable to avoid searches other than exact matches for queries of two characters only. Both snApprox and cnApprox matching returned large result sets (averaging over 100 entries) for all queries of five characters or fewer. cnApprox matching averaged over 20 matches for up to eight character queries. Since the principal use of cnApprox matching on single token names is to do fuzzy matching on forenames (since we can snApprox match to do fuzzy matching on surnames), the DUA designer should consider carefully whether this facility is required given the large number of results it tends to generate.

I also examined the 21 cnApprox match queries which returned over 500 results, all single token with no department name specified, to see if their large result sets could be predicted from the number of characters in the input. These queries were shorter than average: they had an average length of 4.10 characters compared to 6.29 characters for the single token, no department cohort as a whole. Furthermore, there were 31 queries with four or fewer characters in my sample, and 14 of these (45%) returned 500+ results for a cnApprox match.

There were 50 queries in the sample with one or more misspellings. Only one of these was of four characters or fewer, and only two more of five characters.

Since approximate matching is of little benefit for short queries, and also returns large result sets, there is little case for using approximate matching on short queries, except possibly as a last
5.5. MATCHING PEOPLE NAMES

Finally, the DUA designer has to remember that some queries on short input may be disallowed because of access control restrictions. The searchACLs defined in [HKH91] are used by some organisations to prevent short queries which might be used to retrieve large portions of the database.

5.5.4 More Sophisticated Search Strategies for Finding Person Entries

In this section, we examine four search strategies using multiple filter items. Two are based on the UFN filter, and two on the DE search strategy. The four strategies are as follows:

A: UFN. This is the filter suggested in RFC 1781. The filter is:

\[ \text{cnAnySub OR cnApprox OR snAnySub OR snApprox OR uidAnySub OR uidApprox} \]

B: Modified UFN. This is based on the original UFN filter, but attempts to increase the number of queries matched by using the \text{cnInitStarLastname} filter as well as approximate matching on multiple token input, and to reduce the number of entries returned but making less use of approximate matching for short input and userids. The filter is:

\[
\begin{align*}
\text{if (single token input) /* i.e no spaces or dots */} \\
\text{if ((length of input < 6) AND (search base is organisation entry))} \\
\text{cnLeadSub OR snExact OR uidExact} \\
\text{else} \\
\text{cnAnySub OR cnApprox OR uidExact} \\
\text{else /* multiple token */} \\
\text{cnApprox OR cnInitStarLastname}
\end{align*}
\]

C: DE. This is very similar to that implemented by DE (a slight difference is forced by Quipu's dish program failing to handle some filters with multiple wild cards correctly). The \text{cnInitStarLastname} filter matches the first character of the input against the first character of the \text{commonName} attribute and matches the last token against the \text{surname} attribute. The filters are:

\[
\begin{align*}
\text{if (single token input) /* no spaces */} \\
\text{snExact, cnAnySub, cnApprox} \\
\text{else /* multiple token */} \\
\text{cnInitStarLastname, cnAnySub, cnApprox}
\end{align*}
\]

D: Modified DE. This version of DE uses fewer filters than that described in case C. It does not use approximate matching for short input, where the search base object is the organisation node. It does not use \text{cnAnySub} matching for multiple token input. The filters are:

\[
\begin{align*}
\text{if (single token input) /* no spaces */} \\
\text{if ((search base object is organisation) AND (length input < 6))} \\
\text{snExact, cnAnySub}
\end{align*}
\]
else
    snExact, cnAnySub, snApprox
else /* multiple token */
    cnInitStarLastname, cnApprox

The analysis of these filters is shown in Table 5.29.

<table>
<thead>
<tr>
<th>Filter code</th>
<th>% matched by filter</th>
<th>Mean no. res.</th>
<th>Med. no. res.</th>
<th>Max. no. res.</th>
<th>% queries with single res.</th>
<th>% queries with correct res.</th>
<th>% results correct</th>
<th>Mean no. of searches</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>90.23</td>
<td>71.35</td>
<td>2</td>
<td>3834</td>
<td>37.17</td>
<td>88.66</td>
<td>98.26</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>95.11</td>
<td>12.10</td>
<td>2</td>
<td>662</td>
<td>42.58</td>
<td>93.72</td>
<td>98.53</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>95.64</td>
<td>5.34</td>
<td>1</td>
<td>427</td>
<td>61.26</td>
<td>94.07</td>
<td>98.36</td>
<td>1.27</td>
</tr>
<tr>
<td>D</td>
<td>95.46</td>
<td>4.64</td>
<td>1</td>
<td>247</td>
<td>61.08</td>
<td>93.89</td>
<td>98.35</td>
<td>1.21</td>
</tr>
</tbody>
</table>

Table 5.29: Person name matching with various filters

The four strategies found some results for between 90.23% (case A) and 95.64% (case C) of the queries, with each strategy getting at least 98% correct results. The original UFN strategy is the worst in several ways. First, it found matching entries in the fewest cases. Second, it returned by far the most entries on average. Third, the largest result set was nearly 4,000 entries, almost a quarter of the database. Fourth, it got the fewest single entry result sets. The reason for this bad performance is primarily due to the use of cnApprox matching for every query. It got the fewest matches because it did not explicitly handle cases where the user entered a forename and surname, when the directory only had an initial and surname; bizarrely these are not approximately matched by Quipu.

The modified UFN algorithm is much better, being more precise and finding a correct result in nearly half the cases where the original UFN failed. This UFN strategy found approximately the same number of matches as the DE strategies.

Both DE strategies are better than the UFN strategies on two counts: they have lower average numbers of matches; they find more single entry result sets. The relative resolution cost of the different strategies is that the UFN queries use single search operations with compound filters, whereas the DE strategies use a sequence of searches with simpler filters. We examine the response times of these strategies in the NameFLOW-Paradise environment in Appendix G.

In following sections, we will examine ways of improving person name matching to see if we can do better than the algorithms in this section.

### 5.5.5 Reasons for Failing to Find a Person Entry

In this section, we summarise the main reasons why the various search strategies fail to find the required entry for some queries.
Misspelling: Some of the 50 misspelled queries were not found. We examine how successful approximate matching is at finding these entries in Section 5.5.6. We also see whether substring matching on truncated input is a viable alternative.

Too much input: The user’s query contained more name information than was stored in the directory. This problem is expanded upon in Section 5.5.7.

Unusual input formats: The user’s query was in an unusual format: e.g. surname first. Some steps for normalising unusual input formats are suggested in Section 5.5.8.

Minor format errors: The user may have entered a query such as “bob smith” when the directory entry is “Robert Smith”. The \textit{cnInitStarLastname} filter is of no use as the initials do not match. A solution for this problem is evaluated in Section 5.5.10.

Size limits: The DUA imposed a size limit on the result set, and the correct entry was matched but not returned. We investigate this in Section 5.5.11.

5.5.6 Approximate Matching Issues for Person Names

Approximate Matching and Spelling Mistakes

We noted in Chapter 3 that misspelling was a more serious problem with person names than with any other input category, with almost 7.5\% of input misspelled in some way. It follows that there is a commensurate need for some form of \textit{approximate} matching.

I tested how successful Quipu’s Soundex-based \textit{approximate} matching algorithm is at finding misspelled input. There were 50 misspellings in the sample. I conducted two experiments. First, the whole sample was matched, using \textit{snApprox} for single tokens and \textit{cnApprox} for multiple tokens. The second experiment was designed to avoid non-spelling matching problems, such as forename input not matching an initial in the directory; for this experiment the set was pruned to a set of 42 queries which had misspelled surnames, the queries were modified so that they were surname-only queries, and matched using a \textit{snApprox} filter. The success rate of \textit{approximate} matching with misspelled person name input is similar to its effectiveness with other input categories.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
Outcome of approximate match & Freq. as \textit{approximate} match & \textit{approximate} match \\
\hline
Full input & Surname only \\
\hline
Found correct entry & 68.00 & 71.43 \\
\hline
Found incorrect entry & 10.00 & 9.52 \\
\hline
Failed to find anything & 22.00 & 19.05 \\
\hline
\end{tabular}
\caption{Person name spelling mistakes and approximate matching}
\end{table}
CHAPTER 5. EXPERIMENTS WITH NAME MATCHING

Approximate Versus Truncated Substring Matching

We have discovered for other categories of input that substring matching on truncated input is an effective alternative to or companion technique for approximate matching for matching hard-to-find queries. Table 5.31 shows how the two techniques compared for misspelled surname input. The 42 misspelled surnames were matched using $snApprox$ matching, and $snLeadSub$ matching on the names truncated initially to four characters, and subsequently to three characters.

<table>
<thead>
<tr>
<th>Search filter</th>
<th>Percentage of queries</th>
<th>Ave. no. results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>correct</td>
<td>incorrect</td>
</tr>
<tr>
<td>snApprox</td>
<td>71.43</td>
<td>9.52</td>
</tr>
<tr>
<td>snTrunc (4)</td>
<td>38.10</td>
<td>19.05</td>
</tr>
<tr>
<td>snTrunc (3)</td>
<td>50.00</td>
<td>21.43</td>
</tr>
<tr>
<td>snApprox, then snTrunc (3)</td>
<td>78.57</td>
<td>11.90</td>
</tr>
</tbody>
</table>

Table 5.31: Percentage accuracy for approximate and truncated string matching

Truncation appears to be less successful with person name matching. It is appreciably less successful at finding correct entries and finds more incorrect entries. One reason why truncation is less successful with person names than it was with other input categories is that person names are shorter and spelling mistakes occur more in the first few characters. Although truncation does not seem to be a useful replacement for approximate matching, it may have some use as a follow-up technique as it found entries for a few misspelled queries that approximate matching did not find.

Queries Without Directory Entries

We conclude this examination of approximate matching by seeing how often $cnApprox$ matching returned entries for queries which had no entries in the directory: by definition, all these results are spurious. My sample included 108 queries in this category. The analysis is in Table 5.32.

<table>
<thead>
<tr>
<th>Query type</th>
<th>No department</th>
<th>Department specified</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single token</td>
<td>66.7</td>
<td>65.5</td>
</tr>
<tr>
<td>Multiple token</td>
<td>38.2</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 5.32: Percentage of $cnApprox$ queries returning results for which there is no directory entry

About two thirds of single token queries for which there was no entry got spurious matches. We will see in Chapter 6 that this tendency of Quipu's Soundex algorithm to return spurious results reduces confidence in approximate matching.
5.5. MATCHING PEOPLE NAMES

5.5.7 Too Much Person Name User Input

A common reason for matching failures for multiple token input is that a user enters more name information than is stored in the directory. For example, a user enters "Alan Smith", whereas the directory entry has the name "A Smith". Such a case fails to match for the cnExact, cnAnySub and cnApprox filters described in Section 5.5.2. While the cnExact and cnAnySub filters must fail by definition, it is surprising that Quipu's cnApprox matching does not handle this case. There is an extensive discussion of the requirements and capabilities of approximate matching in Chapter 6, and details of the Quipu implementation in Appendix D.

An even simpler case where match failures occur is when the user wrongly omits or includes a dot character after an initial: for example the entry is "A Smith", while the user makes a query of "A. Smith" or "A.Sm ith". In such cases Quipu does match entries approximately.

The cnInitStarLastname filter described in Section 5.5.2 handles both the above cases. However it has some limitations. The most serious is that it does not work in the above cases with misspelled surname input. We assess some possible alternative filters in Section 5.5.8.

Another problem is that it does not work if the first character of the user's input does not match the first character in the directory. There are several reasons for this. First, either the input (unlikely) or the directory name may start with a personal title. Second, the person may be commonly known by a forename other than their first name; possibilities include a second or third forename or a nickname. I evaluate one technique for solving this problem in Section 5.5.10.

5.5.8 Some Alternative Filters for Matching Multiple Token Person Name Input

We have noted in earlier discussion that there are a number of problems with matching multiple token input with directory names. In this section we look at some pre-processing of query formats that assist matching. We also evaluate some filters which could be used for matching multiple token input. The comparison of the techniques highlights the relative severity of the matching problems.

We noted in Section 5.5.2 that the large variety of person name forms in user queries and in the directory means that the cnInitStarLastname filter is a useful basic technique that assists with multiple token matching. However, this technique does not cope with some of the less common formats. An examination of user input suggests that the following steps should be used to remove inconsistencies in multiple token person name input. These steps would get almost all query names into a "standard" forenames-or-initials surname order:

1. If the name contains more than one comma, treat the name as a UFN containing name parts other than a personal name.

2. Convert any dots to spaces.
CHAPTER 5. EXPERIMENTS WITH NAME MATCHING

3. Convert any underscores to spaces.

4. If the name begins with a personal title, remove it.

5. If the name contains a single comma, take the portion after the comma, append a space, and then append the portion before the comma. For example, “barker, paul” becomes “paul barker”.

6. If the shortest token is two characters long, assume it to be concatenated initials, and separate the initials with space characters.

7. If the first token is three or more characters long, and the remaining tokens are initials, move the first token to the end of the input. For example, “barker p f” becomes “p f barker”.

A DUA designer has to decide whether he/she wishes to allow a DUA user to confirm the correctness of any transformations.

Since person entries’ commonName attributes may not include all forenames/initials, discarding tokens other than the leading initial and the surname simplifies the task of matching.

I evaluated six alternative filters for matching multiple token input: the experiments are described in detail in Appendix C.4. The two most effective filters are:

\textbf{cnInitStarSnTrunc} The first character is matched against the beginning of the commonName attribute and the first three letters of the last component are matched against the leading characters of the surname.

\[(cn=\text{firstChar}^*) \text{ AND } (sn=\text{truncatedLastWord}^*)\]

\textbf{cnApproxInitLastname} A name is constructed from the first initial and the last name, separated by a space, and this name is approximately matched. The filter is thus:

\[cn^"=\text{firstChar lastWord}\]

The key issue is how good these filters are at finding the entries not found by techniques such as \textit{cnInitStarLastName}, which we have evaluated earlier. The truncation technique above correctly matched 27% of the queries not matched by the \textit{cnInitStarLastName} filter, with a comparable improvement of almost 70% for the second of the two strategies. The cost of this greater matching success is that both these strategies also get more spurious matches.

\subsection*{5.5.9 The Role of the \textit{cnAnySub} Filter for Person Name Matching}

In this section I examine whether the \textit{cnAnySub} is a useful search tool, both for single and multiple token queries. There are good reasons for believing that it may be preferable to avoid using this type of filter if possible. First, it is relatively difficult to provide indexing support for this type of matching, and thus response times are likely to be slower. Some authors say that:
5.5. *MATCHING PEOPLE NAMES*

it is almost impossible to implement "inversion tables" for the substring test [AH92].

Lending support to this contention, neither the original Quipu nor DEC's DSA implements indexing for *any substring* matching. Performance figures for the DEC implementation show that this type of matching is indeed much slower than matching types supported by indexing [Eme95].

However, contrary to these arguments, indexing support for *any substring* matching has been implemented in some later versions of Quipu [How95b]. Furthermore, we will see evidence in Appendix G that this type of search operation is usually not much slower than other operations.

On balance, we can conclude that there may be a performance penalty for *any substring* matching using some implementations. The penalty will be worse for large databases.

Another reason for wanting to avoid this type of search is that DBAs may refuse to perform non-indexable searches\(^1\), since this type of search requires more processing by the DSA, which can lead to denial of service to other search requests if the service is heavily loaded.

In Table 5.23, we saw that there were 10.3\% single and 3.3\% multiple token queries that did not yield any results for *snExact* and *cnExact* filters respectively, but that were matched using a *cnAnySub* filter. By examining the individual queries, we can see whether the results achieved by *cnAnySub* matching are reasonable, and if they could be achieved any other way.

**Single Token Person Name Queries**

Of the 34 single token queries that were not matched by *snExact* matching, but which were matched by *cnAnySub* matching, 33 were either leading forenames or truncated surnames. The only other case produced a match that was clearly spurious. This analysis suggests that the following filter might be an effective alternative to *cnAnySub* matching:

\[
\text{cnLeadSub OR snLeadSub}
\]

We can see from the figures in Table 5.33 that this filter produces considerably fewer results; by definition it produces almost the same set of correct results as *cnAnySub* matching, only missing matches for any forenames that are not leading forenames.

<table>
<thead>
<tr>
<th>Filter</th>
<th>Mean</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>cnAnySub</em></td>
<td>32.5</td>
<td>4</td>
</tr>
<tr>
<td><em>cnLeadSub OR snLeadSub</em></td>
<td>10.9</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 5.33: Average no. of results returned for any-substring and specific forename-surname filter

Since *leading substring* searches are readily indexable, there is a strong case for preferring this alternative strategy over the *cnAnySub* search for single token queries.

\(^1\) A team of developers at the University of Michigan implemented such a restriction.
Multiple Token Person Name Queries

There were only eight cases where a multiple token query that did not yield results with a *cnExact* search filter did find results with a *cnAnySub* filter. My interpretation of the results was that this type of filter usually got the correct results - it was probably wrong once. Although there are too few results to draw meaningful conclusions, the types of match are of interest.

**Surname:** One surname had three name parts: “Van de Koot”

**Middle forename and surname:** “kitty kwan” matched “Pui Yin Kitty Kwan”

**Initial and surname:** These appeared to be “accidental” matches, where the last letter of the forename was the same as the first letter: correctly as for “d knight” matching “David Knight”, but probably incorrectly for “s dodd” matching “Jonathan Soames Dodd”.

**Forenames:** Two tokens entered that exactly correspond to an entry’s forenames.

**Truncated surname:** A forename plus a truncated surname, as in “adrian will” matching “Adrian Wills”.

Of these categories, only the initial plus surname match occurred more than twice, and this type of query is better dealt with by other filters - see Section 5.5.8.

### 5.5.10 Dropping Forenames and/or Initials

One type of person name mis-match not handled successfully by *cnInitStarLastname* filters is when either the directory entry or the user input is of a familiar forename form, while the other uses the equivalent formal forename. In the majority of cases, this causes no problems: e.g. Steve and Stephen, or Vicky and Victoria. In some cases, however, the two forms do not start with the same letter: Bob and Robert, Tony and Anthony, Liz and Elizabeth, or even Peggy and Margaret. Another possibility is that a user may be known familiarly by his/her second or third forename. A further possibility is that the person is known by an adopted name that bears no relationship to their given name; some Chinese people do this while domiciled in western countries.

These problems suggest that it may be useful to drop all components other than what is believed to be the surname from a user’s input and to try matching on surname only.

I tried this for the 41 multiple token person names that were not matched by the *cnInitStarLastname* filter. The evidence from the sample is that the technique is marginally useful, finding three entries not found otherwise. In two cases the forename entered by the user matched the third forename in the directory entry. In the other case, it appeared as though the user had made a phonetic mistake between “Carrie” and “Kerry”.

There are other reasons why dropping the forename(s) and/or initial(s) may be a useful technique. In some cases, there is no commonName form in the directory that begins with either a forename or an initial. There are two principal categories where this occurs. First, a few
organisations have created people entries with commonName attributes of the form “surname, forename(s)” or “surname, initial(s)”. Second, some organisations have created people entries with commonName attributes of the form personalTitle forename surname, such as “Mr Paul Barker”.

The best solution is for data administrators to create a number of alternative common name forms. These should include, as a minimum, a form which starts with the first forename, or its initial, and ends with the surname.

5.5.11 Use of Size Limits

We saw in Section 5.5.2 that in some circumstances, approximate matching can lead to large numbers of results. This may be unacceptable for a variety of reasons: too many results may be user-unfriendly; it may also be inefficient if network bandwidth is limited.

One approach that a DUA can use to restrict the number of results returned is to specify a size limit with the query. A problem with this strategy is that the required entry may be excluded from the result set.

I examined this problem by simulating restricting result sets to 20 and 100 entries for the cohort that suffers most from large result sets: single token, no department name specified.

I calculated the proportion of single token, no department specified queries where one or more directory entries were matched, but where the desired result would be excluded from a restricted result set. The results of this analysis are shown in Table 5.34.

<table>
<thead>
<tr>
<th>Search filter</th>
<th>Limit of ...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>20 results</td>
</tr>
<tr>
<td>snApprox</td>
<td>33.6</td>
</tr>
<tr>
<td>cnApprox</td>
<td>42.2</td>
</tr>
</tbody>
</table>

Table 5.34: Percentage of single token queries returning results where correct result is lost due to restricting the size of the result set

The results show that restricting the result sets to 20 entries can lead to a high proportion of correct results not being returned to the user, over 40% for the sample data for cnApprox matching. Restricting the result set to 100 entries still meant that 10% or more of correct cnApprox matched entries would not be returned to the user. Although several DUAs have used this technique to restrict result sets to manageable numbers, the approach has some severe drawbacks.

Fortunately, the pagedResults feature specified in the 1993 standard (and described in Chapter 2) helps solve some of these problems by allowing the user to request that results be sent a few at a time.
5.5.12 Improved Querying Strategies for Person Names

In Section 5.5.4, we examined the capabilities of two UFN-based and two DE-based querying strategies. In this section, we see whether we can improve further on the performance of those strategies by implementing some of the ideas discussed in the intervening sections. We look at one UFN-based and one DE-based strategy.

**A: Improved UFN** This is based on the UFN filter (case B) in Section 5.5.4. It further reduces the size of the average result set by using \texttt{snApprox} instead of \texttt{cnApprox} matching for single token queries, and uses the most effective filter for matching multiple tokens (case F in Section 5.5.8. The filter is:

```plaintext
if (single token input) /* i.e no spaces or dots */
  if ((length of input < 6) AND (search base is organisation entry))
    cnLeadSub OR snExact OR uidExact
  else
    cnLeadSub OR snLeadSub OR snApprox OR uidExact
else /* multiple token */
  if necessary get query into forename surname order
  modify input to first character followed by last token
  cnApprox
```

**B: Improved DE** This version of DE is based on case D in Section 5.5.4. It uses the more economical \texttt{cnLeadSub OR snLeadSub} filter instead of \texttt{cnAnySub}, and also uses the improved multiple token approximate matching filter discussed as case F in Section 5.5.8. The filters are:

```plaintext
if (single token input) /* no spaces */
  if ((search base object is organisation) AND (length input < 6))
    snExact, cnLeadSub OR snLeadSub
  else
    snExact, cnLeadSub OR snLeadSub, snApprox
else /* multiple token */
  if necessary get query into forename surname order
  modify input to first character followed by last token
  cnInitStarLastname, cnApprox
```

These filters improve on the filters discussed in Section 5.5.4 in two ways. They find about half the entries not matched by the earlier best attempts for both the UFN and DE filters. The improvements in both algorithms over their original forms at finding correct entries is statistically significant at the 1\% level: McNemar's test gives a \textit{z} value of 2.59 (a \textit{P} value of less than 0.01) for the improvements to the DE algorithm and a \textit{z} value of 5.12 (a \textit{P} value of less than 0.001) for the improvements to the UFN algorithm.
5.6. SPECIFYING THE WRONG DEPARTMENT NAME

Both improved algorithms also return smaller result sets than the original versions. Furthermore, the DE search strategy uses fewer operations than the versions proposed in Section 5.5.4.

An inspection of those queries not matched suggests that it would be hard to improve much on the performance of these algorithms without getting more spurious matches. Most of the unmatched queries are misspellings that are not matched by the Soundex algorithm. We look at possible alternatives to Soundex in Chapter 6.

### 5.6 Specifying the Wrong Department Name

We saw in Section 5.5.2 that one of the ways for a user to restrict the number of results returned for a person name query is to specify a department name to tighten the focus of the query. This usually narrows the query to a few tens or hundreds of entries to be examined, as opposed to the thousands or tens of thousands of entries in a large organisation. A potential problem with this strategy is that the user may not identify the correct department. There are two aspects to this problem. First, a user may precisely specify a department, but it is the wrong one: for example the user specifies "computer science" when they wanted "Computer service". Second, a user may enter a department name which is semantically correct, but that nevertheless matches the wrong department; a bizarre example of this is that "maths" does not match "Mathematics", not even approximately if we use Soundex, but that it does approximately match "Medicine". How serious is this problem of mismatching of department names?

The first stage in assessing the severity of this problem was to select those queries where the user specified a department, and where I believed the person entry the user was seeking to be in the directory. I presumed an entry to be in the directory if the following conditions were met:

- If the person's name was a single token, the entry could be found by an \texttt{snExact} filter. Alternatively, if the person's name was multiple tokens, the entry could be matched by a \texttt{cnInitStarLastname} filter.

- The user's department name input was either correct, or at least semantically related to the correct department. For example, the user specified a department name of "medicine" whereas the true department appeared to be "Medical School".

<table>
<thead>
<tr>
<th>Filter code</th>
<th>% matched by filter Mean no. res.</th>
<th>Med. no. res.</th>
<th>Max. no. res.</th>
<th>% queries with single res.</th>
<th>% queries with correct res.</th>
<th>% results correct Mean no. of searches</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>97.56</td>
<td>8.65</td>
<td>2</td>
<td>235</td>
<td>41.19</td>
<td>95.64</td>
</tr>
<tr>
<td>B</td>
<td>97.91</td>
<td>3.57</td>
<td>1</td>
<td>99</td>
<td>62.48</td>
<td>96.16</td>
</tr>
</tbody>
</table>

Table 5.35: Improved Person Name Matching Filters
CHAPTER 5. EXPERIMENTS WITH NAME MATCHING

This procedure resulted in a sample size of exactly 250 queries. I tried two different strategies to assess whether the user was likely to search the correct department.

The first strategy I tested was the search strategy used by DE, using the filter sequence: ouExact, ouAnySub, ouApprox. If none of these filters returned any results, DE's strategy is then to ignore the department name information and to resort to an organisation-wide search. The results are shown in Table 5.36.

<table>
<thead>
<tr>
<th>Search filter</th>
<th>Queries where results include</th>
<th>the correct match</th>
<th>all incorrect matches</th>
<th>all matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>ouExact</td>
<td></td>
<td>134</td>
<td>10</td>
<td>144</td>
</tr>
<tr>
<td>ouAnySub</td>
<td></td>
<td>67</td>
<td>5</td>
<td>72</td>
</tr>
<tr>
<td>ouApprox</td>
<td></td>
<td>13</td>
<td>5</td>
<td>18</td>
</tr>
<tr>
<td>Totals for DE</td>
<td>search method</td>
<td>214</td>
<td>20</td>
<td>234</td>
</tr>
<tr>
<td>ouAnySub</td>
<td>leading 4 chars</td>
<td>237</td>
<td>8</td>
<td>245</td>
</tr>
</tbody>
</table>

Table 5.36: Number of person name queries where department query correctly identifies correct department

Exact matching on the department name input found at least one department entry 144 times out of the 250, but on 10 of those occasions it was the wrong department. Overall, at least one of the filters returned some results in 234 of the 250 queries; in the other 16 cases, the person's entry could be found by doing an organisation-wide search. Unfortunately, for 20 of the queries that did match some directory entries, the result sets did not include the correct department.

The problem of a mis-specified department name is evidently quite serious if we use DE's matching strategy, if as many as 8% of department matches do not identify the correct department. The problem may be more serious than that, since if more than a single department match is found, the user may then have to select the correct department to search - the DE search strategy forces a choice in these circumstances. In fact, using the DE search strategy, this was not too great a problem with the data I examined, as we can see from the results in Table 5.37. All but one of the ouExact queries, which matched the correct entry, matched that entry alone. 189 of the 214 departments correctly identified by DE's search strategy were matched alone (80.8% of the time a match was made), while 25 correct matches were in result sets including other departments. On average the DE search strategy returned 1.41 department entries per query.

The other strategy tested was to do an ouAnySub match on the first four characters of the user's purported department name; in Section 5.4.6 we saw that this was an effective matching technique for department names. Table 5.36 shows that this technique successfully matched the
5.6. **SPECIFYING THE WRONG DEPARTMENT NAME**

<table>
<thead>
<tr>
<th>Filter</th>
<th>No. queries that matched</th>
<th>Single correct result</th>
<th>Correct res with other res</th>
<th>All results wrong</th>
<th>Ave No. results</th>
</tr>
</thead>
<tbody>
<tr>
<td>ouExact</td>
<td>144</td>
<td>133</td>
<td>1</td>
<td>10</td>
<td>1.01</td>
</tr>
<tr>
<td>ouAnySub</td>
<td>72</td>
<td>46</td>
<td>21</td>
<td>5</td>
<td>1.82</td>
</tr>
<tr>
<td>ouApprox</td>
<td>18</td>
<td>10</td>
<td>3</td>
<td>5</td>
<td>3.06</td>
</tr>
<tr>
<td>Totals for DE search method</td>
<td>234</td>
<td>189</td>
<td>25</td>
<td>20</td>
<td>1.41</td>
</tr>
<tr>
<td>ouAnySub leading 4 chars</td>
<td>245</td>
<td>55</td>
<td>182</td>
<td>8</td>
<td>3.80</td>
</tr>
</tbody>
</table>

Table 5.37: The precision of the department name matching techniques

A correct department 237 times out of 250, and incorrectly identified a department only 8 times. These results compare very favourably with the 214 correct and 20 incorrect for the DE strategy. However, while matching the first four characters of the user's input is good at finding the correct department, it also finds many incorrect ones as well. Table 5.37 shows that the correct department is identified alone only 55 times (only 22.4% of the time when at least one match is found) and that the average number of matches returned is 3.80 departments per query. If a user is forced to select one of these matches, the technique of matching on department name prefixes can only be regarded as successful so long as users are able to correctly identify the required department from the offered list. My hunch is that this is quite likely to be the case; it is generally easier to recognise names than to remember them.

Before trying to draw some conclusions on this topic, it is worth remembering that the data I have used here may well under-estimate the severity of the problem of users identifying departments. The data I have is for users within an organisation looking for people in departments within their own organisation. Even if we cannot say for sure that their knowledge of department names is good, we can be confident that it is likely to be better than for people querying from outside that organisation.

I suggest the following approach. The DE strategy is a good initial strategy as it automatically prefers exact matches over less good matches, and offers good precision. Since the DE strategy sometimes selects the wrong department, a user must be allowed to override the initial selection. The truncation strategy could then be used to select other possible matches. Preferably a DUA should search all the matches identified by truncation, in parallel or sequentially. At worst, a DUA should make it simple for a user to cycle through the list of departments matched while searching for the person entry. Finally, a DUA should allow a user to drop the department name altogether and do an organisational subtree search; users cannot be expected to get the department name right every time. A DUA should also do an organisation-wide person search if the search strategy
does not find any department matches.

5.6.1 Summary of Findings on Person Name Matching

- Whereas country, organisation and department queries have to be matched against tens or hundreds of entries in a DIT currently of limited size, person name matching has to work against organisations in the directory with over 100,000 person entries.

- The emphasis is on devising matching algorithms that find the correct entries, but that do so without returning too many spurious entries.

- To restrict the size of result sets, a user should supply a department name, should specify more than a single name token, and the DUA should not use approximate matching, or only use approximate matching as a technique of last resort.

- User queries are not often the same as entry RDNs: for my sample, this was true for one query in eight.

- We can largely solve the problem of the mismatch between initials in the directory and forenames in queries (or vice-versa) by reducing input to an initial and surname and using \textit{cnInitStarLastname} matching. Compared with approximate matching, the technique gets more results, and the results are more accurate.

- Approximate matching found about two-thirds of misspelled input (the same success rate as for other categories).

- Truncating hard to match queries to three or four letters was not a very successful alternative to approximate matching: human names are shorter than organisation and department names and spelling mistakes in human names occur nearer the start of the input.

- Using size limits to restrict the size of result sets means that many correct results will not be returned to the user if the database is large.

- \textit{cnAnySub} matching is better replaced by filter of the form \textit{snLeadSub OR cnLeadSub} for single token queries.

- Dropping forenames from multiple token input only marginally improved matching success with the test data; it might be useful as back-up technique to be used as it copes with databases with unusual name formats.

- Users do not always get the department name right. If a user selects the wrong department, even a good person entry search strategy will not find the required entry.
5.6. Conclusions on Person Name Matching

Many organisations in the directory have tens or hundreds of thousands of people entries. The size of organisational databases means that the emphasis on person name matching is to find algorithms that find the right entry but which do NOT return too many spurious entries. An analysis of the basic matching types show that it is best to avoid using approximate matching if the following conditions obtain:

- the user's query consists of a single token (usually a surname), particularly if the input consists of fewer than six characters;
- the user does not supply a department name, which narrows the search to a portion of the database.

It would clearly help matching algorithms if a DUA knew roughly how big a database was before constructing its search filters. The need to avoid the more profligate filters is most pressing for large databases.

As for other categories, the DE strategy was better at restricting the size of the result set than the UFN approach. The main reason for this is that the original UFN algorithm always uses approximate matching, and this tends to produce large result sets.

We have evaluated a number of improvements to the UFN and DE algorithms. A key improvement in the UFN algorithm, already implemented in DE, is to normalise multiple token user input to a form consisting of a leading initial and a surname.

The technique of truncating user input to a few characters and then trying substring matching is less successful for person names than it was for other categories. The main reason for this is that truncation is mostly useful with the other types of matching for dealing with form errors rather than spelling errors; the rate of misspellings for person name input is much higher than for other categories.

It is possible, by incorporating the best suggested improvements to the algorithms, to get both the UFN and DE strategies to find the correct answer for over 95% of all queries. It is harder to get this figure close to 100% than for other categories due to the higher proportion of person name misspellings.

Unlike other categories, there is little scope for the read then search strategy; only 12% of queries in my sample queries are RDNs. In fact the scope is less than this as a read strategy can only work if the user also supplies a department name as well.

A DUA designer should also be aware that however good his/her person name matching strategy is, it can only work if the search is made on the correct part of the database. While it is generally beneficial for users to enter department names, as this reduces the scope of the search and the consequent size of result sets, it is counter-productive if users enter the wrong department name. A search strategy must allow a user to override a department name match, and search the whole organisation. Doing this in DE does more to find correct matches than all the algorithm
improvements incorporated in the revised DE in Section 5.5.12.

5.7 Conclusions on Name Matching Experiments

In this final section of this chapter, we attempt to draw together the main themes of the experiments with matching algorithms. While some lessons hold true for all categories of input, we have also discovered that each category of input has to be treated individually: each category has its quirks.

One area where there are notable differences between the various input categories is the number of entries that a query is matched against. There are, and will remain, relatively few country entries in the directory, and department entries within any given organisation. For the moment, there are also relatively few organisations in the directory, but this will change if the directory expands as anticipated to include most if not all organisations. However, there are already databases in the directory with over 100000 person entries. Databases of this size pose the classic matching problem: how to match the correct entries but not to match many spurious entries, or, to use the normal terminology, how to get good recall while maintaining good precision.

A way of improving matching algorithms that applies to all input categories is to do some pre-processing of user input. Organisation and department name queries often include “stop list” words that add little to the semantics of a query, and yet which may prevent a successful match. Matching person names is made much easier by reducing queries to initial surname format.

It is useful to handle short input specially. Short organisation and department names are often sets of initials, and these forms may not be stored in the directory: the input can be split into a set of single character tokens and then matched using substring or approximate matching techniques. Short names also tend to match more entries: a DUA designer should consider treating short input differently if it is important for the querying algorithm to have good precision.

We have also found that approximate matching often leads to large result sets. This tendency is exacerbated with short input. If precision is important, then approximate matching should be used as sparingly as possible. One solution that works well for hard-to-match organisation and department name queries is to truncate the input to a few characters and then using substring matching on the shortened input. However, this technique does not work well for person names, which suffer from more and worse misspelling. Since it seems as though we cannot do without approximate matching, we will investigate some alternative approximate matching techniques in the next chapter.

We have examined evidence on three querying strategies: read then search, UFN and DE. While the relatively low proportion of queries that are equivalent to RDNs suggests that read then search will not be very efficient, we have also seen that if we use this technique with name-to-RDN name mapping tables, this sort of strategy can be effective at reducing the number of operations on the directory. However, the exact matching proposed by Afifi and Huitema is
5.7. CONCLUSIONS ON NAME MATCHING EXPERIMENTS

clearly inadequate for matching many queries.

A problem with the UFN algorithm as specified in RFC 1781 is that it always uses approximate matching, and is thus prone to large result sets. However, we have been able to suggest improvements to the algorithm so that it offers better precision and, in some cases, better recall as well.

Our attention has mostly been turned towards improving DE's algorithm. Although there are trade-offs between recall, precision and use of directory operations, we have been able to identify improved sequences of search filters that, cumulatively, would allow 20% more correct person entries to be found, mostly due to improvements in organisation and department name matching. This improvement is achieved hand-in-hand with slightly better precision and slightly fewer search operations. The number of directory operations can be further reduced by using mapping tables within DUs to resolve popular queries to their corresponding RDNs.

Finally, querying algorithms should be flexible. DE's department name matching algorithm matched the wrong department name in 8% of queries where a department name was specified. Users are not infallible and confuse names such as "Computer Science" and "Computer Service", and so on. If the wrong department is matched there is then no chance of finding the appropriate person entry. In some cases, matching a person entry may be best served by ignoring what the user has entered as department name input altogether! It is up to the DU designer whether this is done automatically by the DU or by suggesting it to the user.
Chapter 6

Approximate matching

6.1 Introduction

Although when I started on this thesis I did not intend to study approximate matching algorithms in any detail, the need for a better understanding of approximate matching algorithms emerged as work on other chapters progressed. There are two main reasons why a further study of approximate matching algorithms is useful. First, we identified in Chapter 3 that there is a substantial proportion of input, especially of human names, which is misspelled. While we also saw that we can use techniques such as truncating user input to match entries even when input is misspelled, a number of cases remain where some form of approximate matching is required to help find entries. However, we discovered in Chapter 5 that Soundex [Knu73] approximate matching, as implemented in Quipu, seems to be a rather blunt instrument and tends to produce large numbers of matches. Furthermore, anecdotal evidence, from directory users and experts alike [Wau95], is that many of these results look inexplicable. This tendency to produce peculiar, confusing results has caused some DUA designers to stop using approximate matching at all. Maybe the problems are solely due to the way that Soundex is implemented in Quipu? Or, maybe Soundex is a poor choice of algorithm? The bulk of this chapter is an assessment of Soundex and various alternative algorithms.

A second problem occurs when a user's input is correctly spelled but does not match the form of the name in the directory entry. An example of this type that we considered in Chapter 5 is where the user's query is specified as "Foobar University" while the entry name is "University of Foobar". A further example is to consider whether a user query of "P Barker" should approximately match the following possible directory names: "Paul Barker"; "P. Barker"; "P F Barker"; or "Barker, P". Although we can solve all these matching problems by appropriate transformations of the user input and/or directory names, it seems reasonable to the author to expect that an approximate matching algorithm should match the given user input to any of the example names. We do not address this issue further in this chapter. However, Appendix D gives a number of examples of form mis-matches that a DSA could usefully handle.

The structure of the chapter is as follows. Section 6.2 contains an extensive review of the approximate matching problem. Despite the prevalence of Soundex as an approximate matching algorithm, we will see that many other techniques have been proposed. However, we will also see that the effectiveness of the various techniques varies according to the application and its data; no one technique suits all approximate matching requirements. Section 6.3 describes some experiments with four algorithms: Soundex, which is used in several directory service implementations; Metaphone [Phi90], which has been used in DSAs authored by a team at the University of Michigan; a technique first proposed by Damerau [Dam64] which focuses on correcting typographical
errors; and an algorithm by Bickel [Bic87] which assigns weights to letters to construct likeness values. In these experiments we examine the precision and recall of these algorithms, as well as their computational overhead. Finally we draw conclusions in Section 6.4.

6.2 Approximate Matching Problems and Techniques

In this section we review the literature on approximate matching. We will see that there are many techniques that could be used. As we discuss the various techniques, we examine evidence from work elsewhere in this thesis to test the applicability of an idea or a particular technique. We also note which techniques are actually used in directory services in use today.

There is a huge literature on the detection and correction of misspellings. Most of the literature focuses on correcting text, usually English text, and much less on names specifically. However, the techniques for correcting English words and names are in some cases so similar that they are often discussed together in literature surveys: see, for example, the surveys by Hall and Dowling [HD80] and by Kukich [Kuk92]. Not all techniques are applicable to directory user input. For example, several techniques have been developed for correcting text scanned by an OCR. For a detection and correction technique to work well, it must be tuned to the types of errors in the names or words in the particular application.

6.2.1 Types of Misspelling Error

Kukich notes that a distinction is sometimes drawn between three types of misspelling error: typographical, cognitive and phonetic. It is not always possible to categorise an individual error: e.g, if the correct name is "Barker", is "Parker" a typographical or a phonetic error? However, the categories are useful conceptually to help us consider what may be the most appropriate technique for matching misspelled input.

Typographical Errors

Typographical, or motor coordination errors, are frequently described in the literature. The assumption is that the person typing knows the correct spelling but their lack of skill as a typist results in an incorrect spelling. A seminal piece of work in this area is that by Damerau [Dam64]. He noted that over 80% of all spelling errors fall into one of four classes of error.

Omission The typist omits a single character.

Insertion The typist inserts a single extra character.

Substitution The typist types the right number of characters but gets one character wrong.

Transposition The typist transposes two successive characters.
These errors are sometimes referred to as single error misspellings. In the same vein, Wagner has classified this type of spelling error as having an *edit distance* of one [Wag74]; there is a discussion of edit distances in Section 6.2.3. For the sake of conciseness we refer to *Damerau matching* or *Damerau errors* when we are considering matching a query that has a single typographical error.

Damerau's finding that at least 80% of spelling errors are one of these types is confirmed in studies by Peterson [Pet86] and Pollock and Zamora [PZ84]. Other types of spelling error involving multiple characters are noted by Peterson: two letters inserted, two letters omitted, two letters transposed around a third character. However, these errors are addressed much less than in the literature of spelling detection and correction. This is understandable for two reasons: first, these multiple character errors occur much less frequently; second, such spelling errors are much harder to correct.

Grudin has studied typing mistakes [Gru83], and observed that experts predominantly make insertion errors, by striking an adjacent key in addition to the correct key, while novice users make more substitution errors than any other type.

I have presented evidence elsewhere [Bar95a] that users make a lot of errors when typing names. An analysis of user input to an early version of the DE user interface showed that 6.8% of ordinary queries and 8.1% of UFN queries contained control characters that resulted from trying to edit the input. These control characters were either back-space characters or character sequences generated by the arrow keys; the UNIX standard input routines require the use of the DELETE character to erase input. If we assume that many users were able to rectify their typing errors correctly, it suggests that users make a lot of mistakes while entering names.

On the basis that it is surely easier to detect a typing error in a string of a few characters than to detect the same error embedded in a page or screenful of text, it would not be surprising if our sample of directory query data did not follow the pattern described by Damerau. However, for country, organisation and department names, over 80% of misspellings were Damerau errors. People's names are clearly a bit different; 60% of person name misspellings were Damerau errors.

**Cognitive Errors**

Cognitive errors occur when the user has a misconception about how a word or name is spelled. This is particularly problematic when the name originates in a different ethnic group to that of the person trying to spell the name; there is often little understanding of the spelling rules for foreign names. It is easy to see that an Anglo-Saxon English person might have difficulty spelling the following names from the UCL database: Chyriwsky, Avwenagha, and Warnakulasuriya.

These cognitive problems are further muddied by different transliterations: for example, consider Peking and Beijing.

One measure of the cognitive problem of spelling names can be deduced by considering digram frequencies. A digram, or bigram, is a pair of letters occurring sequentially in words or names.
For example, the name "paul" has the digrams "pa", "au" and "ul". Riseman and Hanson [RH74] report a claim by Sitar that over 40% of possible digrams do not occur in English. However, I found that only 13.6% possible digrams did not occur in the 16174 names in the UCL database. This greater diversity of permissible spellings suggests that the cognitive problem of spelling names is harder than for language in general.

**Phonetic Errors**

A phonetic error is a special class of cognitive error, where the writer misspells a name (or word) but the misspelled name is phonetically correct. In some cases, the writer may have a cognitive grasp of the way that the name is spelled, but opts for the wrong spelling. For example, the writer types "Sheppard" instead of "Shepherd", or "Haynes" instead of "Haines". In other cases, a user may have no idea of how a name is spelled but, knowing how the name is pronounced, makes a phonetically correct but misspelled guess.

While some writers make the distinction between typographical, cognitive and phonetic errors, Kukich notes that many writers do not. Fortunately many detection/correction techniques deal with all three cases. However, Kukich also notes that there is a tendency for phonetic misspellings to distort words more than the other error types.

There is some evidence that some applications do suffer from a lot of phonetic spelling errors. Specifically, phonetic errors have proved to be common in two directory service applications [Boi81] [Osh88]. Furthermore, in a study that shows how we think in sounds, Mitton found that 44% of errors in a corpus of hand-written student essays were homophones [Mit87].

Directory users who are likely to particularly benefit from phonetic matching support are telephone switchboard operators, or receptionists, who hear a name and then have to enter that name into a directory system to find the required phone or room number. Research demonstrates the usefulness of phonetic matching support for this task, although operators can always ask callers to spell the required person's name. A study by Van Berkel and DeSmedt [BD88] showed that subjects misspelled 38% of names which they were transcribing from a tape-recording of a set of randomly chosen names, and yet the misspelled names were phonetically plausible.

The evidence from my own work is that Damerau typographical errors form over 80% of errors for input categories other than person names. The misspelling rate for person names is much higher, 7% compared with 3% for the other categories, but the proportion of these errors that are typographical is lower at roughly 60%. For person names, many of the other errors are phonetically similar or identical to the target words: e.g., "Orme" for "Oram", "Seymore" for "Seymour", "Pitten" for "Pittam", and "Kerry" for "Carrie". My conclusion is that phonetic matching has a definite, albeit small, role to play in approximately matching names.
6.2. APPROXIMATE MATCHING PROBLEMS AND TECHNIQUES

6.2.2 Other Issues in Misspelled Names

Errors in the First Character Position

Several studies show that not many errors occur in the first character position of words. Pollock and Zamora found that only 3.3% of 50,000 spelling errors had first letter errors (FLEs). Other studies have found even fewer FLEs: Yannakoudakis and Fawthrop [YF83] found only 1.4% FLEs, although Kukich found as many as 15% of spelling errors were FLEs. My own findings, reported in Chapter 3, are that there were 9 FLEs in 271 spelling errors for organisation, department and person names, an error rate of 3.32%.

The importance of FLEs is that if we can ignore them, we can reduce the search space to just those words or names with the same first letter. A few matches may now be missed, but the technique can speed up searches considerably.

This technique is widely used in practice, for example in the Soundex algorithm [Knu73], by Davidson [Dav62], Bourne and Ford [BF61], Bickel [Bic87], and Pollock and Zamora [PZ84].

Keyboard Adjacencies

There are strong keyboard effects in typographical errors, and it is possible to improve correction techniques by understanding the types of mistakes that occur most often. Grudin found that expert typists' most common errors are single letter insertions, caused by striking two keys almost simultaneously [Gru83]. He also found that the majority of novice typist errors are substitution errors, and that 58% of such errors involve adjacent typewriter keys. Another finding was that typists were more likely to substitute high frequency letters (such as the vowels) in error for a low frequency letter rather than vice-versa.

Researchers such as Church and Gale [CG91] have exploited these probabilities to improve error correction rates, by allowing ranking of typing errors with the same edit distance.

Word Length

It follows that if a large proportion of spelling errors are Damerau errors, then most misspelled words will be within one character of the correct length. Matching algorithms can exploit this fact to reduce the search space. The relationship between the length of phonetically correct misspellings and the target words is less direct. Kukich settles for the observation that "most misspellings tend to be within two characters in length of the correct spelling" [Kuk92].

Another area of interest is whether word length is a predictor of the frequency of misspellings. Studies have produced widely differing results. Some research has shown that few errors occur in short words [PZ84], while others have found that words of four characters or less account for the majority of misspellings [Kuk90]; the type of application has a huge effect on the nature of the misspellings. Pollock and Zamora also note that although three or four character word misspellings only formed 9.2% of total misspellings, these errors were hard to correct and generated
42% of their miscorrections [PZ84].

Context

A problem for correcting misspelled names is that there is often no context to help decide between plausible alternatives. This problem is classed as isolated word error correction. In some cases, a misspelled surname may have an accompanying initial or forename, which provides sufficient context to allow a choice between candidates to be made with reasonable confidence. However, I found (see Chapter 3) that single token queries are the most popular form of person name queries, and so we are often forced into isolated word error correction.

Kukich reported a test where humans were asked to correct a set of isolated misspelled words [Kuk92]. She found that humans on average performed as well as, but no better than, the best automated techniques. She suggested that maybe there is an upper bound on the effectiveness of isolated error correction techniques.

6.2.3 Correction techniques

In this section, we review some correction techniques. Four of these techniques, Soundex, Metaphone, Damerau and Bickel, get particular attention as these algorithms are compared in some experiments described in Section 6.3.

Before examining the algorithms, we should first note an important characteristic that differentiates these algorithms: that is whether or not they rank candidate matches. Two of the techniques described, Soundex and Metaphone, have no way of ranking candidate matches; the algorithm decides either that a candidate is a near match or that it is not. The other techniques described in this section all offer ways of ranking results, such that very near matches are offered ahead of not so near matches, ahead of poor matches. Despite the flexibility of the ranking approach, most directory service implementations use the non-ranking Soundex for approximate matching.

Soundex

The Soundex algorithm [Knu73] is the most widely used approximate matching algorithm for directory service applications. It is invariably cited in the literature of approximate matching: e.g., [Alb67] [HD80] [Kuk92]. It is used by several X.500 implementations: Quipu and its many derivatives, EAN [NBGS92], DEC, Datacraft, Messageware and Unisys. In a study I conducted on behalf of UKERNA into directory service policy in the UK academic community [Bar95b], I found that most sites running self-written directory service software used Soundex, if they provided approximate matching.

Two other factors weigh in its favour and help account for its popularity. First, it is included in Knuth’s reference work on algorithms [Knu73]. Second, several implementations of Soundex are freely available on the Internet.
6.2. APPROXIMATE MATCHING PROBLEMS AND TECHNIQUES

Soundex, by computing standards, has a long history, being patented in 1918 by Odell and Russell[OR18]. It is often spoken of as a phonetic matching algorithm. This is a misunderstanding due to the name Soundex. Some authors note that Soundex is more aimed at coping with spelling variations rather than being a phonetic matching algorithm [Mav95]. The algorithm is better regarded as a similarity key technique [Kuk92].

The method is to derive a key for a name as follows: the first letter of the key is the first letter of the name; subsequent letters in the name are mapped into numeric digits. These mappings are shown in Table 6.1.

<table>
<thead>
<tr>
<th>Soundex code</th>
<th>Letters</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>B, F, P, V</td>
</tr>
<tr>
<td>2</td>
<td>C, G, J, K, Q, S, X, Z</td>
</tr>
<tr>
<td>3</td>
<td>D, T</td>
</tr>
<tr>
<td>4</td>
<td>L</td>
</tr>
<tr>
<td>5</td>
<td>M, N</td>
</tr>
<tr>
<td>6</td>
<td>R</td>
</tr>
<tr>
<td>0</td>
<td>A, E, H, I, O, U, W, Y</td>
</tr>
</tbody>
</table>

Table 6.1: Soundex codes

Vowels and the soft consonants, 'H', 'W' and 'Y', are given a code of zero. Successive instances of the same code are collapsed into a single code, and after that (and the order of events is important) zeroes are removed to produce the final key. The examples shown in Table 6.2 should clarify the method.

<table>
<thead>
<tr>
<th>Name</th>
<th>Soundex key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barker</td>
<td>B626</td>
</tr>
<tr>
<td>Knight</td>
<td>K523</td>
</tr>
<tr>
<td>Attlee</td>
<td>A34</td>
</tr>
<tr>
<td>Livingstone</td>
<td>L152</td>
</tr>
<tr>
<td>May</td>
<td>M</td>
</tr>
</tbody>
</table>

Table 6.2: Example Soundex keys

The usual specification for Soundex says that a Soundex key should be four characters long: a leading letter plus up to three digits. We will see later whether arbitrarily long keys have much impact on the matching characteristics of the algorithm.

The algorithm is very fast to evaluate. If we consider its application in a directory service, we can see that Soundex keys for names in the directory can be pre-computed (at directory
CHAPTER 6. APPROXIMATE MATCHING

start-up time or as a housekeeping function). When a search is made, a user's input needs to be transformed into a Soundex key, and this key is then compared with the directory Soundex keys. These directory keys can be indexed and Soundex matching is often comparable in response time to exact string matching. Evidence for this is given in Section 6.3.7 and also in [Eme95].

Despite its simplicity Soundex is surprisingly effective at correctly matching misspelled input. One of the reasons why it is often successful is that it handles many typographical errors; vowel frequencies are on average higher than consonant frequencies, and vowel errors do not affect the Soundex key.

However, Soundex matching does not always work well. The author's impression, gained from talking to many people who have used Soundex matching in the directory service, is that a common perception is that it does a useful job, but that it often produces seemingly ridiculous matches, which reduces confidence in the system. (We will see later that this is largely due to prefix matching: see Appendix D.) As a result of reports like this, at least one group of prolific DUA implementors now configure their software by default not to use approximate matching. Damanjit Mahl from Brunel University said:

Approximate searches usually produced too many superfluous matches (especially when searching for person entries due to sheer quantity).

The main problem, as Kukich notes, is that Soundex's matching is very “coarse-grained” [Kuk92]. While Soundex generally offers good recall, it also suffers from poor precision: in other words, you often get what you want, but you also get a lot of what you do not want.

The main reason for this is that the key-space is small. There are \((26 \times 6 \times 6 \times 6)\) four character keys, \((26 \times 6 \times 6)\) three character keys, \((26 \times 6)\) two character keys and 26 one character keys, giving a total of 6734 keys. This is insufficient for medium and large size databases. For example, the UCL database which I used for much of the work described in this thesis has 16174 people entries, with 9681 unique names, but these map onto only 3880 keys. Larger organisations of 100,000 entries or more (such as the University of Michigan) already exist in the directory; for such databases the majority of names will not be mapped uniquely onto a Soundex key, hence the lack of precision.

While the ignoring of vowels and soft consonants pays dividends when finding entries corresponding to misspelled names, it is often too lenient and allows some obvious mismatching to take place. For example, the name “Seow” maps onto a single letter Soundex code of “S”, as do “Suuya”, “Shaha” and 73 other surnames in the UCL database. Worse still, the names “Fan” and “Fawehinmi” both map to the same Soundex code of “F5”.

These examples suggest that we might be able to improve the performance of Soundex by tightening up the algorithm a little. Indeed, some variants of Soundex have been developed.

We have mentioned that Quipu's implementation allows prefix matching. If prefix matching is enabled, a match is made if the Soundex key for the user input matches the leading substring (or prefix) of a candidate entry's Soundex key: e.g., the key 'A' matches the prefix of key 'A123'. We
will see in Section 6.3.3 that prefix matching has a very bad effect on the precision of Soundex. An effect of the decision to make prefix matching the default in Quipu has been to convince many directory users and managers that Soundex is a poor algorithm.

Soundzee, developed by George Hlavka, is identical to Soundex except that the first character is also coded [Mav95]. This does not look useful as a problem we have found with Soundex is that it is too indiscriminate, and Soundzee offers even looser matching.

Another approach is that adopted by Davidson, who modified the algorithm so that the four letters in the code are the first four consonants (excluding 'H', 'W', and 'Y') in the name [Dav62]. This makes the matching more stringent. He backed up this method by calling an ill-spelled names routine if no match was found initially: this fallback routine makes matches on the basis of the number of letters the keys have in common.

Other possibilities are evident. First, we could pay some attention to vowels and soft consonants. Second, we could insist that the length of two names are similar. We will examine these and other possibilities in Section 6.3.3.

**Metaphone**

Lawrence Philips has proposed an algorithm called Metaphone [Phi90] which is similar in approach to Soundex, but which uses a larger set of codes to represent letters, digrams and trigrams. It takes more account of the sound of letters in particular combinations, but still falls short of being a phonetic matching algorithm as it ignores vowels.

The algorithm is of particular interest as it has been used in DSAs developed by the University of Michigan: their team found that Soundex often produced too many matches. This was a severe problem at Michigan with its database of over 100,000 entries.

The assignment of codes to letters is more complicated than for Soundex: a few examples are shown in Table 6.3; the encodings are reproduced in full in Appendix E. Note that the encodings given in the Appendix, and used in the experiments described in Section 6.3.4, are slightly different to those used by Philips; the Metaphone code provided by the team at the University of Michigan was derived from another source [Par91].

As with Soundex, vowels are dropped unless they occur in the first position in the word. Unlike Soundex, the soft consonants are not always dropped, particularly if they occur next to a vowel. Metaphone treats certain word beginnings differently: for example, the digrams 'GN', 'KN', and 'PN' are coded as 'N' if they occur at the beginning of a word. See Appendix E for more details.

It should be apparent from this brief discussion that Metaphone is far more sophisticated than Soundex. The extra sophistication means that more information about a name is represented in a Metaphone key than its equivalent Soundex key. Not surprisingly there are far more Metaphone keys. There are 22 valid first character codes and 17 character codes for subsequent character positions. This gives a total of 114,840 Metaphone keys of up to four characters, more than
CHAPTER 6. APPROXIMATE MATCHING

<table>
<thead>
<tr>
<th>Letter</th>
<th>Code</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>if first letter</td>
</tr>
<tr>
<td></td>
<td>n/a</td>
<td>otherwise</td>
</tr>
<tr>
<td>B</td>
<td>n/a</td>
<td>no code if at end of word as in LAMB</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>otherwise</td>
</tr>
<tr>
<td>C</td>
<td>n/a</td>
<td>no code if -SCE-, -SCI- or -SCY-</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>if -CIA- or -CH-</td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>if -CI-, -CE, -CY-</td>
</tr>
<tr>
<td></td>
<td>K</td>
<td>otherwise</td>
</tr>
</tbody>
</table>

Table 6.3: Some of the Metaphone codes

seventeen times as many keys as Soundex.

Metaphone is fast, being comparable to Soundex for speed. Although the extra complexity means that the key generation is fractionally slower, key comparison is the same operation as for Soundex.

One potential problem with Metaphone is that the extra precision brought by the more sophisticated keys has a cost of a lower recall rate: in other words, you get much less of what you do not want, but you sometimes do not get what you do want.

Another problem is that Metaphone is highly tuned to the idiosyncrasies of English spelling and pronunciation. It is probable that it would work less well with databases with predominantly non-Anglo-Saxon names. Unfortunately I have no evidence as to how well it works for databases of other European names, let alone Indian or Chinese names. Mavrogeorge notes, however, that even English spelling is so idiosyncratic that it is easy to find contradictory cases even with complicated rules [Mav95]. Consider, for example, the ‘ch’ sound in the names “Richards”, “Buchanan” and “Crichton”.

Minimum Edit Distance

The prevalence of typographical errors, particularly single-error misspellings, has led to the development of minimum edit distance error correction techniques. The minimum edit distance is the number of insertions, deletions, substitutions or transpositions required to transform the incorrect string into the correct string. This edit distance is sometimes referred to as the Damerau-Levenshtein metric, after two pioneering researchers. It follows from Damerau's research reported in Section 6.2.1 that 80% of the spelling errors he found had an edit distance of one. Edit distances are usually integer values, although Veronis has produced a non-integer scheme based on phonemic similarity (a measure of much they sound alike) to weight candidate matches [Ver88].

Note that not all researchers include all four error types in their definition of edit distance. For example, Wu and Manber only allow insertions, deletions and substitutions [WM92].
Another possibility is to use keyboard adjacencies to weight candidate matches: e.g., the entry “clark” would be preferred over the entry “clare” for misspelled input of “clair” as “j” is nearer on the keyboard to “k” than to “e”.

Some researchers have used reverse edit distance techniques, where a list of candidate matches is generated and these are then matched against the lexicon to see if any of the candidates exist.

Boivie has used an edit distance technique in a directory service called da; this was used at Bell Labs. The technique is based on recursive searching of a virtual tree comprising candidate names [Boi81]. The technique allows the search tree to be pruned if the algorithm can calculate that part of the name tree cannot contain a better match than that already found. If two matches are found with the same edit distance, da prefers matches with the most leading characters in common with the user input. Thus, “pedersen” is preferred over “peterson” if the query is “pederson”. Boivie tested his program on a directory of 25,000 entries and found that its performance was more than adequate. He does not give detailed test results but notes that if the required entry was not found initially, it was generally ranked near the top of the list.

Wu and Manber [WM94] have developed a tool called glimpse which allows the user to specify the edit distance he/she will accept for approximate matches. Glimpse builds on some earlier work of theirs on a general approximate matching tool called agrep [WM92]. Although, glimpse was not designed for directory service use, Howes has expressed interest in using the algorithm in a directory service [How95b].

**Speedcop**

Pollock and Zamora’s Speedcop system [PZ84] is an interesting hybrid of ideas. It uses probabilities of certain types of spelling mistake, and uses these to rank possible single error misspellings. The method requires two keys to be generated for words in the lexicon and any misspelled words; a skeleton key and an omission key.

The skeleton key is constructed by taking the first letter of the name, then the remaining consonants in order of appearance, followed by the vowels in order of appearance. Repeated consonants and vowels are omitted.

The omission key is again consonants followed by vowels. The order of the consonants is determined by how often each letter was found to be omitted in misspelled words. For example, “R” was found to be omitted more frequently than “J”, and so “J” precedes “R” in an omission key. The reasoning is that “J” is typographically a more significant letter than “R”. The omission key bunches these typographically significant characters at the start of the key. The vowels follow in the order they occur in the name.

The skeleton and omission keys are first sorted. A search is first made on the skeleton keys, and, failing that, omission keys are tried. In both cases, candidate words should be close to the point in the sorted lists where the misspelled word would be placed. The search stops when either a single-error misspelling is found, or the search has exhausted the reasonable candidates.
The emphasis on single-error misspellings is a limitation of the technique. Their use of letter probabilities, however, is used by several other systems, including that by Bickel described shortly.

**Trigrams and Triphones**

Some other techniques have been built on the premise that parts of misspelled words are undisturbed by spelling mistakes, and will correctly match parts of the target word. Angell *et al* [AFW92] devised a system based on matching trigrams. They defined a function which was weighted according to the number of matching trigrams and the length of the two strings being matched. While the technique performed impressively with their test data, they noted that the technique does not work well with short words, where a single error may leave no matching trigrams.

A related technique has been proposed by Van Berkel and DeSmedt [BD88]. They were tackling the problem of phonetic misspellings, where people heard a name but could not spell it correctly. They developed a system of triphones, where a triphone is a set of three phonemes; a phoneme is an individual speech sound and may be based on one or more letters. Their technique worked very well with phonetic misspellings. They did not evaluate the technique fully for typographical errors, but noted that it was more effective than some alternative methods for this type of error. A limitation of their experiments is that they were done using a small set of names; 254 in all. It is not clear whether their techniques would be as effective with large lexicons or databases.

**Bickel’s algorithm**

Bickel has proposed a method based on unigram (single letter) probabilities; this is described fully in [Bic87]. The basis of Bickel’s algorithm is the assumption that, due to their rarity, letters such as “Q” and “Z” convey more information than more frequently occurring letters such as “A” and “S”. Furthermore, he argued, character omissions and insertions mean that algorithms based on positional matching of characters often fail.

Bickel’s method is to assign weights to characters, such that common characters are given low weights and less common characters get higher weights. These weights are given in Table 6.4.

A likeness value is achieved for two names by summing the weights for all the letters which occur in both names. As with Soundex, the first letter of both names must be the same for a match to be considered. Multiple occurrences of a letter are only counted once. The relative length of the names and their character ordering are ignored. The likeness value allows possible alternative matches to be ranked, whereas Soundex and Metaphone only produce a TRUE or FALSE outcome for each possible match.

The following illustration of the algorithm should help explain how it works. Let us assume that a user has entered a name of “Baker” and this is compared with three candidate names, “Barker”, “Baxter” and “Bakefield”.
6.2. APPROXIMATE MATCHING PROBLEMS AND TECHNIQUES

<table>
<thead>
<tr>
<th>Weight</th>
<th>Letters</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>A, E, I, N, O, S, T</td>
</tr>
<tr>
<td>4</td>
<td>D, H, L, R, U</td>
</tr>
<tr>
<td>5</td>
<td>C, F, G, M, P, W</td>
</tr>
<tr>
<td>6</td>
<td>B, V</td>
</tr>
<tr>
<td>7</td>
<td>K, Q</td>
</tr>
<tr>
<td>8</td>
<td>J, X, Y</td>
</tr>
<tr>
<td>9</td>
<td>Z</td>
</tr>
</tbody>
</table>

Table 6.4: Weights assigned to letters by Bickel matching algorithm

| B A K E R | 6 3 7 3 4 |
| B A R K E R | * * * * * |
| B A X T E R | * * * *
| B A K E F I E L D | * * * * * |
| 6 3 4 7 3 | 6 3 3 4 |
| 6 3 7 3 |

The maximum likeness value is a sum of all the weights in the name “Baker”, i.e. 23. The letters marked with an asterisk in the candidate words are those which occur in the name “Baker”. Every character in “Barker” is matched and so this achieves (by counting the ‘R’ only once) the maximum likeness value of 23. Similarly, “Baxter” has a likeness value of 16 and “Bakefield” a likeness value of 19. Thus “Barker” is judged to be the most likely correct match.

Bickel’s experiments were made on a database of just under 1000 names at the Johnson Space Center. He randomly modified these names to provide simulated misspelled queries for the cases of insertion of an extra letter, omission of a required letter and transposition of two adjacent letters. He found that with appropriate tuning that his algorithm was superior to Soundex in that it was better able to uniquely identify the correct result. He also found that it produced the correct result in over 95% of test cases, although we should note that his test data comprised artificial spelling errors rather than users misspellings.

While Bickel’s results are impressive, they were achieved for a relatively small database. Bickel notes several possible improvements that could be made to the algorithm.

One effect that Bickel notes is that the more letters that a candidate name has, the more likely it is to match any offered name. A possible enhancement noted by Bickel is that some restriction should be placed on word lengths, but he does not quantify the degree of improvement achieved by doing this. I report on some tests on this in Section 6.3.5.

Since a lot of research shows that most spelling errors are single errors, ignoring the position of characters allows more freedom in matching than strictly necessary. Bickel suggests that extra values can be added to the likeness value based on positional matching. Unfortunately he does
not say how he does this. I report on some experiments to enhance the Bickel algorithm with positional matching in Section 6.3.5.

The weights that Bickel assigned to each character were derived by analysing the relative frequency of letters in his test database of about 1000 names. The formula used is based on first-order information theory, and is:

\[
\text{weight(letter)} = -\log_2(\text{relativeFrequency(letter)})
\]

He observed that these weights matched closely those for the English language. I have tested the affect of using weights tuned for the target database, and report on this in Section 6.3.5.

Bickel does not attempt to justify his decision that multiple matches of a letter should only count once. Intuitively, this seems to be ignoring useful information. My suspicion is that he did this for reasons of computational efficiency, to allow him to use a masking technique to calculate the likeness value. I assess the difference this makes in Section 6.3.5.

Finally, calculating likeness values is computationally more expensive than Soundex and Metaphone for two reasons. First, Soundex and Metaphone can pre-compute and index their keys. Second, the short string comparisons used by Soundex and Metaphone are intrinsically simpler and faster than the Bickel likeness comparisons. The danger exists that, even if the algorithm produces better matches than Soundex or Metaphone, the algorithm will be too slow for medium and large size databases. These performance issues are addressed in Section 6.3.7.

### 6.3 Some Approximate Matching Experiments

In this section I describe some experiments I undertook to determine the effectiveness of the approximate matching algorithms described in the previous section. Much of the work is assessing various tuning options for the algorithms; we noted some of these options in the previous section. Having understood how we can tune the various algorithms, I then make a broad comparison of the four methods of approximate matching.

The structure of this section is as follows. In Section 6.3.1, I describe the experimental methodology. The experiments were designed to show both the precision and the recall of the various algorithms and their options. In Sections 6.3.2 to 6.3.5, I look at the precision and recall of the four algorithms, concentrating on how the algorithms may be tuned. In Section 6.3.6, I test the performance of the algorithms purely for handling typographic errors. In Section 6.3.7, I discuss the speed with which the algorithms are evaluated, since an algorithm that offers a high recall rate and good precision may be impractical because it is too slow. I conclude the chapter by summarising and comparing the broad characteristics of the four algorithms.

#### 6.3.1 The Experimental Methodology

The experiments described in this section are an extension of the experiments described in Section 5.5; they are based on real user queries of the UCL X.500 database.
6.3. SOME APPROXIMATE MATCHING EXPERIMENTS

In order to avoid the issues of mis-matches of form mentioned in the introduction of this chapter, the experiments are all based on surname matching only.

The experiments were designed to show both the precision and the recall of the approximate matching algorithms.

Assessing Precision

The precision of the algorithms is shown by trying to approximately match the 209 surname-only, no department name specified queries for entries in the UCL database, and averaging the number of results as a per query figure. It is impossible to give an exact figure for the optimum number of entries that an algorithm should return. Some queries, for names such as "Davies", correctly match a substantial number of entries in the database; "Davies" is a popular name. A few queries have no corresponding entry in the database. Some queries are misspelled, but sometimes in such a way that it is hard to be sure which entry or entries the query should match.

However, it is possible to make a reasonable estimate of the number of entries that an ideal approximate matching algorithm would return on average by considering:

- The number of results returned by exact matching for the correctly spelled queries;
- An estimated number of entries that should be found for misspelled queries, assuming that only plausible candidates are returned. These estimates are based on my judgement of reasonableness, as well as the view of some referees: the refereeing process is described in the following section.

Doing this, I would conclude that an optimum approximate matching algorithm should return about 4.40 entries per query given the UCL database and query set. We will see that, in practice, algorithms returned on average between seven and eighty entries per query.

Assessing Recall

The effectiveness of each algorithm at finding the correct results with misspelled input, the recall of the algorithm, is much harder to assess. Should, for example, user input of "Coomb" match "Coombs" or "Combe"? A problem I faced with my experiments was that if I decided what constituted a reasonable match, I could bias the results by acknowledging some types of spelling mistakes, but failing to spot others.

I have attempted to avoid this problem as much as possible by allowing others to decide what were reasonable matches. The procedure I used was as follows:

1. I selected 31 candidate misspelled surnames from the UCL queries. In most cases I believed it was obvious which entry the user was trying to find; in some cases I believed that a query was plausibly intended to match an entry in the database, but was not sure which was the likeliest candidate.
2. I constructed lists of plausible candidate names by merging the entries matched by the various matching algorithms. There was one exception. For the query “Pshanks”, I added a further candidate name of “Shanks”; none of the matching algorithms found this match as all algorithms insisted on matching on the first character.

3. I showed seven colleagues the message reproduced in Appendix F.1. My colleagues were invited to act as referees, and to select plausible candidate entries from the lists of merged names. My colleagues could alternatively select a single name from the list for each query, several names if they thought they were equally plausible, or no name at all if they did not believe that any of the candidate names were plausible matches.

I merged the results of my colleagues’ individual judgements to give a weighted overall judgement. I did this as follows. If a candidate name was identified by every referee, it scored a maximum of seven points (i.e, one point per referee). If a candidate name was identified by a single referee as being plausible, it scored one point. The majority of the candidate names were not considered plausible by any of the referees and thus scored no points. For example the query “Assimakopoylos” was believed to be a reasonable match for “Asimacopoulos” by all seven referees, while five of them believed that “Assimakopoulou” was also a plausible match. 25 other candidate names were considered implausible by all the referees.

Each algorithm was then assessed in two ways. First, I tallied the points for each algorithm according to how many of the plausible names were found by that algorithm. Thus, from above, if an algorithm matched both Asimacopoulos and Assimakopoulou for input of Assimakopoylos, it scored twelve points (seven plus five) for that input. The process was repeated for all 31 misspelled input names. An algorithm with good recall would have a high overall tally, one with poor recall a low tally. The maximum possible score is 255 points. In the tables of results that follow, this figure is given in a column headed “Recall total”.

The second type of assessment was to select the candidate name, for each query, chosen by the most referees. Thus, Asimacopoulos with seven votes is regarded as the preferred match for the query Assimakopoylos, beating the other candidate name of Assimakopoulou which had only five votes. A good algorithm would return a large number of the preferred matches. If the two most popular candidate names received the same number of votes, each of these names was given a half point as a joint favourite. The maximum points for any algorithm is 31 points. In the tables of results that follow, this figure is given in a column headed “Recall favourites”. The referees’ preferred matches are shown in Appendix F.2.

In practice, the two methods of assessing recall correlate very closely.

6.3.2 Damerau

An implementation of an algorithm that finds Damerau spelling mistakes, but no other misspellings, provides a useful benchmark for assessing the other algorithms. Such an algorithm
cannot be tuned; correctly spelled input and those with Damerau errors match, all others fail. I based my implementation of a Damerau matching algorithm on the description given by Pollock and Zamora in [PZ84], who rightly claim that their method is delightfully simple.

<table>
<thead>
<tr>
<th>Ave no. matches</th>
<th>Recall total</th>
<th>Recall favourites</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.02</td>
<td>139</td>
<td>19</td>
</tr>
<tr>
<td>Max</td>
<td>255</td>
<td>31</td>
</tr>
</tbody>
</table>

Table 6.5: Damerau matching: precision and recall

The recall rate of the referee-determined correct matches is as expected from our analysis of the query data in Chapter 3, with just over 60% of the misspelled queries being Damerau errors. We will see that other algorithms, not restricted to finding Damerau misspellings, can achieve better recall rates than this. The important finding from the experiment is that the average number of entries returned is 7.02 per query. As we look at other algorithms, we will see that the precision of a Damerau matching algorithm is on a par with the best of any of the other algorithms.

6.3.3 Soundex

We saw in Chapter 5 that Quipu's implementation of Soundex tends to be a rather noisy algorithm, often producing a large number of matches. The emphasis of the experiments with Soundex was thus to find ways of reducing the number of spurious entries matched, without reducing the number of correct matches.

I experimented with five variables of the Soundex algorithm. These variables are as follows, where the abbreviations in brackets are used as headings in Table 6.6 below:

**Key length (KL):** The Soundex specification is for keys of up to four characters in length: one alphabetic character following by three numeric codes. However, this ignores information in long words. I have tested the effect of various key lengths. (UL in Table 6.6 stand for unlimited length).

**Prefix matching (PM):** If prefix matching is enabled, a match is made if the Soundex key for the user input matches the leading substring (or prefix) of a candidate entry's Soundex key: e.g., the key 'A' matches the prefix of key 'A123'. This could be used for matching abbreviated input.

**Vowels (V):** The Soundex specification is to ignore vowels and soft consonants. As we saw in Section 6.2.3 in the case of Fawehinmi matching Fan, ignoring vowels and soft consonants ignores most of the name, and allows some absurd matches. I have experimented with treating vowels and soft consonants as significant.
First two characters correct (F2): The Soundex specification requires the first character to match exactly, and the rest of the name to be transformed into a numeric code. I have experimented with a more stringent specification, requiring both the first two characters to match exactly.

Name lengths (NL): The Soundex specification takes no account of word lengths when matching, and thus allows obviously wrong matches such as Fan and Fawehinmi. I have experimented with only allowing matches for names that are similar in length. My definitions of similar in length were arbitrarily, but I hope reasonably, chosen as follows:

- If the query is fewer than six characters, the target word must be within one character of the same length.
- If the query is six or more characters, the target word must be within two characters of the same length.

<table>
<thead>
<tr>
<th>Code letter</th>
<th>KL</th>
<th>PM</th>
<th>V</th>
<th>F2</th>
<th>NL</th>
<th>Ave no. matches</th>
<th>Recall total</th>
<th>Recall favourites</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>UL</td>
<td>Y</td>
<td>V</td>
<td>F2</td>
<td>NL</td>
<td>77.84</td>
<td>171</td>
<td>22.5</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td>78.81</td>
<td>196</td>
<td>24.5</td>
</tr>
<tr>
<td>C</td>
<td>UL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>18.56</td>
<td>169</td>
<td>22.5</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20.41</td>
<td>194</td>
<td>24.5</td>
</tr>
<tr>
<td>E</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>42.99</td>
<td>199</td>
<td>24.5</td>
</tr>
<tr>
<td>F</td>
<td>UL</td>
<td>Y</td>
<td>V</td>
<td></td>
<td></td>
<td>7.73</td>
<td>139</td>
<td>18.5</td>
</tr>
<tr>
<td>G</td>
<td>UL</td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
<td>16.66</td>
<td>168</td>
<td>22.5</td>
</tr>
<tr>
<td>H</td>
<td>UL</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td>55.14</td>
<td>148</td>
<td>19.5</td>
</tr>
<tr>
<td>I</td>
<td>UL</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td>11.61</td>
<td>147</td>
<td>19.5</td>
</tr>
<tr>
<td>Max</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>255</td>
<td>31.0</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.6: Soundex matching: recall and precision with various options

A number of experiments were made with various combinations of settings of the variables described. Each experiment is identified by a code letter in Table 6.6. If a variable was enabled for a particular experiment, the appropriate column contains the letter “Y” (for Yes).

The default settings for Soundex matching in Quipu are those in code A in Table 6.6. The problem with these settings is that it produces a very high average (mean) number of matches. However we can see from Table 6.6 that there are a number of ways of considerably improving the precision of Soundex, with relatively modest impact on the rate of recall.

Comparing the algorithm settings with codes A and C, or comparing B with D (D has the standard Soundex settings), shows the effect of dis-allowing prefix matching. For both comparisons, the average number of matches is reduced by almost 75% if prefix matching is not allowed,
with little reduction of the number of misspelled queries correctly matched. The cost of allowing prefix matching as part of an approximate matching strategy seems very heavy, given its limited advantages. It can only help with misspelled abbreviated input as leading substring matching can be used for correctly spelled abbreviated input.

Another way of reducing the number of entries matched is to insist on queries and matched entries being approximately the same length. We can see this by comparing options C and G; a further 10% of spurious matches are eliminated.

An alternative way of improving the precision is to insist on the first two characters matching rather than merely the leading characters. If we compare options C and F, we see that this method is extremely effective at reducing the average number of matched entries; however it also reduces the number of misspelled queries that are correctly matched.

A further option to consider is counting vowels and soft consonants as significant, by allocating them their own Soundex code. If we compare option A with H, or option C with I, we see that this change reduces the average number of matches by about a third, but it reduces the number of misspelled queries found correctly.

I also examined the impact of using Soundex keys of differing lengths. I compared unlimited length keys, a key length of four characters (following the specification), and a key length of three characters. These different settings can be seen in options C, D and E respectively. The differences between using unlimited length keys and keys of four characters are quite small. The unlimited length keys (they will only be longer than standard keys for long names) produce slightly fewer matches on average, but also find slightly fewer misspelled entries. A key length of three produces over twice the average number of matches, compared with a key length of four, and yet found no extra matches. A key length of four characters seems a good compromise.

Conclusions on Soundex

First, Soundex’s recall, with the standard settings, appears to be better than the Damerau algorithm (24.5 matches compared with 19). However, the differences are not statistically significant, using McNemar’s small sample test, at the 5% level.

Second, the disadvantages of prefix matching outweigh the advantages. Prefix matching produced almost four times as many matches, but no additional correct matches. Its chief useful function is to match initials in user input with names in the directory, but it would be better to provide this function directly rather than as a side effect of generalised prefix matching.

Third, it is possible to reduce the number of false matches by insisting either that the first two characters of word and directory name must match, or by treating vowels and soft consonants as significant characters. However doing this reduces the recall of the algorithm to the same as Damerau matching, while the precision is slightly worse than Damerau.
6.3.4 Metaphone

The experiments with Metaphone were similar to those with Soundex, although the emphasis was somewhat different. The reason for this, demonstrated by the results in Table 6.7, is that while Metaphone produces fewer spurious matches, it also tends to find fewer correct matches, and so there is less emphasis on tightening up the algorithm.

I experimented with three variables of the Metaphone algorithm. These variables are as follows, where the abbreviations in brackets are used as headings in Table 6.7 below:

Key length (KL): The University of Michigan Metaphone implementation uses keys of up to four characters in length. As for Soundex, I have tested the effect of using keys of different lengths.

Prefix matching (PM): This is used in the same way as with Soundex matching.

Name lengths (NL): This is the same restriction as described for, and applied to, the Soundex algorithm.

<table>
<thead>
<tr>
<th>Code letter</th>
<th>KL</th>
<th>PM</th>
<th>NL</th>
<th>Ave no. matches</th>
<th>Recall total</th>
<th>Recall favourites</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td>Y</td>
<td></td>
<td>55.65</td>
<td>152</td>
<td>18</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
<td></td>
<td>Y</td>
<td>11.85</td>
<td>151</td>
<td>18</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td></td>
<td>Y</td>
<td>10.50</td>
<td>150</td>
<td>18</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>Y</td>
<td>Y</td>
<td>20.21</td>
<td>151</td>
<td>18</td>
</tr>
<tr>
<td>E</td>
<td>3</td>
<td></td>
<td></td>
<td>25.36</td>
<td>162</td>
<td>19</td>
</tr>
<tr>
<td>F</td>
<td>10</td>
<td></td>
<td></td>
<td>10.85</td>
<td>126</td>
<td>16</td>
</tr>
<tr>
<td>Max</td>
<td></td>
<td></td>
<td></td>
<td>255</td>
<td>31</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.7: Metaphone matching: recall and precision with various options

The default settings of the University of Michigan Metaphone algorithm are those with code letter A: a key length of 4 with prefix matching allowed. Unfortunately, this setting shows Metaphone at its worst. The average number of matches found by Metaphone is 30% less than found by Soundex when it is used with the same options, but it finds over 25% fewer correct matches.

Turning off prefix matching (code B) cuts the average number of entries matched by four fifths, with almost no effect on the recall of correct entries. The average number of entries returned is now comparable with the better Soundex options, with a similar recall rate too.

If prefix matching is not used, and there seems little justification for using it, restricting matches to those similar in length further reduces the average number of matches (code C). The reduction of about 10% is similar to that for Soundex.
6.3. SOME APPROXIMATE MATCHING EXPERIMENTS

Increasing the key length from four to ten characters (codes B and F) reduced the average number of matches by about 10%, but also reduced the recall rate.

It is easy to find settings for Metaphone that produce a low average number of matches, but the recall rate is noticeably worse than that for Soundex. Using McNemar’s test to compare Metaphone with Soundex using their standard settings, we get a $P$ value of 0.016, indicating that Soundex is significantly better (at the 5% level) at finding correct matches.

We will now examine Bickel’s algorithm to see if we can combine Soundex’s rates of recall with Metaphone’s precision.

6.3.5 Bickel

Whereas the Soundex and Metaphone algorithms evaluate a candidate match as either TRUE or FALSE, the Bickel algorithm calculates a likeness value which allows candidate entries to be ranked in order of most likely to least likely match.

Stemming from this difference in approach, the experiments with the Bickel algorithm are a little different. First, I have assessed the algorithm using the misspelled queries to see whether the algorithm placed the referee-decided entries at the top, or near the top, of the ranked list. Second, I have used a slightly modified version of the Bickel algorithm for tests similar to those in the previous two sections. The modification is to fix a threshold likeness value, as a proportion of the maximum likeness. Thus if the maximum likeness value is 40, and the chosen threshold is 0.80, then any entry with a likeness of more than 32 (40 * 0.80) is deemed to match. This adaptation of the Bickel algorithm to return TRUE or FALSE for any candidate match meant that I could test the algorithm within my test DSA with relatively few code changes.

Bickel Ranking

The first test was to see how well the algorithm performed at finding the referee adjudged entries for the 31 cases of misspelled input. A good result would be for the algorithm to place the referee determined entry at the top, or near the top, of the ranked list for every misspelled query.

The algorithm ranked the referee’s choice top, or equal top, 22 times out of 31. Six more queries were ranked within the top five. The worst ranking was 55th, apart from the "pshanks" query which was unranked as its first letter did not match the first letter of the target entry "Shanks". The algorithm is clearly effective.

These results can be improved further by insisting that the matched name must be of similar length to the query input. I used the same length similarity measure as for Soundex and Metaphone (described in Section 6.3.3). 25 queries were now ranked top or top equal, with the worst ranking now 11th. These are, in the author’s view, very good results. It may be possible to improve these results even further by using some of the techniques described in the next section.
Bickel with Threshold Likeness Values

I experimented with six variables of the Bickel algorithm. These variables are as follows, where the abbreviations in brackets are used as headings in Table 6.8 below:

**Threshold value (TV):** I experimented with various threshold values between 0.75 and 0.90 of the maximum likeness value. Note that an entry is only considered to match if the likeness value *exceeds* the threshold value.

**Name lengths (NL):** This is the same option as tried with both the Soundex and Metaphone algorithms.

**First two characters correct (F2):** As for Soundex: see Section 6.2.3.

**All characters weighted (AC):** The Bickel specification counts multiple instances of a letter once when producing the likeness value. This means, for example, that Baker and Barker both have the maximum likeness value when matched against Barker, since both candidate names have at least one instance of all the letters in Barker. I have experimented with having all letters count when calculating the likeness values, as to not do so seems to ignore useful information.

**Letter position (LP):** The Bickel specification says that the position of letters should be ignored when calculating likeness values. One example from my experiments showed that McGruther was considered a good match for Murtough; it has all the necessary letters. However, the letters G and T are in completely the wrong position, and it is arguable that it makes little sense to count them when calculating a likeness value. I have experimented with only including letters in a likeness value for a target word if they are within one character position of the same letter in the user’s query.

**Modified weights (MW):** The weights Bickel used for the letters were derived from letter frequencies in his target database. I have experimented with weights tuned to the UCL database.

My initial experiments with the Bickel algorithm were to assess a suitable threshold value. A threshold value of 0.75 (code A in Table 6.8) produced roughly the same average number of matches as the worst instances of the Soundex and Metaphone algorithms. However, the recall with this threshold was better than that achieved with any settings of either Soundex or Metaphone. I tried other thresholds: see codes B, C and D. A threshold of 0.90 is clearly too high; this setting has relatively poor recall and the average number of matched entries does not compare well with the best achieved by Soundex and Metaphone. The threshold values of 0.80 and 0.85 offer above average recall (compared with the various Soundex and Metaphone options), but with a high number of average matches. We will now see if we can use some of the variables of the Bickel algorithm to improve the precision while retaining a good rate of recall.
The most important modification to the Bickel algorithm is to only allow matches of similar length. If we compare code A with code E, or C with F, we see that insisting that the lengths of the query and the target entry name are similar has no effect on the recall rate, but halves the average number of entries matched.

Another effective way of reducing the number of entries matched is to insist that the first two characters of the query and the target name match. Doing this with a threshold value of 0.75 (code G in Table 6.8) more than halved the average number of entries returned, although a few correct matches were now missed. Using this option in conjunction with the length restriction for a threshold of 0.75 (code H) reduced the average number of matches to just over a quarter of the matches found without these options.

We noted in Section 6.2.3 that Bickel's algorithm counts multiple instances of the same letter only once when calculating a likeness value. This means, for example, that the queries "Ban" and "Banan" both match the target word "Banana" equally well, since no allowance is made for the repetition of letters. I experimented with calculating the weighting based on all instances of the letters in a name. The effect of this is shown by comparing the results for codes A and I. The
average number of matches is reduced by about a third, with minimal impact on the recall of the algorithm. This technique needs to be used in combination with some of the techniques described above. If used with the length restriction (as in code J) it reduces the average number of matches to just over a quarter the number found without these options, with no further reduction in the rate of recall. Codes K and L show other combinations of the thresholds and options discussed so far.

Another way of restricting the number of matches made is to insist that for letters to match, they must also be in approximately the right place in the word. Codes M and N show the effect of this restriction for threshold values of 0.75 and 0.80 respectively. This insistence on approximate positional matching reduces the average number of matches to about a third of those found without the restriction, while it has a small effect on the number of correct matches: compare code A with M, and B with N. Using positional matching and a length restriction (as in code O) further reduces the average number of matches to less than a fifth of those found without these options.

All the experiments described so far in this section use likeness values based on the weights in Bickel's paper. I tried some experiments with weights tuned to the letter frequencies of the UCL database; this is a practical proposition for a DSA which could calculate these weights as a house-keeping function or even at DSA start-up time. The UCL weights are broadly similar to Bickel's weights, although only 8 of the 26 letters have the same weight; both sets of weights are shown in Table 6.9.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
<th>L</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bickel</td>
<td>3</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>8</td>
<td>7</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>UCL</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>7</td>
<td>5</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>O</th>
<th>P</th>
<th>Q</th>
<th>R</th>
<th>S</th>
<th>T</th>
<th>U</th>
<th>V</th>
<th>W</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bickel</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>8</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>UCL</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>10</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>6</td>
<td>10</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 6.9: The weights used by Bickel and those calculated for the UCL database

The performance of the algorithm with the letter weights tuned to the database was very similar to that with the original weights: e.g., compare codes A and P.

For the sake of curiosity as much as for science I also tried experiments with the letter weights used in the game of Scrabble: the results were markedly poorer!

Conclusions on Bickel

We have considered several ways of using and modifying the Bickel algorithm. A very attractive option is to use the algorithm as originally designed as a way of ranking candidate matches. If we do this and modify the algorithm to insist that matched names are similar in length to the query
name, the algorithm ranks 25 of the correct matches top or equal top, and all except "Shanks" of the correct names were within the top 11. Compare this with the fact that Soundex at best found 24.5 of the correct matches.

I have also demonstrated a way of using the Bickel algorithm to return TRUE or FALSE using threshold values; this allows the algorithm to be compared directly with the other algorithms tested here. The raw algorithm shows good recall rates, finding 26 correct matches with a threshold value of 0.75, but the precision is poor. However, there are several ways of improving the precision of the algorithm. Two methods are particularly recommended. First, candidate matches should be restricted to those similar in length to the query. Second, likeness values should include the weights for every instance of a letter, rather than counting multiple occurrences of a letter just once. These settings work best with a threshold value of 0.75 or 0.80; higher threshold values exclude too many correct matches. These settings offer a balance of recall and precision that appears to be better than Soundex, although the small sample size makes it impossible to verify this using statistical significance techniques. However, Bickel technique J (from Table 6.8) has superior recall compared with the standard Metaphone algorithm and the Damerau algorithm at 5% significance, with $P$ values of 0.039 and 0.016 respectively.

Although it makes sense to tune the letter weights to the database in question, in practice Bickel's weights worked as well as the data-specific ones.

### 6.3.6 Approximate Matching Algorithms and Damerau Misspellings

As typographical mistakes are the predominant type of misspelled input, I thought it would be interesting to test the effectiveness of Soundex, Metaphone and Bickel’s algorithm at dealing specifically with these errors. As with the earlier experiments described in Section 6.2.3, I did two tests of the Bickel algorithm: one with Bickel using threshold values, and a second using the algorithm returning results in ranked order.

We noted earlier that each algorithm can be tuned by a number of parameters. For this experiment I chose:

- **Soundex**: the standard settings - i.e, a key length of four and no modifications to the normal algorithm;
- **Metaphone**: as for Soundex;
- **Bickel**: Bickel algorithm with the name length restriction and a threshold value of 0.82. These settings were chosen because I found, while doing the work described in Section 6.3.5, that they returned a very similar number of results to the Soundex algorithm.

The experiment was constructed as follows. Each name in the UCL database (16174 names in total) was randomly modified according to the four error classes described in Section 6.2.1. Since letter insertions and substitutions are heavily influenced by keyboard adjacencies, I did two
transformations for each word for both these types of error: once with a completely arbitrary letter and once with a letter that was adjacent on the keyboard to the intended letter. My definition of adjacent letters is the letters immediately to the left and right of the intended letter for substitution errors. For insertion errors I also include the letter itself, as many insertion errors are double strikes of a correct letter. Non-alphabetic adjacent characters are excluded: such characters could easily be detected by input routines; furthermore, to include them would complicate interpretation of the results.

All the algorithms expect the first character to be correct, or at least sound correct in the case of Metaphone. As we noted earlier first character errors are indeed relatively infrequent. For this reason, I chose not to produce errors in the first character position. Without this restriction, the results for all the algorithms would have been considerably worse.

Each modified name was approximately matched against all the names in the database. For each name, for each algorithm, we note whether the correct entry is found and the total number of matches returned.

The results of the experiment are shown in Table 6.10. For each algorithm and error type, I show the percentage of the 16174 queries where the correct entry was found and the average number of results returned. A good algorithm will have a high percentage of correct matches (a good recall rate) and a low average number of results (good precision).

<table>
<thead>
<tr>
<th>Error Type</th>
<th>Soundex</th>
<th>Metaphone</th>
<th>Bickel</th>
<th>Damerau</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>% correct</td>
<td>ave.</td>
<td>% correct</td>
<td>ave.</td>
</tr>
<tr>
<td></td>
<td>matches found</td>
<td>matches</td>
<td>matches found</td>
<td>matches</td>
</tr>
<tr>
<td>Omission</td>
<td>62.4</td>
<td>24.49</td>
<td>53.9</td>
<td>14.82</td>
</tr>
<tr>
<td>Transp.</td>
<td>80.6</td>
<td>21.97</td>
<td>65.9</td>
<td>11.73</td>
</tr>
<tr>
<td>Insertion</td>
<td>62.1</td>
<td>16.11</td>
<td>47.6</td>
<td>7.25</td>
</tr>
<tr>
<td>Ins. Adj</td>
<td>77.5</td>
<td>19.37</td>
<td>57.4</td>
<td>8.53</td>
</tr>
<tr>
<td>Subst.</td>
<td>45.3</td>
<td>17.49</td>
<td>35.6</td>
<td>8.37</td>
</tr>
<tr>
<td>Subst. Adj</td>
<td>52.6</td>
<td>17.46</td>
<td>34.0</td>
<td>6.49</td>
</tr>
</tbody>
</table>

Table 6.10: Effectiveness of approximate matching algorithms with common typing errors

As a benchmark I also tested the precision of the Damerau matching algorithm; the recall rate is by definition 100%. As we found with the experiments reported in Section 6.3.2, its precision is impressive and it returned fewer entries than the other algorithms.

The Metaphone algorithm was nearly as economical as the Damerau algorithm, but both Soundex and Bickel returned twice the number of results.

Turning to their recall, we see that Metaphone was worse than any other algorithm for all error types. Other than the Damerau algorithm, the Bickel algorithm was the best by some way in four of the six categories (strongly significant with McNemar test $P$ values of less than 0.001), and marginally worse than Soundex for the other two categories. There was some consistency in
which error types the algorithms handled well and not so well: Soundex, Metaphone and Bickel were all least successful with substitution errors, and most successful with transposition errors.

I also conducted an experiment with the Bickel algorithm, using it as it was originally designed to return results in ranked order. Ideally the correct result would always be returned at the top of the list of ranked possibilities. The results of the experiment are shown in Table 6.11.

<table>
<thead>
<tr>
<th>Error type</th>
<th>Correct entry within top</th>
<th>1</th>
<th>3</th>
<th>5</th>
<th>10</th>
<th>50</th>
<th>100</th>
<th>Worst ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Omission</td>
<td></td>
<td>100.0</td>
<td>100.0</td>
<td>100.0</td>
<td>100.0</td>
<td>100.0</td>
<td>100.0</td>
<td>1</td>
</tr>
<tr>
<td>Transposition</td>
<td></td>
<td>100.0</td>
<td>100.0</td>
<td>100.0</td>
<td>100.0</td>
<td>100.0</td>
<td>100.0</td>
<td>1</td>
</tr>
<tr>
<td>Insertion</td>
<td></td>
<td>69.6</td>
<td>82.5</td>
<td>87.3</td>
<td>92.4</td>
<td>99.0</td>
<td>99.8</td>
<td>382</td>
</tr>
<tr>
<td>Insertion Adj</td>
<td></td>
<td>78.7</td>
<td>87.6</td>
<td>90.9</td>
<td>94.0</td>
<td>99.3</td>
<td>99.8</td>
<td>193</td>
</tr>
<tr>
<td>Substitution</td>
<td></td>
<td>65.6</td>
<td>79.0</td>
<td>84.2</td>
<td>90.4</td>
<td>98.7</td>
<td>99.7</td>
<td>280</td>
</tr>
<tr>
<td>Substitution Adj</td>
<td></td>
<td>59.6</td>
<td>74.8</td>
<td>81.0</td>
<td>87.6</td>
<td>98.1</td>
<td>99.7</td>
<td>487</td>
</tr>
</tbody>
</table>

Table 6.11: Effectiveness of Bickel algorithm with ranked results

The results are impressive. The correct result is ranked first, or equal first, for almost two-thirds of the errors for five of the six error types. The correct result is within the top ten ranked results for over 90% of errors for five of the six error types. These results persuade me that, used to give ranked results, the Bickel algorithm offers substantial advantages over the Soundex and Metaphone algorithms for dealing with typographic spelling mistakes.

### 6.3.7 The Speed of the Algorithms

We noted earlier in this chapter that some algorithms are likely to be more computationally efficient than others. Algorithms such as Soundex and Metaphone, where the names/words are transformed into keys before comparisons are made, can be very fast as the keys for the candidate matches can be pre-computed and indexed. Alternatively an algorithm such as that proposed by Bickel offers less scope for pre-computation and indexing: more calculation has to be done as the comparison between the input and the candidate words/names are made. Furthermore, algorithms differ in their basic computation complexity.

The intention of this section is to give an indication of the relative speeds of the algorithms discussed in this chapter. The reader should note that the speeds given are merely intended as a rough guide. The speeds could be improved in two ways.

1. The author has no doubt that the speeds given could be considerably improved with more efficient coding of the algorithms.

2. Some restrictions on the algorithms, such as insisting that the first two characters match rather than just the first one, allow large speed-ups.
The experiments described in this section were conducted using a specially written test harness. The candidate names were the 16174 surnames in the UCL database. These were read into memory and any indexes were built. The 209 surname-only queries were used as test data. The test data was also read into memory at program start up. The intention was to focus as much as possible on the evaluation time of the algorithms, and exclude other overheads such as I/O.

As a benchmark, I have compared the four approximate matching algorithms with exact matching and any-substring matching.

The "no. of comparisons" column in Tables 6.12 to 6.14 shows how many comparisons of keys had to be made before the search was completed. As more indexing was used, fewer comparisons were necessary.

The speeds given in the tables are in milliseconds and are the average time taken for a single input name to be matched against the candidates: the time given is the lowest achieved in five consecutive tests. The tests were done on a SUN SPARCstation IPC.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>No. of comparisons</th>
<th>Time per query in milliseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any substring</td>
<td>16174</td>
<td>105.39</td>
</tr>
<tr>
<td>Exact</td>
<td>484</td>
<td>1.56</td>
</tr>
<tr>
<td>Soundex</td>
<td>440</td>
<td>2.02</td>
</tr>
<tr>
<td>Metaphone</td>
<td>547</td>
<td>1.71</td>
</tr>
<tr>
<td>Bickel</td>
<td>885</td>
<td>8.63</td>
</tr>
<tr>
<td>Damerau</td>
<td>885</td>
<td>8.73</td>
</tr>
</tbody>
</table>

Table 6.12: Speed comparison of algorithms with minimal indexing

Table 6.12 shows the speed of the algorithms with minimal or no indexing. The minimal indexing consisted of sorting the candidate names or keys, and creating an index pointing to the first instance of a name/key beginning with each of the 26 letters. Note that indexing is not straightforward for any-substring matching (although Howes describes one way of doing it in [How95b]) and this type of matching was unindexed.

As one would expect the unindexed any-substring matching was much slower than any other type of matching, but even these matches took on average only just over a tenth of a second.

At the other end of the scale, exact matching was over sixty times faster than any-substring matching, with much of this benefit stemming from being able to index the search. Soundex and Metaphone matching were almost as fast as exact matching, and are clearly highly appropriate algorithms if speed is of the essence.

Between the two extremes, the Bickel and Damerau algorithms were five times slower than the fastest algorithms, but nevertheless on average matched individual entries in less than one hundredth of a second.
6.3. SOME APPROXIMATE MATCHING EXPERIMENTS

However the figures in Table 6.12 understate the performance advantages of Soundex and Metaphone, which in practice could be backed up by something much more efficient than the crude indexing used in the experiments just described. In order to get a more realistic assessment of the speed advantages offered by Soundex and Metaphone, I spoke to Tim Howes, the author of Quipu's indexing code. Following his advice I did some further experiments using indexing techniques that reduced the number of key comparisons to a level similar to that used in the Quipu system. The results of these experiments are shown in Table 6.13.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>No. of comparisons</th>
<th>Time per query in milliseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Soundex</td>
<td>24</td>
<td>0.15</td>
</tr>
<tr>
<td>Metaphone</td>
<td>18</td>
<td>0.14</td>
</tr>
</tbody>
</table>

Table 6.13: Speed of Soundex and Metaphone with enhanced indexing

The enhanced indexing produces more than ten-fold speed-ups. The average match time is now a fraction of a millisecond, even on such modest equipment as a SUN IPC.

Unfortunately it is not so easy to gain such dramatic speed-ups with the Bickel and Damerau algorithms. The effect of some configuration options of the Bickel algorithm are shown in Table 6.14. The abbreviations in the table are as given in Section 6.3.5, namely:

TV: Threshold value;

NL: Name lengths of matching values must be within one or two characters;

F2: First two characters must match;

AC: Repeated instances of a letter all count in calculating the likeness values.

<table>
<thead>
<tr>
<th>TV</th>
<th>NL</th>
<th>F2</th>
<th>AC</th>
<th>No. of comparisons</th>
<th>Time per query in milliseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>75</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>885</td>
<td>8.63</td>
</tr>
<tr>
<td>75</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>162</td>
<td>1.59</td>
</tr>
<tr>
<td>85</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>885</td>
<td>9.95</td>
</tr>
<tr>
<td>85</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>885</td>
<td>11.13</td>
</tr>
</tbody>
</table>

Table 6.14: Speed of various configurations of the Bickel algorithm

The option of insisting that the first two characters must match increases the indexing potential and leads to more than a five-fold increase in speed. The cost, as we noted in Section 6.3.5,
is that the recall rate of the algorithm is now slightly worse. Other options, which were designed to increase the precision of Bickel, increased the execution time by up to 30%.

Other options remain for improving the speed of all these algorithms. For example, on average 43% of candidate names are within one character in length of the query name. Indexing techniques that eliminated much longer or shorter names from the search space could potentially double search speed.

Finally I found that the absolute speed of the algorithms tested here could be improved by a factor of three or four by running the experiments on a SPARCstation 5.

Conclusions on the Speed of the Algorithms

What conclusions can we draw from these experiments? First, if matching speed is the highest priority, then algorithms such as Soundex and Metaphone are the best choice, as these can be almost as fast as exact matching if the keys are pre-computed.

Second, the Bickel and Damerau algorithms are appreciably slower than Soundex and Metaphone: they are more complicated to compute, but the main problem is that searches cannot be indexed to the same degree.

Third, even the slowest variant of the Bickel algorithm took only one hundredth of a second on average to match entries in a 16000 entry database on a SPARC IPC.

Fourth, the slowest variant of the Bickel algorithm was almost ten times faster than the any-substring algorithm used with reasonable success in the Quipu directory system.

Fifth, these relative differences in speed are diminished as we consider all the other factors involved in a directory search operation, such as decoding the query, encoding the results, and passing the query and results across the network. We present some evidence on response times in Appendix G.

Sixth, we should consider a philosophical point. I imagine that most users would expect an approximate match to take longer than an exact match, or at least find it reasonable if that was the case. I would thus argue that speed differences should not be over-emphasised when considering the choice of approximate algorithm.

6.4 Conclusions on Approximate Matching

The motivation for the work described in this chapter was to assess whether there were viable alternatives to Soundex, as that algorithm appeared to return too many incorrect results. However, the problem that we observed in Chapter 5 of Soundex returning large result sets is largely a problem of Quipu using prefix matching. The option is usually enabled as it helps match initials in user input to directory names. However, it also has a disastrous effect on matching names; it makes the algorithm return roughly four times as many results, with hardly any improvement in finding correct results.
6.4. CONCLUSIONS ON APPROXIMATE MATCHING

Although Soundex is much better than first feared, it is still worth considering whether any of the many alternative algorithms are better suited to the task of approximate matching.

We should start by considering the nature of the problem. There are two aspects: misspelled input, and input which is in a different form to that the names in the directory. Let's consider misspellings first. We noted in Chapter 3 that 80% of misspellings for country, organisation and department names were typographical mistakes. People's names were somewhat different. First, the misspelling rate was much higher, at about 7% compared with 3% for the other categories. Second, the proportion of typographical errors was lower at about 60%; it seems reasonable to assume that people find it harder to spell human names - there are more cognitive errors.

It follows from these figures that an algorithm that deals solely with single typographical errors (Damerau errors) would correct about 80% of country, organisation and department name misspellings. The results in Chapter 5 show that Soundex achieved about 70% correction rates. Furthermore, the Damerau algorithm is more precise than Soundex, returning far fewer spurious matches.

However, an algorithm that corrects only Damerau errors would perform worse than Soundex for human names: we need to consider other algorithms. One alternative that has been tried by some DSA developers is Metaphone. While it is impressive in that it returns small result sets, it is also worse at finding correct results than Soundex.

A further possibility is an algorithm by Bickel. It finds more correct results than Soundex, if the algorithm is configured to return result sets the same size as Soundex. However it also differs from Soundex and the other algorithms by ranking its results. My tests showed that it placed the correct results at or very near the top of the list of candidate matches. Another possible advantage of this approach is that it may work better with very large databases. If Soundex returns ten matches for a query for a database of 10000 entries, it would return roughly 100 entries for a database of 100000 entries. A ranking algorithm could return the best 10 or 20 entries, whatever the database size.

Bickel is slower computationally than Soundex, particularly because it is harder to index searches to the same degree. However, my tests showed that the Bickel algorithm could still be evaluated in a tiny fraction of a second for a database of over 16000 entries; it was also faster than a non-indexed any-substring matching routine. I believe that the Bickel algorithm is a realistic alternative to Soundex, particularly for small and medium size databases. It is best used as a ranking algorithm, where it is able to place the correct entry top of the list 80% of the time.

However, there is more to an approximate matching algorithm than dealing with misspellings. In Chapter 5, we saw that we could improve matching performance by various techniques such as removing stop list words from input, removing punctuation characters, matching names even when words were not in the same order, and matching names and initials, particularly in human names. An approximate matching algorithm could usefully perform all these tasks.

One problem for a DUA designer is that the standard does not specify what an approximate
matching algorithm should do. It is possible that DUA designers want very different things from approximate matching algorithms. For example, the UFN algorithm always uses approximate matching: if approximate matching returns large result sets, every query yields a large result set. On the other hand, the DE algorithm tries various other matching options before it uses approximate matching; DE is less concerned with large result sets for approximate matching since it only uses approximate matching when other options have been exhausted.

Fortunately the 1993 standard provides the framework of a solution to this problem. The standard now offers a feature called *extensible matching*, which allows a directory user to specify which algorithm it wants to use to match the query data. The scheme is useless unless DSA implementors actually implement alternative approximate matching algorithms. However, the scheme clearly has potential: a user or DUA designer could then select Soundex, Metaphone, Damerau, Bickel or some alternative algorithm to match their queries. Chadwick suggests that extensible matching could be used to select an algorithm that matches English versions of foreign place names to their local versions [Cha94]. As a DUA designer, I look forward to the day when facilities such as these are generally available.
Chapter 7

Conclusions

7.1 Outline

In this final chapter, we tie together the work described in the earlier chapters. In Section 7.2, we review the main findings made in this thesis. In Section 7.3, we compare three name matching algorithms, and note some improvements to these algorithms that have been discovered during the work on this thesis. In Section 7.4, we set out a number of changes that would help to improve the effectiveness of DUAs. In Section 7.5, we describe how a DSA could implement an algorithm that uses many of findings in this thesis. Section 7.6 looks at how the work in this thesis could both be improved and developed further. Finally, in Section 7.7, we consider the relevance of this thesis in the light of recent developments in directory services.

7.2 The Contribution Made by This Thesis

The main contribution of this thesis is to provide authors of directory user interfaces with information which should help those authors to write more effective DUAs. While there is a substantial body of material on the design of directory services and specification of their facilities, there has been relatively little work on how best to use those facilities to provide users with the most effective service. Furthermore, the papers on directory user agent algorithms present ideas that have either not been evaluated, or have been evaluated in special circumstances that do not apply to a production directory service running in a typical networking environment. By experimenting with real directory data and a large sample of query date, I have shown in detail how to improve the name matching performance of the UFN and DE algorithms.

The work described in this thesis is based on X.500, the international standard for directory services. However, the similarity of the core features of X.500 and other widely used directory services means that the majority of the findings are broadly applicable to directory services in general.

Although the main beneficiaries of the work should be DUA designers, there is also something here for DSA implementors, as the analysis should help them to understand better the facilities that help matching. The material on approximate matching should be of particular interest to implementors of directory servers.

The predominance of a single implementation makes it impossible to generalise about response time issues. Nevertheless the work on response times described in Appendix G demonstrates some pitfalls that directory administrators should be aware of: poor configuration of DUAs can have as much impact on response times as DSAs performance.
7.2.1 Summary of the Main Findings

The main findings are as follows:

- The study of name forms for the input categories shows that each category should be treated individually. Each category has its own quirks and the best matching performance cannot be achieved by applying identical matching techniques – e.g., exact matching followed by approximate matching – irrespective of the type of input.

- Organisation and department name input contains a substantial proportion of “stop list” words which can inhibit matching. If matching fails on input including these words, matching should then be tried on input stripped of these words. The same applies to punctuation characters.

- The distributed directory has considerable diversity of name forms. While most country and organisation entries include alternative values of their naming attributes, only about a quarter of person entries include more than one value of the commonName attribute. A successful DUA must be prepared to work hard to match directory entries, and cannot rely on matching one of many alternative name forms.

- There are very many forms of person names, both in user queries and in the directory. Single token queries are mostly surnames, which can be exact matched using the surname attribute in entries. Multiple token queries are best matched by transforming the query into an initial and surname and using cnInitStarSurname matching; about 95% of the directory’s person entries have a commonName that would be matched by this type of filter.

- The amount of misspelled input varies between the input categories. In my sample, less than 1% of country names were misspelled; about 3% of organisation and department names were misspelled; over 7% of person names were misspelled. Approximate matching is a vital matching technique for person names.

- Truncating user input to four characters, and using substring matching on the shortened string, is a useful alternative technique to approximate matching for hard-to-match organisation and department name queries. Truncation helps to match misspelled input, as misspellings tend to occur after the fourth character. Truncation also helps to match entries when there is a form mismatch between user input and the directory name.

- Short input needs special treatment. Organisation and department name input of four characters or fewer is usually either an abbreviated name or a set of initials, and matching algorithms should deal with this input accordingly. Short input also tends to match more entries than longer input. Result sets can be kept smaller if approximate matching is not used on short input.
7.3. COMPARING THREE ALGORITHMS

- Quipu's implementation of Soundex, used with the default settings, returns four times as many entries on average as conventional Soundex. Despite this profligacy it fails to match a query consisting of a forename and a surname with a directory entry of an initial and a surname.

- An algorithm that corrects typographical errors is more effective than Soundex for organisation and department name misspellings, but less effective for person name misspellings.

- The Bickel approximate matching algorithm offers greater recall and greater precision than Soundex at a cost of slower performance.

- The analysis of response times in the NameFLOW-Paradise environment (described in Appendix G revealed that (for that environment) slow DUA start-up is caused more by bad DUA configuration than by the OSI connection establishment being slow. In general, one can say that large DUA binaries and reading myriad configuration files mounted over NFS are a recipe for poor performance.

7.3 Comparing Three Algorithms

Much of the work in this thesis has been to test the ideas behind three name matching algorithms: Afifi and Huitema's read then search; Kille's UFN; and the author's own DE. In this section, we review the strengths and weaknesses of these algorithms, drawing together the findings of all earlier chapters.

However, before we compare these algorithms, we will note that the need to use these algorithms can be reduced by partly resolving queries within the DUA.

7.3.1 Query Resolution Within the DUA

There are two reasons why the bulk of query resolution can often be done within the DUA. First, a typical DUA within an organisation will be set up with default values for some of the input fields. These defaults will reflect the fact that most directory usage is for people within the same organisation: thus the country and organisation name will be defaulted to be the DN of the user's organisation. In some cases, this principle might be taken further and the department name be defaulted as well. If these default fields are accepted by the user, then there is no need for the defaulted names to be resolved against names in the directory.

Second, we can take this principle of locality a step further, as there is a tendency for an organisation's directory users to query a relatively small set of other organisations repeatedly; the same country and organisation names occurred often, although these names are not necessarily the RDN forms. However, we noted in Chapter 3 that if we use query-to-RDN name mapping tables, a DUA can convert most cases of non-RDN input into its RDN equivalent: queries known to be RDNs do not need resolving against names in the directory.
There are pros and cons to this use of mapping tables. An advantage for a DUA is that it should be faster; it can also minimise its use of a potentially congested DSA. Another advantage is that it could save money if a directory service provider were to charge for queries. A disadvantage is that the name mapping tables have to be kept up-to-date with directory usage and with names in the directory if they are to remain useful.

### 7.3.2 Read then Search

The success of this strategy depends on user input corresponding closely to names in the directory. The evidence suggests that this is generally not the case. For example:

- users often omit department names in their queries, whereas the vast majority of person entry DNs include at least one `organizationalUnitName` component;
- users tend to favour surname-only queries, whereas surnames are not used as RDNs.

One possibility is to use read and search selectively and only to try an initial read operation if a department name and a multiple token personal name have been specified in a query: for my sample, this was about 25% of all person queries.

However, the variety of formats used in queries and directory names means that even if the technique were used selectively as suggested, the read would only succeed one time in four when the technique was used. If we can establish that read operations are generally much faster than exact match search operations, the technique may be useful. The limited evidence I gathered on response times does not suggest to the author that this response time differential is likely to obtain.

### 7.3.3 UFN

A virtue of the UFN algorithm is that it always uses a single operation to try to resolve user input against a given set of directory names for any category of input, whereas the other algorithms may require multiple operations. However, the big problem with the UFN name matching strategy is its lack of selectivity. The UFN search must include filter items that do substring matching and approximate matching to handle cases where the input does not closely match the target directory name; the problem is that it does this loose matching even when the user input exactly matches the intended target entry. While this is not too severe a problem if the set of names being searched is small, it can cause large result sets if used against big organisational databases with many tens of thousands of names, and large result sets mean slower response times.

However, there are ways of mitigating these problems. A DUA can choose not to use approximate matching on short input: misspelling is not much of a problem with short input, and the large results are almost all spurious matches. A DUA can simplify the task of matching multiple token person name input by pre-processing the input into an `initial surname` form.
7.3. COMPARING THREE ALGORITHMS

Other improvements require modifications to DSAs. Although the Soundex algorithm is widely used for approximate matching, there are alternatives which offer better precision, and better recall too. However, some of the problem of large result sets that arose in my experiments occurred because the Quipu implementation of Soundex is configured to do prefix matching.

<table>
<thead>
<tr>
<th>Version of UFN</th>
<th>% matched by filter</th>
<th>Mean no. res.</th>
<th>Med. no. res.</th>
<th>Max. no. res.</th>
<th>% queries with single res.</th>
<th>% queries with correct res.</th>
<th>% results correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>90.23</td>
<td>71.35</td>
<td>2</td>
<td>3834</td>
<td>37.17</td>
<td>88.66</td>
<td>98.26</td>
</tr>
<tr>
<td>New</td>
<td>97.56</td>
<td>6.37</td>
<td>1</td>
<td>218</td>
<td>48.69</td>
<td>96.86</td>
<td>99.28</td>
</tr>
</tbody>
</table>

Table 7.1: Improvements to UFN matching of person names

We can see the effect of using the improvements to the UFN algorithm suggested in Chapter 5 in conjunction with the more selective Bickel algorithm for approximate matching in Table 7.1. The figures relate to person name queries. The revised combination of UFN algorithm and approximate matching algorithm is better on almost every count. In particular, the improvement in recall of correct results is statistically significant with a McNemar test $z$ value of 5.538, which has a corresponding $P$ value of less than 0.001.

The discussion in this section concerns the UFN name resolution. However, we should also remember that possibly the biggest problem with UFN queries is caused by the UFN query format. We found in Chapter 3 that the typeless, free format allowed by UFN makes it harder to get users to specify queries that have any chance of working: the most frequent problem is that users omit important name components in their queries.

7.3.4 DE

A key advantage of the DE strategy over the UFN approach to matching is that DE is much more selective: it has similar levels of recall to the UFN technique but offers much greater precision. The DE technique is often able to uniquely match the correct entry and its average result set size is much smaller than that for UFN. If a query proves hard to match using conventional substring or approximate matching techniques, looser matching techniques such as truncating input to four characters can be used with reasonable confidence that the result set will not be too large.

If we compare the effect of using both the improvements suggested in Chapter 5 and the Bickel algorithm for approximate matching, we see (in Table 7.2) that a DUA using the improved algorithm can correctly resolve over half the person queries not found by the original DE, and achieves this with smaller result sets on average. The improvement in the recall rate is statistically significant with a McNemar test $z$ value of 3.343, with a corresponding $P$ value of less than 0.001.

A DE that implemented all the changes suggested in Chapter 5 for all the input categories would find over 20% more person entries, largely due to improvements in organisation and de-
CHAPTER 7. CONCLUSIONS

<table>
<thead>
<tr>
<th>Version of DE</th>
<th>% matched by filter</th>
<th>Mean no. res.</th>
<th>Med. no. res.</th>
<th>Max. no. res.</th>
<th>% queries with single res.</th>
<th>% queries with correct res.</th>
<th>% results correct</th>
<th>Mean no. of searches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>95.64</td>
<td>5.34</td>
<td>1</td>
<td>427</td>
<td>61.26</td>
<td>94.07</td>
<td>98.36</td>
<td>1.27</td>
</tr>
<tr>
<td>New</td>
<td>97.91</td>
<td>3.10</td>
<td>1</td>
<td>57</td>
<td>65.27</td>
<td>97.38</td>
<td>99.47</td>
<td>1.19</td>
</tr>
</tbody>
</table>

Table 7.2: Improvements to DE matching of person names

While, in the worst case, the DE approach uses up to four searches to resolve a category of input, on average it uses between 1.2 and 2.0 searches, depending on the input category, to resolve each name component.

The evidence on response times in the NameFLOW-Paradise environment is that the DE sequence of searches approach on average takes one and a half times as long to resolve a query than a single exact match search. While we cannot be sure that this figure will be typical in a more heterogeneous environment, there are good reasons to suspect that the sequence of searches approach should not be unduly slow: the first search has to bear the cost of connection set-up; subsequent searches to the same DSA will therefore (other things being equal) be faster.

7.4 Facilities That Would Help DUA Developers

7.4.1 Features in the 1993 Standard

The work in this thesis has necessarily been based on the 1988 X.500 standard; implementations conforming to the 1993 standard have not yet been deployed extensively. However, it seems likely that implementations conforming to the new standard will be deployed over the next few years. The new standard contains several features that promise to help DUA developers to write more effective user agents.

A highly useful feature is the paged results facility which allows a DSA to pass results to a DUA a "page" at a time: the page size is determined by the DUA. This facility means that DUAs can pay less attention to phrasing their queries so that result sets are likely to be small: large result sets take longer to deliver to a DUA.

The 1993 standard offers several useful new facilities under the umbrella term of extensible matching. A feature that has enormous potential is the ability for a DUA to request that a particular non-standard matching algorithm be used to match a user's input. The usefulness of this can be understood if we consider approximate matching. This is a standard matching rule, but the standard does not specify how it is implemented. A DUA has no a priori knowledge of the characteristics of any DSA's approximate matching algorithm: for example, it might be standard Soundex, Quipu's Soundex with prefix matching, or Metaphone. With extensible matching, a
7.5. A POWERFUL STRATEGY

DSA could offer all three types of matching, and the DUA could select the algorithm it wished. While this facility would undoubtedly help DUAs, it is not clear whether DSA implementors will provide such alternative algorithms. I urge them to do so.

Another valuable feature included as part of extensible matching is the facility for a DUA to request typeless matching: this should simplify search filters when there is uncertainty about which attribute type(s) should be matched against.

The new standard includes several new attributes for naming people: these allow the component parts of a personal name to be split between different attributes, whereas the existing commonName attribute is free format. This splitting up of a person's name should simplify the matching process, or at least make it more accurate. It remains to be seen whether DSA administrators use these extra attributes.

7.4.2 Other Changes

One thing that would allow DUAs to fine-tune their queries would be knowing how many entries were in the scope of a search. For example, it makes little sense trying to avoid approximate matching when searching an organisation of fifty entries, whereas the same search of an organisation of twenty thousand entries might match hundreds of entries.

The work in Chapter 6 should have persuaded the reader that, despite its predominance, Soundex is not the only approximate matching algorithm. An algorithm that handles simple typographical errors is more effective than Soundex at matching misspelled organisation and department name input. Algorithms, such as that proposed by Bickel, that rank results allow very near matches to be offered ahead of less-obvious-but-still-feasible matches. In contrast, the Soundex algorithm simply evaluates TRUE or FALSE for any potential match. The ranking approach also allows the best $n$ matches to be offered independent of database size, whereas Soundex result sets grow proportionately with database size.

The work on approximate matching should also have demonstrated that the approximate matching problem is more than just resolving misspellings. An even bigger problem is matching input that differs in form to the name forms in the directory. While we have been able to demonstrate DUA-based techniques that deal with these form mismatches, the author believes that this matching problem falls within the remit of a DSA's approximate matching algorithm. If DSA's take on more of these form mismatches, the benefits are provided to all DUAs and DUA design can be simplified.

7.5 A Powerful Strategy

In this section I set out a matching strategy that I would like DSAs to facilitate. The algorithm is a combination of several features that we have discussed in this chapter. It builds on the DE approach, although it closely resembles the interpret functions proposed by Peterson in his paper.
CHAPTER 7. CONCLUSIONS

on Profile [Pet88].

• Name resolution of any name component requires a single search to be sent to the DSA.

• DSAs should implement a matching algorithm that applies the improved DE filters described in Chapter 5 in sequence. Each object class would have an associated matching algorithm.

• DUAs should use the extensible matching feature (available in the 1993 standard) to request the DE algorithm.

• This approach means that network traffic is minimised and there are no delays while DSAs communicate search failures back to the DUA and wait for the DUA to send the next search operation.

• The approximate matching algorithm should be based on the Bickel algorithm, or any similarly effective algorithm that ranks results.

• The DSA uses local knowledge to strip out stop list words and superfluous punctuation, both from directory names and user queries.

• The results of the search are passed back to the user one page at a time until the user decides that he/she has seen all the results he/she wishes to see.

The advantage of parceling all these features into the DSA is that DUA design is simplified and sophisticated matching facilities are available to all DUAs.

This strategy differs in one important aspect from the existing DE approach. The proposed algorithm is interactive at each stage of query resolution, as the user has to indicate if he/she is not interested in any looser matches. For example, even if a unique exact match is found, the user will also be presented with any substring and/or approximate matches; there may be several pages of such looser matches. The user has to indicate when he/she has found the required match for a given name component, so that the query can continue beneath that matched entry for the next name component. However, the existing DE behaviour of a search being resolved as soon as a match is found could also be provided by the DSA.

7.6 Possible Future Work

In many ways, I feel that the research described in this thesis has only scratched the surface of the name matching problem. There is certainly plenty of work that others could do. This work falls into three broad categories:

• There is work that could be done immediately that would help to verify or rebut some of my findings. Lack of time prevented me from doing more.

• There is work that cannot be done until a service is established that is based on many different software implementations.
7.6. POSSIBLE FUTURE WORK

- There is some work on DUA querying algorithms that is beyond the scope of this thesis.

7.6.1 Consolidating the Findings of this Thesis

Some work could be done now that would verify or rebut the findings in this thesis. Some other work would help to complete the story.

- The query data gathered was all for the DE user interface. It is possible that the usage of the DE user interface is in some way atypical, although I have no reason to believe that input to other services would be markedly different. Nevertheless, it would be reassuring to test the findings of this thesis against different sets of queries, particularly for queries made to non-X.500 directories.

- The algorithms would benefit from being assessed against larger data sets. For example, the University of Michigan database is seven times larger than the UCL database which I used in my experiments.

- The work on approximate matching algorithms would be enhanced by considering how well the different algorithms perform given different database sizes. This is clearly important given that the University of Michigan developers abandoned Soundex in favour of Meta-phone because Soundex's result sets were too big. I have produced evidence in favour of an algorithm that ranks results. However, would such an algorithm put the correct matches near the top of the list with very big databases?

- A lot of the work on improving name matching strategies is based on residual query sets that are not handled by straightforward matching techniques. These data sets constitute, by definition, a small proportion of the total query data analysed. However, many of the findings in this thesis depend on this data. Analysis of more of this residual data, for example of misspelled input, would help to bolster the findings.

- The analysis has concentrated on matching names in the UK part of the directory: some of the detailed analysis depended on my familiarity with English organisation, department and personal names. It is clear that the work on matching human names will vary between different cultures: for example, Chinese personal names differ in structure and component order to English names. It would be useful to have a clear exposition of the best person name matching strategies for different countries.

- More work is needed on how to handle result sets. A simple strategy is to request all the required attributes with every search operation. An alternative strategy is to only ask for the DNs of matching entries on the first search, and then to fetch the required attributes with a follow-up read operation: this may be a more efficient way to handle large result sets. Maybe a mixed strategy is best of all: ask for all attributes on exact match searches, but
ask for DN only when using approximate matching as the result set is likely to be bigger. Other variants could include asking for minimal result sets of one or two attributes such as telephoneNumber and rfc822Mailbox.

- The work in this thesis concentrates on the facilities provided by X.500. A useful extension to the work in this thesis would be to examine some of the name matching facilities offered by other directory services. In particular, both CCSO and Whois++ offer word-based matching. Whois++ also offers matching based on regular expressions: how useful is this in practice?

7.6.2 Work That Awaits a Mature Heterogeneous Directory

- The current X.500 directory largely comprises academic and research organisations, with relatively few commercial organisations. The directory thus lacks the diversity that we should expect in a fully fledged service. Moreover, the academic and research organisations may be thought of as a community which largely follows coordinated policies. There may be less cohesion as more commercial organisations independently join the directory service. The work on matching organisation names should be revisited when the service includes a substantial proportion of commercial organisations.

- The directory is still dominated by one implementation of X.500, the public domain Quipu, or implementations that are derived from Quipu. A consequence of this is that certain aspects of the directory service, such as database performance, the protocol handling software, choice of approximate matching algorithm, and so on are largely homogeneous whereas one would expect considerable variety in these areas in a longer-established service.

- The predominance of a single implementation in the existing directory means that it is impossible at the moment to undertake a general analysis of directory service response times. Although I have been able to analyse the matching capabilities of various querying strategies, this work lacks the context of a well-founded understanding of response times, which might indicate that certain strategies should be preferred or deprecated on the grounds of fast or slow response.

- This thesis examines name matching, an important component of a querying strategy. However, there are several other aspects of DUA querying algorithms that are not covered. One simplification I have made is assuming that all target entries are grouped together so that they can be searched in a single operation. Unfortunately this is not the case. For example, there are organisation entries under the root of the DIT, under country entries, under organisation entries, and under state and locality entries in some countries. It is not clear at the moment how to search all these places in the directory effectively. We cannot even be sure how serious a problem this will become as we do not know how the DIT will evolve. What is clear is that research is needed on this topic.
Another problem area, and one that is currently attracting considerable attention in the Internet Engineering Task Force, is the issue of how to find an entry for a person if the directory user is unsure of that person's organisation. The problem is essentially that if the target person's organisation is unknown, it is impossible to identify the DSA to be searched. The feasibility of Whois++ centroids [WFS96] and/or Index DSAs [Bar96] needs to be examined closely.

Another reason for re-examining some of the issues discussed in this thesis will come with the widespread deployment of 1993 X.500 systems. We will need to examine the impact of features such as *paged results* and *extensible matching*, if they are implemented at all. One group of implementors have told me privately that implementing *paged results* is not high on their list of priorities.

### 7.6.3 Interesting New Work Areas

Finally, there are some interesting ideas that could be explored that fall outside the remit of this thesis.

- Network bandwidth is a precious commodity. The author's experience in recent years is that international connections are chronically overloaded during working hours and that interactive services such as telnet, the World-Wide Web and white pages directory services are almost unusable. My hunch is that this situation is unlikely to change in the near future. One implication of this gloomy prognosis is that it is important to consider the use of bandwidth in terms of packet counts and packet sizes when trying to design an optimum directory querying algorithm. Maybe algorithms should adjust automatically according to criteria such as time-of-day or a history of response times to particular parts of the directory?

- The UFN algorithm, specified in RFC 1781, provides a very flexible way of resolving queries onto directory names. We have analysed in this thesis the effectiveness of the UFN search filters for matching entries. However, the UFN specification also describes several other elements of a querying algorithm, including a strategy for guessing the types of name components, and a system for searching multiple paths in the directory for poorly specified queries. The UFN algorithm is potentially very powerful. However, the ideas need to be properly evaluated, both in terms of how often it finds the correct results and how efficient it is at finding correct results.

- We noted in the introductory chapter that general information tools such as gopher and the World-Wide Web are sometimes used to provide directory services. A particularly promising approach is that used by Web indexing tools such as AltaVista, which rank results according to which fields are matched in an HTML [BLC95] document. It is easy to see how HTML
documents in the World-Wide Web could be marked in some way to indicate that they contain white pages information, and that a service like AltaVista could be tuned to search and retrieve white pages information only. However, it is not clear how well such a design would compare with services such as X.500 and LDAP. The topic is worthy of investigation.

7.7 The Relevance of the Contribution

Halfway through the work on this thesis I started to wonder how relevant the findings would be by the time the thesis was complete. At that time, deployment of X.500 seemed to have lost early momentum, and all the talk was of Whois++. However, in the last year interest in X.500 has picked up again as more implementations of the 1993 standard have become available: commerce is now interested in X.500, whereas commerce was not so interested in the incomplete 1988 standard.

Another significant change is that the major software vendors have recently decided to back LDAP. LDAP was initially derived from X.500, uses the same information model and provides almost identical querying facilities. LDAP was initially used purely as an access protocol to X.500 services. During 1996 the emphasis has changed with more sites deploying pure LDAP systems: this is possible as the most recent LDAP software from the University of Michigan now includes a stand-alone LDAP server, slapd. The similarity of X.500 and LDAP means that even if X.500 were to fall by the wayside, the usefulness of the work in this thesis is not lost as the majority of the work can be applied directly to LDAP systems.

I also believe that while the detail of the discussions in this thesis relate to X.500 (and LDAP), the findings can be adapted quite simply for services such as CCSO and Whois++.
Appendix A

Attributes and Search Filters

A.1 Attribute Types

A number of attribute types are mentioned throughout this thesis, mostly when used in search filters for matching entries. These are defined in the X.500 standard and RFC 1274. Table A.1 lists these attributes, giving their short and long mnemonics, with an explanatory note.

<table>
<thead>
<tr>
<th>Short mnemonic</th>
<th>Long mnemonic</th>
<th>Explanation/Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>countryName</td>
<td>ISO 3166 2 letter code</td>
</tr>
<tr>
<td>co</td>
<td>friendlyCountryName</td>
<td>Full country name</td>
</tr>
<tr>
<td>o</td>
<td>organizationName</td>
<td>Used for naming organisation entries</td>
</tr>
<tr>
<td>n/a</td>
<td>associatedDomain</td>
<td>E.g. ucl.ac.uk for UCL</td>
</tr>
<tr>
<td>ou</td>
<td>organisationalUnitName</td>
<td>This is used for naming departments</td>
</tr>
<tr>
<td>cn</td>
<td>commonName</td>
<td>This is used for naming person entries</td>
</tr>
<tr>
<td>sn</td>
<td>surname</td>
<td>Mandatory attribute in person entries</td>
</tr>
<tr>
<td>uid</td>
<td>userid</td>
<td>Computer login name</td>
</tr>
<tr>
<td>n/a</td>
<td>givenName</td>
<td>Defined in X.500 1993</td>
</tr>
<tr>
<td>n/a</td>
<td>initials</td>
<td>Defined in X.500 1993</td>
</tr>
<tr>
<td>n/a</td>
<td>generationalQualifier</td>
<td>Defined in X.500 1993</td>
</tr>
<tr>
<td>n/a</td>
<td>photo</td>
<td>Encoding defined in RFC 1274</td>
</tr>
<tr>
<td>n/a</td>
<td>audio</td>
<td>Encoding defined in RFC 1274</td>
</tr>
<tr>
<td>n/a</td>
<td>searchGuide</td>
<td>Suggested search criteria</td>
</tr>
</tbody>
</table>

Table A.1: Attribute types mentioned in this thesis

A.2 Filter Items

A search filter comprises one or more filter items; these are combined with Boolean ANDs, ORs and NOTs. A search filter item takes the form:

```
<AttributeType><MatchingType><AttributeValue>
```

The representations of the various types of matching are shown in the examples in Table A.2. Note that in some cases the type of matching is indicated by a combination of the matching type symbol and the format of the attribute value field: e.g, substring matching uses the equality sign as the matching type and the attribute value includes asterisk characters to represent wildcard matches. The representation of search filter items is that used by the the Quipu system's
**APPENDIX A. ATTRIBUTES AND SEARCH FILTERS**

In this thesis, I have preferred the generality of being able to indicate a type of matching without having to invent example input. This is the symbolic format.

<table>
<thead>
<tr>
<th>Matching type</th>
<th>Example query</th>
<th>Symbolic format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact</td>
<td>cn=paul bark</td>
<td>cnExact</td>
</tr>
<tr>
<td></td>
<td>cn=paul*</td>
<td>cnLeadSub</td>
</tr>
<tr>
<td></td>
<td>cn=*barker</td>
<td>cnTrailSub</td>
</tr>
<tr>
<td></td>
<td>cn=<em>bark</em></td>
<td>cnAnySub</td>
</tr>
<tr>
<td></td>
<td>cn=p*barker</td>
<td>cnInitStarLastname</td>
</tr>
<tr>
<td>Substring</td>
<td>cn~=paul berker</td>
<td>cnApprox</td>
</tr>
<tr>
<td></td>
<td>cn=*</td>
<td>cnPresent</td>
</tr>
<tr>
<td>Approximate</td>
<td>cn&gt;=p</td>
<td>cnGreaterThanEqual</td>
</tr>
<tr>
<td>Present</td>
<td>cn&lt;=p</td>
<td>cnLessThanEqual</td>
</tr>
<tr>
<td>Greater than or equal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Less than or equal</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table A.2: Examples of the various types of matching in filter items

The reader should be able to extrapolate from these examples to understand other filter items used throughout this thesis. Filter items that do not follow any of the patterns above are explained in the text.

### A.3 Boolean Expressions

These search filter items can be combined using Boolean operators. Parentheses are used when there is more than a single filter item to make interpretation of the expression clear. The following examples should clarify usage.

\[
(cn=p*) \text{ AND } (sn=barker)
\]

\[
((o=*oxford*) \text{ OR } (o=oxford)) \text{ AND } (objectClass=organization)
\]

The first filter matches all entries where the common name begins with "p" and the surname is "barker". The second filter matches all organisation entries with a name attribute which includes the string "oxford" or is approximately the same as "oxford".

### A.4 Sequences of Filters

The following syntax is used to indicate that a DE style sequence of searches is used, each search operation having a different filter. The search with filter 1 is tried first. If that search fails to find any results, the search with filter 2 is tried, and so on.

\[
<\text{filter 1}>, <\text{filter 2}>, <\text{filter 3}> \text{ etc.}
\]
Appendix B
Country and Organisation Name Mapping Tables

In this appendix, we show two instances of the names that would be stored in mapping tables containing ten entries for some of the sample data described in Chapter 3. Against each name, we show the percentage of user input (excluding the acceptance of default values) for that name.

The query data is for the combined queries made by the UCL and UCL-CS users. Mapping tables are shown for both country and organisation names. The contents of such mapping tables will clearly differ from organisation to organisation, but the distribution of queries is probably fairly representative for academic institutions in the UK.

B.1 Country Name Queries

<table>
<thead>
<tr>
<th>Query</th>
<th>%age of input</th>
</tr>
</thead>
<tbody>
<tr>
<td>uk</td>
<td>21.7</td>
</tr>
<tr>
<td>usa</td>
<td>14.9</td>
</tr>
<tr>
<td>australia</td>
<td>3.3</td>
</tr>
<tr>
<td>england</td>
<td>3.0</td>
</tr>
<tr>
<td>canada</td>
<td>2.5</td>
</tr>
<tr>
<td>germany</td>
<td>1.9</td>
</tr>
<tr>
<td>france</td>
<td>1.7</td>
</tr>
<tr>
<td>iceland</td>
<td>1.4</td>
</tr>
<tr>
<td>italy</td>
<td>1.1</td>
</tr>
<tr>
<td>new zealand</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Table B.1: Country names that would be in a ten entry mapping tables for the UCL and UCL-CS query data
### Organisation Name Queries

<table>
<thead>
<tr>
<th>Query</th>
<th>%age of input</th>
</tr>
</thead>
<tbody>
<tr>
<td>ucl</td>
<td>10.0</td>
</tr>
<tr>
<td>imperial</td>
<td>2.3</td>
</tr>
<tr>
<td>cambridge</td>
<td>1.9</td>
</tr>
<tr>
<td>qmw</td>
<td>1.4</td>
</tr>
<tr>
<td>oxford</td>
<td>1.3</td>
</tr>
<tr>
<td>kcl</td>
<td>1.3</td>
</tr>
<tr>
<td>brunel</td>
<td>1.2</td>
</tr>
<tr>
<td>kings college london</td>
<td>1.2</td>
</tr>
<tr>
<td>south bank</td>
<td>1.0</td>
</tr>
<tr>
<td>birkbeck</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Table B.2: Organisation names that would be in a ten entry mapping tables for the UCL and UCL-CS query data.
Appendix C
Details of Matching Experiments

C.1 Hard-to-match Multiple Token Organisation Queries

In Section 5.3.8, I noted that there were 66 multiple token organisation queries that failed to match any entries using the basic search filters. In this appendix I describe in detail the five strategies I tried on these hard-to-match queries.

In some of the strategies I refer to single token input: these cases are where multiple token input is reduced to a single token by the removing of stop list words. The five strategies were as follows:

A: *Approximate matching* is tried on an input string, modified as described in Section 5.3.8.

B: *Approximate matching on truncated input.* The input is reduced to the first two words, and these are truncated to four characters each. Approximate matching is tried on the modified input. For multiple token input:

\[(o^*=\text{truncatedFirstWord}) \text{ AND } (o^*=\text{truncatedSecondWord})\]

and for single token input:

\[o^*=\text{truncatedWord}\]

C: *Substring matching* is tried with the following filters. For multiple token input:

\[o=*\text{truncatedFirstWord}\text{*truncatedSecondWord*}\]

and for single token input:

\[o=*\text{truncatedWord*}\]

D: *Substring matching with ANDed components*. The input is reduced as in case C. Single token input is treated as in case C, but multiple token input uses the filter:

\[(o=*\text{truncatedFirstWord*}) \text{ AND } (o=*\text{truncatedSecondWord*})\]

E: *Substring matching with ORed components*. Identical to case D except that the multiple token filter is:

\[(o=*\text{truncatedFirstWord*}) \text{ OR } (o=*\text{truncatedSecondWord*})\]

The results of trying these five different filters are shown in Table C.1.

All the methods found the correct results in over half the cases. Approximate matching on input transformed according to the steps described at the beginning if this section was the least successful method.
APPENDIX C. DETAILS OF MATCHING EXPERIMENTS

<table>
<thead>
<tr>
<th>Filter code</th>
<th>No. correct matches</th>
<th>No. incorrect matches</th>
<th>Not found</th>
<th>Ave. no. matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>35</td>
<td>1</td>
<td>30</td>
<td>1.06</td>
</tr>
<tr>
<td>B</td>
<td>44</td>
<td>0</td>
<td>22</td>
<td>2.35</td>
</tr>
<tr>
<td>C</td>
<td>42</td>
<td>0</td>
<td>24</td>
<td>1.12</td>
</tr>
<tr>
<td>D</td>
<td>44</td>
<td>0</td>
<td>22</td>
<td>1.15</td>
</tr>
<tr>
<td>E</td>
<td>62</td>
<td>4</td>
<td>0</td>
<td>6.38</td>
</tr>
</tbody>
</table>

Table C.1: Techniques for hard-to-match multiple token organisation queries

An effective improvement was to base matching on just the first two words, and to truncate these words to four letters. This was almost equally effective with approximate matching (case B) and the substring matching techniques C and D. Of these three techniques, D is slightly superior in jointly finding the most matches but with a low average number of matches.

The ORed filter, technique E, was easily the most successful at finding entries (62 out of 66) but at a cost of often finding spurious entries. While this technique is not precise enough for general use, it might usefully be employed when all other techniques have failed. Used this way, it found 18 of the 22 entries not found by technique D, with an average number of matches of 7.23 for the set of 22 queries.

If for the moment we ignore the ORed filter technique (which finds almost everything), the two most common reasons for entries not being found are:

- The user's query contained a word not in the directory name: e.g. "Imperial College London" for Imperial College.
- The directory entry has a different name form to that entered by the user: e.g. the user entered "British Telecom" whereas the directory entry is "BT Plc".

C.2 Difficult Organisation Queries

We noted in Section 5.3.8 that truncating user input is a useful potential alternative to approximate matching for hard-to-match queries. In this appendix, we evaluate whether we should consider replacing all uses of organisation name approximate matching by the prune/truncate/substring methods, or whether we can somehow use these methods in tandem with approximate matching.

These ideas are evaluated on the set of queries which have corresponding entries in the directory, but that are not found by the following DE strategy. It is based on the version of DE described as case F in Section 5.3.4, except that it does not use approximate matching. The strategy is:

if (single token)
if (input length <= 3)
    oExact, assocDomainLeadSubDot, oLeadSub
else
    oExact, assocDomainLeadSubDot, oLeadSub, oAnySub
else /* multiple token */
    oExact, oLeadSub, oAnySub

This produced a set of 434 unmatched queries, 12.15% of all queries for which there is an entry in the directory.

Several different search strategies are evaluated. First, we try some experiments with approximate matching, to see if we can use it more successfully with some pre-processing of the input. Examples of this pre-processing include:

- Separating the input into words and matching the words individually.
- Only considering the first two words in the input.
- Stripping the input of stop list words.
- Truncating the tokens to various shorter lengths.

We then evaluate some similar techniques but using substring matching instead of approximate matching.

A further aspect of the evaluation is to determine to what length the input tokens should be truncated, if at all (as hitherto we have assumed that four characters is reasonable choice).

Having examined various matching strategies, we take two of the better substring and approximate matching techniques and see how a multiple search strategy delivers better results still.

Overall, fifteen strategy/truncation length combinations are evaluated. Five examples are given of approximate matching, four of substring matching and six combinations of the two types of matching. The combination searches are where an initial search (using substring or approximate matching) fails to find any results, a follow-up search using the other type of filter is tried. The fifteen cases are as follows:

A: Original approximate matching. The strategy of oApprox matching on the full input is included primarily for comparison with the other methods.

B: Approximate matching on individual words, filter items ANDed. This strategy makes use of all the input, but uses approximate matching on individual words. If the input is three characters or fewer, the input is treated as a set of initials. If the input is a single token, it is approximately matched. If there multiple input tokens, the words are individually approximately matched, and the filter items are ANDed. The filter is:
if (single token AND (input length <= 3))
   oApproxOnInitials
else if (single input token)
   oApprox
else
   (o "=firstInputWord) AND (o "=secondInputWord) AND ...

C: As for case B, except that only the first two words in multiple token input are used in the filters. The filter for multiple token input is:
   (o "=firstInputWord) AND (o "=secondInputWord)

D: As for case C, except that the input is first stripped of stop list words. The filter is thus:
if (single token AND (input length <= 3))
   oApproxOnInitials
else if (single input token)
   oApprox
else
   (o "=firstInputWord) AND (o "=secondInputWord)

E: As for case D except that the first two input words, both truncated, are matched in a single filter. The filter for multiple token input is thus:
   (o "=truncatedFirstWord truncatedSecondWord)

F: Case D, except that input tokens are truncated to five characters.

G: Case D, except that input tokens are truncated to four characters.

H: Case D, except that input tokens are truncated to three characters.

I: Substring matching on ordered truncated words from input data stripped of stop list words.
The input is stripped of stop list words. Single token input of fewer than four characters is treated as a set of initials. Single token input is truncated to four characters and matched with an oAnySub filter. Multiple token input has its first two words truncated to four characters and matched as an ordered pair of substrings using the filter as shown below.
The complete filter is:
if (single token AND (length of input <= 3))
   o=firstChar* secondChar* thirdChar
else if (single token)
   o=truncatedWord*
else
   o=truncatedFirstWord*truncatedSecondWord*
J: As for case I except that multiple token words are matched individually, and thus can be matched even if the input is in a different order to that in the directory. Multiple token input is matched using the following filter:

\[(o=^{*\text{truncatedFirstWord}*}) \text{ AND } (o=^{*\text{truncatedSecondWord}*})\]

K: Case J, except that input tokens are truncated to three characters.

L: Case J, except that input tokens are truncated to five characters.

M: Case J, except that input tokens are truncated to six characters.

N: Substring matching technique J (4 characters) followed by approximate matching technique D (no truncation).

O: Approximate matching technique D (no truncation) followed by substring matching technique J (4 characters).

<table>
<thead>
<tr>
<th>Filter code</th>
<th>No. of correct results</th>
<th>No. of incorrect results</th>
<th>No. of matches not found</th>
<th>No. of OK single matches</th>
<th>Average no. of results</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>198</td>
<td>25</td>
<td>211</td>
<td>153</td>
<td>3.19</td>
</tr>
<tr>
<td>B</td>
<td>327</td>
<td>6</td>
<td>101</td>
<td>269</td>
<td>1.05</td>
</tr>
<tr>
<td>C</td>
<td>343</td>
<td>28</td>
<td>63</td>
<td>210</td>
<td>16.86</td>
</tr>
<tr>
<td>D</td>
<td>378</td>
<td>4</td>
<td>52</td>
<td>258</td>
<td>1.90</td>
</tr>
<tr>
<td>E</td>
<td>377</td>
<td>3</td>
<td>54</td>
<td>257</td>
<td>1.89</td>
</tr>
<tr>
<td>F</td>
<td>390</td>
<td>6</td>
<td>38</td>
<td>178</td>
<td>3.01</td>
</tr>
<tr>
<td>G</td>
<td>396</td>
<td>7</td>
<td>31</td>
<td>160</td>
<td>3.57</td>
</tr>
<tr>
<td>H</td>
<td>403</td>
<td>8</td>
<td>23</td>
<td>114</td>
<td>9.44</td>
</tr>
<tr>
<td>I</td>
<td>371</td>
<td>1</td>
<td>62</td>
<td>207</td>
<td>2.51</td>
</tr>
<tr>
<td>J</td>
<td>374</td>
<td>1</td>
<td>59</td>
<td>210</td>
<td>2.51</td>
</tr>
<tr>
<td>K</td>
<td>390</td>
<td>4</td>
<td>40</td>
<td>165</td>
<td>3.24</td>
</tr>
<tr>
<td>L</td>
<td>351</td>
<td>1</td>
<td>82</td>
<td>198</td>
<td>2.38</td>
</tr>
<tr>
<td>M</td>
<td>338</td>
<td>1</td>
<td>95</td>
<td>211</td>
<td>2.23</td>
</tr>
<tr>
<td>N</td>
<td>401</td>
<td>2</td>
<td>31</td>
<td>223</td>
<td>2.72</td>
</tr>
<tr>
<td>O</td>
<td>399</td>
<td>4</td>
<td>31</td>
<td>266</td>
<td>2.03</td>
</tr>
</tbody>
</table>

Table C.2: Methods for matching organisation entries with hard-to-find queries

All the techniques other than case A found the correct result for over three-quarters of the queries. Technique B is similar to technique A except that the input tokens do not have to be in the same order as they appear in the directory. The dropping of the word order criterion can be
seen to be a useful step: B gets 65% more matches than A. A further increase in the number of matches can be achieved by only matching the first two words of input (case C). However, this technique is of limited usefulness (in the academic community at least) as with input such as "University of Foo", the crucial component is dropped. We can solve this problem by stripping stop list words from the input before trying matching (case D). This appears to be a very useful technique, with a relatively high number of matches, few errors, a high number of single result sets and a low average number of matches. We can see that the word order problem is almost entirely due to stop list words by looking at the results for technique E; the results are almost identical to case D.

There are advantages and disadvantages in truncating the tokens with approximate matching. Techniques F to H show that more matches are found, but there are more errors, far fewer cases where a single result is returned and appreciably bigger average result sets. These methods could be considered if the number of results returned is not an issue.

We consider fewer variations with substring matching as we have confirmed from the approximate matching analysis that stripping stop list words appears to be a vital step to getting a high proportion of correct results. Techniques I and J both use stripped input, with tokens truncated to four characters. Insisting on the order of input tokens has very little impact on results; this confirms the analysis from the tests on approximate matching. Techniques K to M repeat technique J except with the input tokens truncated to different lengths. Truncating to three characters gets a few more matches, but at a cost of fewer single result sets and a higher average number of entries returned. Truncating to five or six characters finds fewer correct entries, but is more precise with a lower average number of matches. Truncating to four characters strikes a balance between the extremes.

Two of the best balanced techniques are D, using substring matching, and J, using approximate matching, with the approximate matching technique appearing marginally superior because of the lower average number of results returned.

Cases N and O give two examples of combinations of these two techniques. Technique N tries substring matching first (technique J), and then approximate matching (technique D) if the substring search fails; case O tries the same two filters in the reverse order. The combination of the substring and approximate techniques returns about 6% more correct matches (from the sample of 434) than the single search strategies, with the best option probably being to use approximate matching first, followed by substring matching: this has an appreciably lower number of average results.

### C.3 Difficult Department Queries

In this appendix, we assess whether some of the techniques that we found to be useful for matching organisation queries are also useful for department name matching. A summary of this section is
presented in Section 5.4.7.

The techniques are evaluated on the set of 150 queries which had corresponding entries in the
directory, but that were not found by the following DE strategy. This is the normal DE strategy,
except omitting the ouApprox filter.

ouExact, ouLeadSub, ouAnySub

This set of 150 queries constituted 19.12% of all queries for which there was an entry in the
directory.

Several different search strategies are evaluated. First, we try some experiments with approx­
imate matching, to see if we can use it more successfully with some pre-processing of the input.
This pre-processing includes:

- Splitting the input into separate tokens and approximately matching the words individually.
- Only considering the first two words in the input.
- Stripping the input of stop list words.
- Truncating the tokens to various shorter lengths.

We then evaluate some similar techniques but using substring matching instead of approximate
matching. Having examined various matching strategies, we can take some of the better substring
and approximate matching techniques and see how a multiple search strategy delivers better
results still.

Overall, fourteen strategies are evaluated. Six examples are given of approximate matching,
six of substring matching and two combinations of the two types of matching. The combination
searches are where an initial search (using substring or approximate matching) fails to find any
results, a follow-up search using the other type of filter is tried. The fourteen cases are as follows:

A: Original approximate matching. The strategy of ouApprox matching on the full input is
included for comparison with the other methods.

B: Approximate matching on individual words, filter items ANDed. This strategy makes use of
all the input, but uses approximate matching on individual words. If the input is four
characters of fewer, the input is treated as a set of initials. If the input is a single token,
it is approximately matched. If there are multiple input tokens, the words are individually
approximately matched, and the filter items are ANDed. The filter is:

\[
\text{if (single token AND (input length < 5))}
\]

\hspace{1cm} \text{ouApproxOnInitials}

\[
\text{else if (single input token)}
\]

\hspace{1cm} \text{ouApprox}

\[
\text{else}
\]

\hspace{1cm} (ou"=firstInputWord) AND (ou"=secondInputWord) AND ...
C: As for case B, except that the input is first stripped of stop list words, and only the first two
words of multiple token input are considered. The filter is thus:

```plaintext
if (single token AND (input length < 5))
    ouApproxOnInitials
else if (single token)
    ouApprox
else
    (ou"=firstWord) AND (ou"=secondWord)
```

D: As for case C, except that the words are truncated to four characters.

E: As for case D, except that all input (apart from that of four letters or fewer) is reduced to
the first word. The filter is:

```plaintext
if (single token AND (input length < 5))
    ouApproxOnInitials
else
    (ou"=firstWord)
```

F: As for case E, except that all input is reduced to four characters, or to the first word if that
is shorter.

G: Substring matching on ordered truncated words from input data stripped of stop list words.
The input is stripped of stop list words. Single token input of fewer than five characters
is treated as a set of initials. Other single token input is truncated to four characters and
matched with an ouAnySub filter. Multiple token input has its first two words truncated
to four characters and matched as an ordered pair of substrings using the filter as shown
below. The complete filter is:

```plaintext
if (single token AND (input length < 5))
    ouApproxOnInitials
else if (single token)
    ou=*truncatedWord*
else
    ou=*truncatedFirstWord*truncatedSecondWord*
```

H: As for case G except that multiple token words are matched individually, and thus can be
matched even if the input is in a different order to that in the directory. Multiple token
input is matched using the following filter:

```plaintext
(ou=*truncatedFirstWord*) AND (ou=*truncatedSecondWord*)
```

I: As for case H except that only the first word is matched. The filter, other than for short input,
is:
C.3. DIFFICULT DEPARTMENT QUERIES

(ou=*truncatedFirstWord*)

J: As for case I except that ouLeadSub matching is used instead of ouAnySub. The filter, other than for input of four characters or fewer, is:

(ou=truncatedFirstWord*)

K: As for case I except that the filter is based on the first word of the original input.

L: As for case J except that the filter is based on the first word of the original input.

M: Substring matching technique H is tried first. If this fails, approximate matching technique C is tried.

N: Approximate matching technique C is tried first. If this fails, substring matching technique I is tried.

<table>
<thead>
<tr>
<th>Search strategy</th>
<th>% matched by filter</th>
<th>Mean no. res.</th>
<th>Med. no. res.</th>
<th>Max. no. res.</th>
<th>% queries with single res.</th>
<th>% queries with correct res.</th>
<th>% results correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>46.00</td>
<td>2.07</td>
<td>0</td>
<td>28</td>
<td>30.00</td>
<td>40.00</td>
<td>86.96</td>
</tr>
<tr>
<td>B</td>
<td>66.67</td>
<td>1.20</td>
<td>1</td>
<td>9</td>
<td>56.67</td>
<td>62.00</td>
<td>93.00</td>
</tr>
<tr>
<td>C</td>
<td>84.00</td>
<td>1.88</td>
<td>1</td>
<td>17</td>
<td>66.00</td>
<td>84.00</td>
<td>100.00</td>
</tr>
<tr>
<td>D</td>
<td>92.67</td>
<td>2.90</td>
<td>1</td>
<td>17</td>
<td>52.00</td>
<td>88.67</td>
<td>95.68</td>
</tr>
<tr>
<td>E</td>
<td>92.67</td>
<td>3.07</td>
<td>1</td>
<td>27</td>
<td>48.67</td>
<td>90.00</td>
<td>97.12</td>
</tr>
<tr>
<td>F</td>
<td>98.00</td>
<td>4.85</td>
<td>3</td>
<td>27</td>
<td>34.67</td>
<td>92.67</td>
<td>94.56</td>
</tr>
<tr>
<td>G</td>
<td>82.67</td>
<td>1.25</td>
<td>1</td>
<td>9</td>
<td>67.33</td>
<td>82.00</td>
<td>99.19</td>
</tr>
<tr>
<td>H</td>
<td>84.00</td>
<td>1.33</td>
<td>1</td>
<td>9</td>
<td>61.33</td>
<td>83.33</td>
<td>99.21</td>
</tr>
<tr>
<td>I</td>
<td>89.33</td>
<td>2.48</td>
<td>2</td>
<td>12</td>
<td>38.67</td>
<td>87.33</td>
<td>97.76</td>
</tr>
<tr>
<td>J</td>
<td>84.67</td>
<td>1.27</td>
<td>1</td>
<td>4</td>
<td>57.33</td>
<td>80.67</td>
<td>95.28</td>
</tr>
<tr>
<td>K</td>
<td>89.33</td>
<td>2.44</td>
<td>2</td>
<td>12</td>
<td>38.67</td>
<td>86.67</td>
<td>97.01</td>
</tr>
<tr>
<td>L</td>
<td>84.00</td>
<td>1.26</td>
<td>1</td>
<td>4</td>
<td>57.33</td>
<td>80.67</td>
<td>96.03</td>
</tr>
<tr>
<td>M</td>
<td>89.33</td>
<td>1.60</td>
<td>1</td>
<td>9</td>
<td>62.00</td>
<td>88.67</td>
<td>99.25</td>
</tr>
<tr>
<td>N</td>
<td>94.67</td>
<td>2.35</td>
<td>1</td>
<td>17</td>
<td>68.67</td>
<td>92.67</td>
<td>97.89</td>
</tr>
</tbody>
</table>

Table C.3: Methods for matching department entries with hard-to-find queries

All the techniques that are based on input stripped of stop list words find at least 80% of the correct entries, with few errors; the worst accuracy was 94.56% of query result sets containing the correct result. There are the inevitable trade-offs between the precision and recall of the techniques. If good precision is important, techniques C, G and H can all be recommended, with each of these techniques returning a single result in over 60% of all these hard-to-find queries. Technique F gets the most correct results, but at a cost of a relatively high average result set size.
As for organisation name matching there is little to choose between the recall rate for approximate matching on full input words, and substring matching on words truncated to four characters. Both techniques could be made to return more matches if input words were truncated to shorter lengths.

However, there is a simple technique (cases K and L) that works nearly as well as any of the other techniques. This approach is to truncate the original input to four characters, and to try either ouLeadSub or ouAnySub matching on the four character string; strings of four characters or fewer are still handled specially as sets of initials. Using ouLeadSub matching is more selective, but gets slightly fewer matches than ouAnySub matching.

The technique works well for several reasons. First, there are relatively few stop list words in my sample of department name input, and most of the first words in users' input are semantically useful terms. Second the problem of apostrophe 's'es occurs at word ends rather than word beginnings. Third, stop list characters such as ampersand are removed by truncation. Fourth, spelling mistakes tend to come later in input, rather than in the first few characters. Fifth, the less input there is, the greater the likelihood that it will match the intended entry. Due to its simplicity this appears to be a valuable matching technique.

Using a sequential combination of approximate and substring filters as in case N is a potentially useful solution. Some queries are matched using one technique, others using the other technique, depending on the nature of the query. Technique N allows the same recall rate as the best individual filter technique, but with less than half the average number of results. Furthermore this technique produced a single result for over two-thirds the queries.

C.4 Techniques for Matching Multiple Token Person Queries

In this appendix we evaluate six alternative filters for matching multiple person name token input. The query set consists of 241 multiple token names; the queries are applied to the UCL database. A full description of the queries and data is given in Section 5.5. The findings of this section are summarised in Section 5.5.8. The six filters are:

A: cnInitStarLastname The filter already described in Section 5.5.2.

B: cnInitStarLastnameStar The same as case A except that a trailing wild-card is appended. The filter is thus:

\[ \text{cn=firstChar*lastWord*} \]

C: cnInitStarSnExact The first character is matched against the beginning of the commonName attribute and the last component is matched exactly against the surname.

\[ (\text{cn=firstChar*}) \text{ AND (sn=lastWord}) \]
D: \textit{cnFirst Name Sn Ezact} As for case C except that the first word is matched rather than just the first character.

\begin{align*}
\text{cn}=&\text{firstWord}^* \land \text{sn} = \text{lastWord}^*
\end{align*}

E: \textit{cnInitStar Sn Trunc} The first character is matched against the beginning of the commonName attribute and the first three letters of the last component are matched against the leading characters of the surname.

\begin{align*}
\text{cn}=&\text{firstChar}^* \land \text{sn} = \text{truncatedLastWord}^*
\end{align*}

F: \textit{cnApproxInULastname} A name is constructed from the first initial and the last name, separated by a space, and this name is approximately matched. The filter is thus:

\begin{align*}
\text{cn}=&\text{firstChar lastWord}
\end{align*}

<table>
<thead>
<tr>
<th>Filter type</th>
<th>% matched by filter</th>
<th>Mean no. res.</th>
<th>Med. no. res.</th>
<th>Max. no. res.</th>
<th>% queries with single res.</th>
<th>% queries with correct res.</th>
<th>% results correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>86.72</td>
<td>1.21</td>
<td>1</td>
<td>23</td>
<td>74.69</td>
<td>86.31</td>
<td>99.52</td>
</tr>
<tr>
<td>B</td>
<td>87.14</td>
<td>1.47</td>
<td>1</td>
<td>61</td>
<td>73.44</td>
<td>86.72</td>
<td>99.52</td>
</tr>
<tr>
<td>C</td>
<td>86.31</td>
<td>1.10</td>
<td>1</td>
<td>8</td>
<td>75.52</td>
<td>85.89</td>
<td>99.52</td>
</tr>
<tr>
<td>D</td>
<td>64.32</td>
<td>0.76</td>
<td>1</td>
<td>8</td>
<td>58.09</td>
<td>64.32</td>
<td>100.00</td>
</tr>
<tr>
<td>E</td>
<td>91.29</td>
<td>2.31</td>
<td>1</td>
<td>17</td>
<td>60.58</td>
<td>90.04</td>
<td>98.64</td>
</tr>
<tr>
<td>F</td>
<td>97.51</td>
<td>4.30</td>
<td>1</td>
<td>116</td>
<td>59.34</td>
<td>95.85</td>
<td>98.30</td>
</tr>
</tbody>
</table>

Table C.4: A comparison of several multiple token person name filters

The results of the comparison are presented in Table C.4. Filters A, B and C produce broadly similar results. Filter B gets a few more matches than A and C, but the extra results are mostly incorrect. C appears to be marginally more accurate than A, but the results are so similar that the choice of filter should probably be decided on other grounds, for example response time.

Filters C and D provide an interesting comparison; they are the same except that filter C matches the first initial, while D matches the first word. While C is a little less accurate, it gets 33% more matches. Insisting on matching a first name can exclude a lot of valid matches. As the UCL database is broadly typical of the directory as a whole with respect to the proportion of entries lacking a forename in the \textit{commonName} attribute, this finding should hold for the directory in general.

The second most successful technique for finding correct results is filter E which matches truncated surnames, and thus matches some misspelled input. The most successful filter is filter F which uses approximate matching on input which is reduced to an initial and a surname. Its ability to match misspelled surnames means that it gets over 5% more matches than the next
best technique. However, its accuracy is the worst of the six techniques and its average result set is twice as large as the next most profligate filter.

There are various problems of matching too many or too few entries with these techniques. A potential problem of matching too many entries that occurs with any of these filters that normalise the input to a single initial plus surname is that input such as “arthur smith” matches “Alan Smith”. However, all the filters that normalised the input returned median result sets of a single entry. Over-matching was not a problem for the UCL database.

Two other potential problems of over-matching were not exposed by the UCL database. First, a query of “mike smith”, if reduced to “m smith”, matches “Mr David Smith”. About 5% of UK person entries have names in this format (see Chapter 4), but the format appears to be used little elsewhere.

Another problem only affects filters C and E, and only in cases where the forename initial in the query is the same as the first letter of the surname. A query such as “steve smith”, reduced to “s smith”, matches the entry “Eric Smith” if that entry includes its surname or any surname-first forms as alternative values of the commonName attribute. In Chapter 4 we saw that neither of these formats is very common.

There are also problems of not finding the required entries. We saw in Chapter 4 that 95% of entries had a name that conformed to the initial-star-surname format. Assuming a name of Alan Smith, the following examples do not conform to this format.

- Alan Smith Jr.
- Mr Alan Smith
- Smith, Alan
- Alan Smith 2

The ideal situation would be for every entry to include at least one commonName value in the normal format. However, DUA writers cannot assume this. While filters could be constructed to match these name forms, the simplest way round this problem is probably to try and match on the surname attribute only if other types of matching fail. The technique is discussed in Section 5.5.10.

### C.5 Database Size and Entries Matched

The size of an organisational database clearly has ramifications on the choice of querying algorithm. If the database is small a profligate algorithm may still only match a few entries; if the database is large, even exact matching may return a lot of entries for common names such as “smith” and “jones”.

29 database administrators provided me with information on the relationship between database size and the number of entries with unique surnames, surnames with two entries, surnames with three entries and so on.
Figure C.1 shows the relationship between database size and the proportion of unique surnames. It shows that for a database of approximately 5000 entries, half of surname-only queries will match two or more entries. For a database of 100000 entries, only a quarter of surname queries will uniquely match an entry.

Figure C.2 shows the proportion of surname queries that will return fewer than 24 entries (i.e., a screenful). For a database of 15000 entries, roughly 10% of surname queries will return more than a screenful of names. For a database of 100000 entries, roughly 30% of surname queries will return more than a screenful of names.

Given that these result set sizes are those for exact matching, these figures suggest that it is important to avoid using looser forms of person name matching unless strictly necessary on large databases. The DE sequence of searches approach achieves this. DUA designers would also benefit from knowing the size of a database: this would allow some tailoring of search filters. This information could be stored as an attribute of an organisation’s entry. Ideally it would indicate how many entries there were of each object class in the organisational subtree.
Figure C.2: The relationship between database size and the proportion of queries returning less than a screenful (24) of results
Appendix D

Approximate matching in Quipu

The experiments described in Chapter 5 which use approximate matching use Quipu's Soundex-based algorithm. A full description of Soundex is given in Chapter 6. Quipu's approximate matching algorithm also includes facilities to handle punctuation characters and the matching of multiple tokens. Essentially Quipu turns input and directory names into sequences of Soundex tokens, and these tokens are then matched as described below. Some configuration options are possible; however, Chapter 5's experiments use Quipu's default settings. These settings are as follows:

- All punctuation characters are treated as word-separating characters such as spaces or tabs. Thus input of "U.K." is transformed into "U K ", and input of "Queen's University" is transformed into "Queen s University". These transformations are done before the individual tokens are turned into Soundex keys for matching.

- Soundex keys are of unlimited length, whereas the standard algorithm sets a maximum key length of four characters. A maximum length of four characters can be configured.

- Quipu allows prefix matching. This means that a short key based on user input is regarded as matching a longer key based on a directory name if the short key matches the leading characters of the longer key. This is useful for matching abbreviated input. For example, "P" matches "Paul" and "Uni" matches "University". However, input of "Paul" does not match a token of "P" in a directory name. Prefix matching can be disabled to allow normal Soundex matching.

- Input that is transformed into the sequence of Soundex tokens "A C" or "A B" matches a directory name which has the Soundex tokens "A B C". The match succeeds so long as the target name includes all the query's Soundex tokens in the correct order: query tokens "B A" do not match target name tokens "A B".

- The match does not work if the input string contains tokens that do not appear in the target name. Thus, a query of "A B C" does not match a target name of "A C".

Table D.1 gives a number of examples of mis-matches of form, and notes whether or not these mis-matches are handled successfully by Quipu.
### Table D.1: Examples of user input and directory name form mis-matches

<table>
<thead>
<tr>
<th>User Input</th>
<th>Directory Value</th>
<th>Matches in Quipu</th>
</tr>
</thead>
<tbody>
<tr>
<td>barker, paul</td>
<td>Paul Barker</td>
<td>No</td>
</tr>
<tr>
<td>paul barker</td>
<td>P Barker</td>
<td>No</td>
</tr>
<tr>
<td>p barker</td>
<td>Paul Barker</td>
<td>Yes</td>
</tr>
<tr>
<td>o’neill</td>
<td>Oneill</td>
<td>No</td>
</tr>
<tr>
<td>genetics &amp; biometry</td>
<td>Genetics and Biometry</td>
<td>Yes</td>
</tr>
<tr>
<td>genetics and biometry</td>
<td>Genetics &amp; Biometry</td>
<td>No</td>
</tr>
<tr>
<td>isd</td>
<td>Information Systems Division</td>
<td>No</td>
</tr>
<tr>
<td>information</td>
<td>Information Systems Division</td>
<td>Yes</td>
</tr>
<tr>
<td>maths</td>
<td>Mathematics</td>
<td>No</td>
</tr>
<tr>
<td>mathematics</td>
<td>Mathematical Science</td>
<td>No</td>
</tr>
<tr>
<td>department of chemistry</td>
<td>Chemistry</td>
<td>No</td>
</tr>
<tr>
<td>university of cambridge</td>
<td>Cambridge University</td>
<td>No</td>
</tr>
<tr>
<td>queen’s university</td>
<td>Queens University</td>
<td>No</td>
</tr>
<tr>
<td>university college, london</td>
<td>University College London</td>
<td>Yes</td>
</tr>
<tr>
<td>u.k.</td>
<td>UK</td>
<td>No</td>
</tr>
<tr>
<td>u.k.</td>
<td>United Kingdom</td>
<td>Yes</td>
</tr>
<tr>
<td>uk</td>
<td>United Kingdom</td>
<td>No</td>
</tr>
</tbody>
</table>
Appendix E

Metaphone Codes in Full

The Metaphone codes listed below are those used in release 2.0 of the ISODE Consortium's version of Quipu. The same encodings are also used in the University of Michigan's LDAP release, version 3.2.

The codes reproduced below differ slightly from those given by Mavrogeorge [Mav95] who reports on the work of Philips [Phi90]. It is not clear why the differences exist. I suspect that they are mostly due to a failure of the author of the original C code [Par91] to faithfully follow Philips's design, although some differences may be deliberate.

<table>
<thead>
<tr>
<th>Letter</th>
<th>Code</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>if first letter</td>
</tr>
<tr>
<td></td>
<td>n/a</td>
<td>otherwise</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>n/a</td>
<td>no code if -SCE-, -SCI- or -SCY-</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>if -CIA- or -CH-</td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>if -CI-, -CE, -CY-</td>
</tr>
<tr>
<td></td>
<td>K</td>
<td>otherwise</td>
</tr>
<tr>
<td>D</td>
<td>J</td>
<td>if in -DGE-, -DGI-, -DGY-</td>
</tr>
<tr>
<td></td>
<td>T</td>
<td>otherwise</td>
</tr>
<tr>
<td>E</td>
<td>E</td>
<td>if first letter</td>
</tr>
<tr>
<td></td>
<td>n/a</td>
<td>otherwise</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>n/a</td>
<td>no code if in -GH- and not at end or before a vowel</td>
</tr>
<tr>
<td></td>
<td>n/a</td>
<td>if in -GN or -GNED, or if in -DGE-, -DGI-, -DGY-</td>
</tr>
<tr>
<td></td>
<td>J</td>
<td>if before I, or E, or Y and not GG</td>
</tr>
<tr>
<td></td>
<td>K</td>
<td>otherwise</td>
</tr>
<tr>
<td>H</td>
<td>n/a</td>
<td>silent if after vowel and no vowel follows</td>
</tr>
<tr>
<td></td>
<td>n/a</td>
<td>silent if after C, G, P, S, T</td>
</tr>
<tr>
<td></td>
<td>H</td>
<td>otherwise</td>
</tr>
<tr>
<td>I</td>
<td>I</td>
<td>if first letter</td>
</tr>
<tr>
<td></td>
<td>n/a</td>
<td>otherwise</td>
</tr>
<tr>
<td>J</td>
<td>J</td>
<td></td>
</tr>
</tbody>
</table>

Table E.1: The Metaphone codes (A-K) as used in the Quipu system
APPENDIX E. METAPHONE CODES IN FULL

<table>
<thead>
<tr>
<th>Letter</th>
<th>Code</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>n/a</td>
<td>no code if after C</td>
</tr>
<tr>
<td></td>
<td>K</td>
<td>otherwise</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>O</td>
<td>if first letter</td>
</tr>
<tr>
<td></td>
<td>n/a</td>
<td>otherwise</td>
</tr>
<tr>
<td>P</td>
<td>F</td>
<td>if before H</td>
</tr>
<tr>
<td></td>
<td>P</td>
<td>otherwise</td>
</tr>
<tr>
<td>Q</td>
<td>K</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>X</td>
<td>if before H or in -SIO- or -SIA-</td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>otherwise</td>
</tr>
<tr>
<td>T</td>
<td>X</td>
<td>if -TIO- or -TIA-</td>
</tr>
<tr>
<td></td>
<td>0 (zero)</td>
<td>if before H</td>
</tr>
<tr>
<td></td>
<td>n/a</td>
<td>silent if in -TCH-</td>
</tr>
<tr>
<td></td>
<td>T</td>
<td>otherwise</td>
</tr>
<tr>
<td>U</td>
<td>U</td>
<td>if first letter</td>
</tr>
<tr>
<td></td>
<td>n/a</td>
<td>otherwise</td>
</tr>
<tr>
<td>V</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>n/a</td>
<td>silent if not followed by a vowel</td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>otherwise</td>
</tr>
<tr>
<td>X</td>
<td>KS</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>n/a</td>
<td>silent if not followed by a vowel</td>
</tr>
<tr>
<td></td>
<td>Y</td>
<td>otherwise</td>
</tr>
<tr>
<td>Z</td>
<td>S</td>
<td></td>
</tr>
</tbody>
</table>

Table E.2: The Metaphone codes (K-Z) as used in the Quipu system

In addition, there are a few exceptions at word beginnings. If the initial digram is a KN, GN, PN, AE or WR, drop the first letter. An initial X is changed to a Z. An initial WH is changed to a W.
Appendix F

Verifying Approximate Matches

F.1 Message to Referees

The following message was shown to some colleagues who were asked to judge the reasonableness of candidate matches produced by some approximate matching algorithms.

This file contains 31 names that were entered as queries to the DE directory service user interface. None of these names exactly matches a name in the database (they are for queries within UCL). However I believe that many, if not all, of the entered names should match an entry in the database. The problem is that the user made a typographical error, or for some reason was unable to spell the name correctly.

As part of my PhD research I am analysing the characteristics of several approximate matching algorithms. These produce quite startlingly different results. A problem I have is that if I select what I believe to be the correct match, I may introduce a bias into the results. Therefore I would like a small group of other people to help me decide which are the most appropriate matches. For any example of user input, you might decide that:

- one of the candidate matches is almost certainly the correct one
- that two (or more) candidates both seem likely and you cannot choose between them (one way in which this can happen is if a name with a typo is one character different to two candidate names).
- that none of the candidates appear likely to be correct

Note that you should only identify a candidate as being correct if you think there is a very good chance that it is what the user intended; do not give marks merely for plausible candidates, if you do not believe that that particular entry is what the user intended.

Each of the queries is underlined. Each name is followed by a list of candidate names (from one to a hundred and something) that were matched by the various approximate matching algorithms.

Could you please mark any candidate name that you think is the intended name with an asterisk, and return the results to me. Any questions, please ask.

Quick example:

bllloggs
-------
Blodwin
Bloggs *

[The data followed here...]
F.2 The Referee-Determined Approximate Matches

The following table lists the 31 misspelled names that were used in the experiments in Section 6.3 along with the referee-determined matching names from the database.

<table>
<thead>
<tr>
<th>Misspelled Name</th>
<th>The Correct Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assimakopoylos</td>
<td>Assimacopoulos</td>
</tr>
<tr>
<td>Bberzon</td>
<td>Berson</td>
</tr>
<tr>
<td>Chaudri</td>
<td>Chaudhri</td>
</tr>
<tr>
<td>Christmast</td>
<td>Christmas</td>
</tr>
<tr>
<td>Chryisky</td>
<td>Chyriwsky</td>
</tr>
<tr>
<td>Chrysky</td>
<td>Chyriwsky</td>
</tr>
<tr>
<td>Cooms</td>
<td>Coombs</td>
</tr>
<tr>
<td>Darleson</td>
<td>Darlison</td>
</tr>
<tr>
<td>Davies</td>
<td>Davis</td>
</tr>
<tr>
<td>Espinbola</td>
<td>Espindola</td>
</tr>
<tr>
<td>Flashner</td>
<td>Flaschner</td>
</tr>
<tr>
<td>Gorgio</td>
<td>Giorgi</td>
</tr>
<tr>
<td>Hanid</td>
<td>Hamid or Hanif</td>
</tr>
<tr>
<td>Harwood</td>
<td>Horwood</td>
</tr>
<tr>
<td>Hermans</td>
<td>Hermans</td>
</tr>
<tr>
<td>Howells</td>
<td>Howell</td>
</tr>
<tr>
<td>Miller</td>
<td>Miller</td>
</tr>
<tr>
<td>Murtough</td>
<td>Murtagh</td>
</tr>
<tr>
<td>Oliviera</td>
<td>Oliveira</td>
</tr>
<tr>
<td>Orme</td>
<td>Oram</td>
</tr>
<tr>
<td>Patelay</td>
<td>Patelay</td>
</tr>
<tr>
<td>Pitten</td>
<td>Pittam</td>
</tr>
<tr>
<td>Pitts</td>
<td>Pitt</td>
</tr>
<tr>
<td>Pahanks</td>
<td>Shanks</td>
</tr>
<tr>
<td>Rupelia</td>
<td>Ruparelia</td>
</tr>
<tr>
<td>Ruperelia</td>
<td>Ruparelia</td>
</tr>
<tr>
<td>Seymour</td>
<td>Seymour</td>
</tr>
<tr>
<td>Shaid</td>
<td>Saeed</td>
</tr>
<tr>
<td>Sheedy</td>
<td>Sheady</td>
</tr>
<tr>
<td>Todd-popropek</td>
<td>Todd-Pokropek</td>
</tr>
<tr>
<td>Yacub</td>
<td>Yaqoob</td>
</tr>
</tbody>
</table>
Appendix G

Response times

G.1 Introduction

In this appendix we analyse directory service response times for read and search operations. We are not interested in broader performance issues such as DSA throughput, but solely the speed of the system from the perspective of a DUA.

This type of measurement is fraught with difficulty: the directory’s distributed and heterogeneous nature means that there are many variables influencing response times. A model of response times would need to be highly complex. However, our aims are more modest. We are interested in discovering if there are general trends in directory service response times, such as whether, for example, a read is twice as fast as a search operation, or a search with a filter A is generally twice as slow as a search with filter B. Furthermore, we are only interested in user-observable differences rather than millisecond differences in response times. The intention is to provide DUA writers with information about the likely response times of various search strategies: are there grounds for preferring or deprecating strategies on the basis of response times?

An important aspect of the work is that the data on response times are intended to reflect genuine system performance. This contrasts with some response time measurements [Eme95] [NBGS92] which show the maximum capabilities of an implementation, often using machines dedicated to directory service use, with lightly loaded local area networks or loopback connections. The tests here have been designed to show the response times that an ordinary casual user of the directory might achieve, rather than what could be achieved in a carefully tuned environment. I have gathered the data using tests against a number of publicly accessible DSAs that are in use today: the tests were predominantly undertaken in Autumn 1994 and Spring 1996. I used DUAs that were installed by system administrators, rather than specially tailored versions of the software; these DUAs were available to ordinary users of the PARADISE system and users within UCL.

Most of the data was gathered using the dish DUA from within a special shell script test harness. Using this method, I gathered response time data for a wide range of query operations. I gathered some other data using a version of the DE DUA, modified to record the time taken to perform its query operations. This special version of DE was run for a short period by the PARADISE project at ULCC. The data gathered by DE is for the limited range of query operations used by DE, but is for a larger group of DSAs.

The structure of this appendix is as follows. First, in Section G.2 there is a discussion of how the response time data was gathered, the DUAs and DSAs used in the tests, and the operations and parameter settings tested. Section G.3 discusses the time it takes for a DUA to start and to establish a connection to the directory.

Sections G.4 to G.8 discuss the response times of read and search operations, and many
different search filters and search strategies.

Section G.9 examines whether there are any performance benefits to chaining or referral.

Section G.10 describes an experiment where I added some instrumentation to DE and measured the response times of the service to a wider range of DSAs than covered in the experiments described elsewhere in this appendix.

Section G.11 describes an experiment which compares system response times for DAP and LDAP versions of DE. It emphasises some of the points made in earlier sections. Section G.12 comprises a summary of the results and the conclusions.

G.2 Experimental Methodology

G.2.1 The Use of the dish DUA

The majority of the response time data discussed in this appendix was gathered using Quipu's minimalist dish DUA. dish is more a directory management user interface, or even a directory protocol exerciser, than a user interface. The user essentially has to enter a text version of the X.500 protocol for the operations he/she wishes to submit to the directory. Although it is less than ideal as a user-friendly DUA, it lends itself well to response time measurement for two reasons:

- it does nothing more than dispatch operations and crudely format results - no time is spent on user-friendly frills;
- it allows access to the full range of X.500 operations, and all the associated options.

The response time results were gathered as sketched below, using the UNIX time program to record dish's elapsed time. The testInputFile could contain none, one or several lines of dish commands.

```bash
for lotsOfTimes
  do
    date >> resultsFile
    /bin/time dish -u "" < testInputFile >> resultsFile
  done
```

One could argue that dish executed from within a shell script is not the perfect tool for collecting response time measurements. First, the timings include shell command processing and file input/output overheads. Second, the timings are of the start to finish execution time of dish. Each timing includes the time taken for dish to bind to and unbind from the directory as well as the time taken for the particular operation under test. While such timings are realistic of query processing time, they do not allow us to precisely allocate time between the bind, the operation(s)
and the unbind. A third limitation is that the time command is too coarse-grained for accurate timing, with granularity of a tenth of a second.

However, despite these shortcomings, I believe that dish is adequate for my main purpose, which is to examine broad response time trends, and to compare candidate techniques, rather than to accurately model the minutiae of directory response times to millisecond accuracy.

G.2.2 Gathering the Response Time Data

The response time data was gathered using dish DUAs on several different machines. In all cases, the DUAs were installed versions of dish available to ordinary users. The DUAs connected to various DSAs running live services. The details of the test set-up are given in Section G.2.4. The test environment was thus not tightly controlled, but was subject to the variations in response times experienced in normal usage, due to factors such as differing machine loads and network bandwidth availability. In order to even out the effect of this variability in the querying environment, individual tests were run repeatedly for periods from a few days to several weeks. There were 155 sets of tests and these were repeated for anything from 40 to almost 1500 times, usually at hour intervals: the average was over 250 iterations per test.

As well as short term variations due to load and congestion, there are also trend effects over the longer term, with increases in network traffic and server loads periodically offset by compensating upgrades to network capacity and computing power. Indeed, some tests that were run in two or more batches show substantial differences in response times between the batches. I have avoided these time-related effects by only directly comparing results from tests that were run concurrently. One effect of this is that if a set of tests is referenced in more than one section of this appendix, the response time figures for that set of tests may differ between the sections.

The timing of the tests had to be arranged very carefully to get results that reflect the response times that an ordinary casual user of the directory might obtain. While I wanted to gather as much experimental data as possible, if I made too many queries I ran the risk of prejudicing the results. There are two main ways in which this could happen. First, I noticed early on when devising the tests that the response times of a DUA seem to improve if the DUA is invoked frequently. After some experiments, I chose a minimum interval of ten minutes between tests, although the precise interval required to prevent any interference between consecutive tests differs from workstation to workstation, and is also affected by resource usage of other processes on the machine. Second, Quipu DSAs by default hold connections to remote DSAs open for five minutes after they are last used. Follow-up queries chained to the same remote DSA will thus not have to bear the cost of their local DSA connecting to that remote DSA.

I have assumed that this level of testing, at a peak of twelve queries an hour from two workstations, has a minimal impact on the DSAs, as the additional queries are a drop in the ocean compared with each DSA’s existing querying load.

Having gone to some lengths to design tests which reflect normal querying conditions, I have
devised a few tests to demonstrate the response time benefits of follow-up queries and faster query rates.

G.2.3 Processing the Results Data

The raw result data has been trimmed to exclude individual results of over 180 seconds. There are two reasons for this. First, very slow response times are usually due to non-directory factors, such as the unavailability of NFS-mounted file-store. I was able to almost eliminate this type of error by not testing during the period when file-servers were being rebooted. Second, users would generally recognise this type of failure and discontinue the queries anyway.

The timings given throughout this appendix are median response times unless stated otherwise. The distributions of response times are all heavily right-skewed (the tail is to the right); the mean is a poor measure of the average for this type of distribution [Jai91]. In some cases I have included values for the 10th and 90th percentiles to give an indication of the spread of response times.

G.2.4 DUAs and DSAs Used in the Tests

One factor that limits the worth of a response time measurement exercise is that the directory is still dominated by a single implementation, Quipu, whereas we anticipate that a mature directory will be built on a variety of implementations. However, I have managed to get response time data from four implementations, using a mixture of direct connections, chaining and referral, and using DUAs on several different machines at UCL and ULCC. Figure G.1 illustrates the configurations tested.

The DUA and DSA at ULCC were run on a SUN SPARC-10 used solely for directory services. The ULCC DSA was used only for chaining queries onto other DSAs, and held no data of interest in the experiments.

Various machines were used at UCL to run the DUAs. The majority of tests were made on a SUN SPARC 1+, the author's workstation. The other tests were run using DUAs on a variety of hardware from under-powered SUN SLCs to twin-processor Solbourne servers. UCL's Computer Science department DSA, which holds a copy of the UCL database, runs on a SUN 4/470. The UCL database contains just over 16000 entries.

Tests were also sent to a number of Quipu DSAs run by universities and colleges in the UK academic community; most institutions use SUN 4/330s for the DSAs. The institutions that assisted my testing were: University of Bath, University of Birmingham, Royal Holloway and New Bedford College, the Open University, Brighton University and the University of Salford. These held between 1600 and 13400 entries. The network bandwidth to these sites was at least 2 Mbits/sec.

The INRIA DSA uses the Pizarro implementation [Af91]: the INRIA DSA held just over a thousand entries. The maximum network bandwidth to this DSA was 256 Kbits/sec.
G.2. EXPERIMENTAL METHODOLOGY

The University of Michigan system is based on a Quipu system, but was enhanced by developers at the University of Michigan and has its own database implementation [How95b]. One difference is that this is the only DSA (of those used in my tests) that indexes *any-substring* matching. Thus, the response time characteristics for this DSA differ from other Quipu DSAs. The Michigan DSA held approximately 110,000 entries. The maximum network bandwidth to this DSA was 2 Mbits/sec.

The Y-NET project used a Siemens DSA to hold a few hundred entries. The maximum bandwidth to this DSA was 64 Kbits/sec.

G.2.5 Distributed Operations

The directory’s distributed operations are potentially complicated, possibly requiring several DSAs to be contacted to find the address of the target server. However, the measurements in this appendix are based on very simple distributed operations of the following types:

- direct connection from DUA to target DSA;

- DUA connects to local DSA, which chains query directly (no multi-DSA chains) to target DSA - the majority of queries use this approach;

- DUA connects to local DSA, which passes a referral back to the DUA indicating the target DSA.
G.2.6 The Operations Tested

I have gathered response time measurements for bind, read and search operations. In particular, I have made detailed comparisons of many search filters, from simple exact match filters, to more complex filters such as those used by the UFN algorithm. I also assess some multiple operation querying strategies such as that used by DE.

Considerable care has gone into the design of the tests to make the comparisons as meaningful as possible. For example, with a few specific exceptions, all tests return a single entry, and a single attribute – the surname – from within that entry. This control of some of the variables of X.500 operations should facilitate comparisons of the results for different tests. The following is a summary of the parameter settings for the tests.

- All binds are unauthenticated, except where indicated otherwise.

- All read and search operations are for person entries.

- The entryInformationSelection for all read and search operations (unless stated otherwise) is for a single attribute type (surname) and its value.

- All searches use wholeSubtree scope, with the organisation DN being the search baseObject.

- The attribute values in search filters are chosen carefully so that all searches yield a single result.

- Search filters differ from test to test and are described on a case by case basis.

- No search or read operations use size or time limits.

- The majority of tests are chained to the target DSA through a connection to the DUA's access point DSA. Some tests use direct connections to the DSAs holding the data: this is indicated where applicable. A few tests use referral to a remote DSA; this is indicated where applicable.

The following shorthand is used throughout this appendix to describe the tests.

\[
\text{DUA-ORG-Read (DIRECT)}
\]
\[
\text{DUA-ORG-ATT-MATCH*OPS (DIRECT)}
\]

The first example is the template for read operations; the second is for search operations. The ATT, MATCH, *OPS and (DIRECT) components can be omitted if not relevant to a particular test.

DUA Takes values UCL or ULCC. If the DUA is at UCL, the word UCL is followed by a letter indicating which workstation was used for the test, as in UCLy.

ORG The organisation being queried. The values are all obvious abbreviations of the organisation names.
Read The test is for an X.500 read operation.

ATT The attribute type used in a search filter. These are values such as CN and SN.

MATCH The type of matching used in a test. This takes values such as EXACT, ANYSUB, APPROX, SEQ and MULTI, where the meanings of SEQ and MULTI are explained in the text before the test.

*OPS Indicates that an operation is performed a number of times, as in *10.

(DIRECT) Indicates that the DUA made a direct connection to the target DSA.

G.3 Starting the DUA and Connecting to the Directory

G.3.1 Introduction

My belief, when I started work on this thesis, was that the directory's major response time problem is the length of time it takes for a DUA to bind to the directory. I showed in [Bar93] that there was relatively little difference between the overall time taken to do a read, a search or a DE sequence of searches when the time taken to bind to the system was included in the timings. In keeping with this belief, I modified DE to use asynchronous binding as soon as the Quipu DUA library provided this facility; binding to the directory is interleaved with the input of the query.

I believed that there were two principal reasons why binding was slow. First, the X.500 directory uses a full OSI communications stack. The layering of the OSI stack clearly adds some complexity to the task of connection establishment. Young summarises the connection establishment packet exchanges in [You95] and argues for a connectionless protocol for simple directory functionality such as name-server lookups.

Second, the Quipu directory software uses the ISODE OSI package, which produces notoriously large binaries. Large binaries are slower to load into memory than small processes, and cause more page faults while running; I present some evidence to support this assertion in Section G.11. My hunch was that this factor was more important than protocol complexity, as I had seen connection establishment being appreciably faster in experimental set-ups where paging was minimised or avoided.

A practical reason for wanting to understand whether binding is intrinsically slow compared with the speed of the lookup operations is that this knowledge can guide DUA design. Let us consider the following scenario. An algorithm that uses two search operations is found to be considerably superior in terms of precision and recall than an algorithm that uses a single search operation. However, we want to know the response time penalty of this strategy. If the bind takes ten seconds, and the search operations take a second each, the two strategies take a barely distinguishable eleven and twelve seconds. If, on the other hand, the bind takes one second, then
the two strategies take two and three seconds, a relatively bigger difference. The reader should appreciate my interest in such calculations, as DE's strategy is to use a sequence of searches.

Despite my experience that connection establishment could be slow, not all authors mention binding when discussing the response times of a directory implementation. Affi and Huitema, who assign relative costs of one and two to read and search operations respectively, make no allowance for bind at all [AH92]. The impressive response time figures for the Digital X.500 implementation make no mention of bind times [Eme95]. Turner and Gjeffe's measurements include bind times in overall operation times for reads and searches in their analysis of how a directory system coped with varying load [TG92].

The authors of the EAN system found that binding carries a substantial overhead [NBGS92]. They reported that a bind/read/unbind sequence took 0.9 seconds, measured at their DUA, whereas a read alone took 0.25 seconds. They suggest that their results show "the relatively high cost of connection-oriented communication". In contrast, Howes describes DAP bind times of 30 milliseconds in a test set-up designed to demonstrate the strengths of his LDAP implementation [How95a].

### G.3.2 Connection Tests

<table>
<thead>
<tr>
<th>DUA</th>
<th>DSA</th>
<th>Start dish, Bind Bind</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>no bind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bind once</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bind 10 times</td>
</tr>
<tr>
<td>ULCC</td>
<td>ULCC</td>
<td>0.6</td>
</tr>
<tr>
<td>UCLb</td>
<td>UCL</td>
<td>4.9</td>
</tr>
<tr>
<td>UCLy</td>
<td>UCL</td>
<td>8.4</td>
</tr>
<tr>
<td>UCLs</td>
<td>UCL</td>
<td>1.2</td>
</tr>
<tr>
<td>UCLf</td>
<td>UCL</td>
<td>5.1</td>
</tr>
<tr>
<td>UCLg</td>
<td>BIRM</td>
<td>4.3</td>
</tr>
</tbody>
</table>

Table G.1: Time taken in seconds to start dish and to do one/ten binds

Table G.1 shows various timings for establishing a connection to the directory. Timings were taken from six different hosts, and to three different DSAs. The results are revealing in several ways.

First, the time to bind to and unbind from the directory varies enormously for the test set-ups from a median of 0.6 seconds for the ULCC DUA and DSA to 8.5 seconds for the UCLy DUA to the UCL DSA.

Before I ran these tests, my belief was that binding was slow because of the overhead of paging the large OSI processes into memory. I tried to determine how long a bind and unbind operation took, stripped of any paging effects, by testing how long it took to bind and unbind ten times. The underlying time of a single bind and unbind can be calculated by measuring the difference
between the results for these tests and those for a single bind and unbind. The ULCC DUA and DSA, the fastest combination, took an extra half a second to do the additional nine binds and unbinds. This indicates a bind plus unbind time of just over 0.05 seconds. The UCL set-up, with queries from my own workstation, was a little slower, but still fast at about 0.08 seconds per bind plus unbind.

While these results appear to confirm the “bind is slow because of paging” theory, they are not the whole story. I ran further sets of tests to see how long it took to start the DUA to the point where it was ready to bind to the directory: these timings show how long the DUA takes to read configuration files and perform any other pre-connection initialisation. The results are illuminating. For my test set-ups, the majority of the time taken from DUA start-up to connection establishment is taken up by DUA initialisation rather than by OSI connection establishment. Paging at start-up is a problem, but bind times per se are negligible.

In the next section we look a little more closely at DUA start-up.

**G.3.3 Analysis of the Speed of DUA Start-up**

In this section, we analyse why DUA start-up can be so slow, by looking at the individual components of the start-up procedure. Although the analysis is specific to the ISODE and Quipu implementations, all implementations have to read configuration files and perform initialisation procedures. At the very least this section serves as a reminder of how important it is to streamline these procedures.

The analysis is based on the results yielded by a version of *disk* which was modified to time each element of the start-up procedure by using UNIX's *ftime* function. The tests were made from my own workstation to the UCL-CS DSA. I ran two sets of tests. I ran one set of tests with each connection at half hour intervals. This was intended to simulate the effect of normal usage by a DUA and to avoid paging effects; there were 421 measurements. I ran the second set of tests as a continuous sequence, to remove as far as possible any paging effects and to get a measure of the underlying capabilities of the system; there were 420 measurements.

What can we learn from these results? First, some elements of the start-up procedure are as much as fifteen times faster when the bind/unbind procedure is repeated rapidly. My interpretation is that these functions are initially I/O bound, either due to page faults or to reading configuration files, but benefit from caching when invoked a second time immediately afterwards. The figures also show that in an optimum set-up, with no slowing of response times due to paging, the initialisation functions took 87% of the total start-up time, with bind taking only 13%.

One problem with the UCL configuration is that some configuration files, and all DUA log files, are on NFS-mounted file-stores. NFS file access is slower than accessing local disks, as it includes the overhead of transferring data over the network.

While it is interesting to understand the factors behind system response times, can we do anything to improve start-up times? I believe that there are steps that can be taken by several
APPENDIX G. RESPONSE TIMES

<table>
<thead>
<tr>
<th>Start-up function</th>
<th>Time in msecs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Normal Usage</td>
</tr>
<tr>
<td>Initialising syntax handlers</td>
<td>411</td>
</tr>
<tr>
<td>Argument parsing</td>
<td>86</td>
</tr>
<tr>
<td>Isode tailoring</td>
<td>292</td>
</tr>
<tr>
<td>Dsap tailoring</td>
<td>116</td>
</tr>
<tr>
<td>Reading oidtables</td>
<td>925</td>
</tr>
<tr>
<td>Bind</td>
<td>1780</td>
</tr>
</tbody>
</table>

Table G.2: Analysing the components of DUA start-up and binding

people: the implementors of DUA libraries; the implementors of DUAs; the systems administrators configuring and installing the software; the DUA user.

DUA library authors can do several things to help. First, the binaries should be as small as possible to reduce paging. I saw a report several years ago of an implementation of the OSI upper layers that was 22 Kbytes of executable code; ISODE is one or two orders of magnitude larger than this. Second, the initialisation procedure should be as streamlined as possible: ISODE processes read a lot of files at start-up time. Third, the DUA should provide facilities for asynchronous binding, to allow binding to be concurrent with other initialisation activities.

The DUA writer should exploit asynchronous binding capabilities if provided by the DUA library.

<table>
<thead>
<tr>
<th>Configuration file</th>
<th>No. of lines</th>
<th>Time in msecs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Initially</td>
<td>Trimming</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Initially</td>
</tr>
<tr>
<td>dsaptailor</td>
<td>216</td>
<td>41</td>
</tr>
<tr>
<td>oidtables.*</td>
<td>1132</td>
<td>315</td>
</tr>
</tbody>
</table>

Table G.3: The processing time speed-up achieved by reducing the size of some ISODE configuration files

The DUA installer should ensure, as far as possible, that the DUA binary and any configuration files are available on a machine's local disk, as accessing files over NFS is much slower. The DUA installer should also check that tailoring and configuration files are trimmed to only include necessary details. I found that all the ISODE tables and configuration files contained data that was not required by a white pages directory service. Several of the files were prefaced by large blocks of comments. These lines are read but ignored at start-up time; a response time penalty is incurred doing this, particularly if more disk accesses are required to read in these verbose files. I was able to trim the tables and configuration files for the directory with just a few minutes
work. Some examples of the degree of trimming, and the effects on response times, are shown in Table G.3.

The user should, where possible, start the DUA at login time: initialisation rather than binding is the major start-up overhead.

We will be able to see the effect of some of these steps in an experiment described in Section G.11. In the next section, we start examining the response times of various read and search operations.

G.4 Some Basic Search Filters and Read Operations

The purpose of this section is to compare the speed of read operations and search operations with some basic filters. The main aim is to assess whether there are substantial speed differences for these operations, and whether these differences are consistent for the various implementations. DUA designers should benefit from an understanding of such differences, if they do indeed exist.

Other authors present a mixed picture. Afifi and Huitema assert that for the Pizarro implementation read is about twice as fast as search, without making it clear whether this difference is measured at the DUA or at the DSA, whether the timings include communication times, and whether they include bind times [AH92]. The evidence about the EAN system is contradictory: a read for a single entry is faster than a search for the same entry if the entry is in the directory; if there is no entry with the given name, then search is faster than read. Howes does not compare read and search directly when describing the University of Michigan DSA, but reports a search for one entry taking only 32 milliseconds measured at the DUA [How95a]; if read is faster, it cannot be that much faster. The DEC implementation, which is faster still, has very similar response times for read and search, so long as the searches are indexed: DEC quote read and search times of approximately 7 milliseconds timed at the DUA [Eme95]. Non-indexed searches are slower, with speed deteriorating markedly as database size increases.

The response times of eight different DUA-DSA combinations are summarised in Table G.4. In two cases, the tests were local to UCL. In two cases (INRIA and BIRM), the tests were made with a direct connection to a remote DSA. In the other four cases, the operations were chained by the access point DSA to the appropriate target DSA.

Looking at the results, we see that read operations are generally faster than search operations, although the degree of difference is small when the start-up times (including bind to the local DSA) are included. Exact, approximate and initial*surname matching were roughly the same speed.

There is a muddled picture for any-substring matching. In some cases, any-substring matching was similar in speed to other types of matching, in one case it was very much slower. The difficulty of providing indexing support for this type of operation makes it likely that this type of matching will be slower.
APPENDIX G. RESPONSE TIMES

<table>
<thead>
<tr>
<th>DUA-ORG-ATT</th>
<th>Read</th>
<th>Search filter</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Exact</td>
</tr>
<tr>
<td>UCLb-UCL-CN (DIRECT)</td>
<td>4.2</td>
<td>5.4</td>
</tr>
<tr>
<td>UCLb-UCL-SN (DIRECT)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UCLb-BATH-CN</td>
<td>4.5</td>
<td>5.7</td>
</tr>
<tr>
<td>UCLb-INRIA-CN (DIRECT)</td>
<td>9.7</td>
<td>9.3</td>
</tr>
<tr>
<td>UCLg-BIRM-CN (DIRECT)</td>
<td>5.4</td>
<td>7.5</td>
</tr>
<tr>
<td>ULCC-MICH-CN</td>
<td>1.3</td>
<td>2.0</td>
</tr>
<tr>
<td>ULCC-BATH-CN</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>ULCC-YNET-CN</td>
<td>7.5</td>
<td>27.5</td>
</tr>
</tbody>
</table>

Table G.4: Comparing the speed (in seconds) of read operations and searches with basic filters

An overall summary of the results in Table G.4 is that the differences in speed between read and search and the different search filters for a given DUA-DSA configuration are smaller than the differences between the various DUA-DSA configurations. The fastest set-ups tended to be best whatever the operation, and the slower set-ups consistently poorer. The variety of implementations under test and different DUA set-ups means that there is no simple relationship between database size and response times. In fact, the second fastest searches were on by far the biggest database (Michigan) and the slowest times on the smallest database (Y-NET).

G.5 Comparing the Speed of the First and Subsequent Operations on a Connection

There are several reasons why we might expect a follow-up operation on a DSA to be faster than the initial operation. If follow-up operations are faster, it means that a querying strategy based on multiple operations may be almost as quick as one based on single operations. This issue is not addressed in any of the other papers which discuss X.500 response times.

Let us review why we might expect follow-up operations to be faster. First, the initial operation has to bear the cost of start-up and binding to the local access point DSA. Furthermore, if the operation is for data on a remote DSA, either the DUA (in the case of referral) or the local DSA (in the case of chaining) has to bind to the target DSA. In fact, the process of locating the target DSA requires additional iterations if the local DSA does not have knowledge of which DSA masters the target information.

Another reason why a follow-up operation may be faster than an initial operation is operating system paging. The access point DSA and/or the remote DSA may need their code and/or their database to be paged into memory when a request arrives.
I examined this issue using tests from the UCL and ULCC DUAs to four DSAs. The tests use read operations and search operations with a variety of filters. The tests compare the speed of an operation performed once with the speed of the operation performed twice. Although this is a rather artificial test, it removes the possibility of there being vastly different response times for two different operations: we will look at sequences of different operations in the next section. The bind times for the UCL and ULCC DUAs are also included as a benchmark. The results of these tests is shown in Table G.5.

<table>
<thead>
<tr>
<th>DUA-ORG-ATT-MATCH</th>
<th>Local bind</th>
<th>Perform operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCLb-UCL-Read (DIRECT)</td>
<td>3.5</td>
<td>4.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.3</td>
</tr>
<tr>
<td>UCLb-UCL-SN-APP (DIRECT)</td>
<td>3.5</td>
<td>5.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6.3</td>
</tr>
<tr>
<td>UCLb-BATH-SN-ANYSUB</td>
<td>3.5</td>
<td>5.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.8</td>
</tr>
<tr>
<td>ULCC-BATH-SN-EX</td>
<td>0.6</td>
<td>1.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.2</td>
</tr>
<tr>
<td>ULCC-BIRM-CN-EX</td>
<td>0.6</td>
<td>2.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.1</td>
</tr>
<tr>
<td>ULCC-BRIG-CN-EX</td>
<td>0.6</td>
<td>1.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.7</td>
</tr>
</tbody>
</table>

Table G.5: Comparing the speed of the first and subsequent operations on a connection

The evidence is unequivocal: the follow-up operations were much faster than the initial operations. In five of the six cases, the follow-up operation took half a second or less. We will examine the cost of sequences of operations, as well as complex search filters, in the next section.

G.6 Complex Search Filters and Sequences of Searches

The evidence from earlier chapters on query data, directory data and the problem of matching shows that search strategies based on a single filter item are not very effective. Such strategies fall substantially short of what can be achieved by using more than one type of matching, or by trying to match on more than one attribute. There is little in the literature on this subject. The description of the DEC system includes data on four component AND filters, but nothing on OR filters or sequences of operations [Eme95]. In this section we evaluate the response times of three multiple filter item querying strategies. These are:

**Sequential searches:** the approach used by DE, where the type of matching is relaxed from exact to any-substring to approximate until one of the searches returns the required result.

The filters used are:

\[
\text{cn=foo, cn=*foo*, cn~=foo}
\]

**Boolean filters:** where the filter items are ORed together in a single search operation. Two specific filters are assessed. The first filter is based on the filters in the previous example.
APPENDIX G. RESPONSE TIMES

(cn=foo) OR (cn=*foo*) OR (cn'=foo)

The second filter is used by the UFN algorithm.

(cn'=foo) OR (sn'=foo) OR (uid'=foo) OR (cn=*foo*) OR
(sn=*foo*) OR (uid=*foo*)

The comparisons are not fully realistic in two respects. First, the names used in the filters were chosen so that only one result was returned for any search. In practice, we have seen that approximate matching is likely to return more matches than exact or substring matching – this is still true even if we assume that the standard Soundex algorithm is implemented, rather than Quipu's less discriminating version. We will see in Section G.10 that the number of results returned has a substantial impact on speed.

Second, the DE sequence of searches strategy often only needs a single exact match filter to find the required result; all three filters are only necessary if the name cannot be matched using exact or substring matching. I found in Chapter 5 that for the UCL queries and data, DE on average used 1.47 searches per person name query. The results of the sequence of search tests are thus a pessimistic estimate of the response time of the DE approach.

The results are presented in Table G.6; timings for exact matching are included as a benchmark for the other search strategies.

<table>
<thead>
<tr>
<th>DUA-ORG-ATT</th>
<th>Search filter / strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Exact</td>
</tr>
<tr>
<td>UCLb-UCL-CN (DIRECT)</td>
<td>5.4</td>
</tr>
<tr>
<td>UCLb-BATH-CN</td>
<td>5.0</td>
</tr>
<tr>
<td>UCLb-INRIA-CN (DIRECT)</td>
<td>9.3</td>
</tr>
<tr>
<td>UCLy-INRIA-CN (DIRECT)</td>
<td>12.3</td>
</tr>
<tr>
<td>UCLb-YNET-CN</td>
<td>11.5</td>
</tr>
<tr>
<td>UCLd-HOLL-CN (DIRECT)</td>
<td>6.9</td>
</tr>
<tr>
<td>UCLp-OPEN-CN</td>
<td>6.4</td>
</tr>
<tr>
<td>ULCC-BATH-CN</td>
<td>1.0</td>
</tr>
<tr>
<td>ULCC-MICH-CN</td>
<td>1.3</td>
</tr>
<tr>
<td>ULCC-YNET-CN</td>
<td>7.5</td>
</tr>
</tbody>
</table>

Table G.6: Comparing the speed of different search strategies with multiple filter items

No clear picture emerges from the tests. In some cases, the sequence of filters is appreciably faster than the ORed Boolean filter, in other cases about the same, in other cases slower. There are four cases where there is a substantial difference. The two YNET cases are both several seconds faster for the Boolean OR filter. However, both sets of YNET response times are very slow,
probably because of the lack of indexing for any-substring matching that we noted in Section G.4. The slowness amplifies the difference between the response times of the two filters.

The other two cases where there is a substantial difference in speed between the two strategies are for the UCL and OPEN DSAs, where the sequence of searches is much faster. It appears that Quipu is slower evaluating the ORed filter, at least one containing a non-indexed component (the any-substring filter item). This tendency can also be seen in the results for the BATH DSA; the disparity is presumably less because the BATH DSA holds far fewer entries - approximately 1600 compared with UCL's 16000+ and OPEN's 4500.

The UFN filter is slightly slower than the other Boolean OR filter; this accords with expectation since the UFN filter is simply a Boolean OR filter with more components to evaluate.

My conclusion is that these results do not show a clear preference on response time grounds for any of these strategies. The outcome seems to depend on which implementation is used and the size of the database.

G.7 Search Filters Including Object Classes

The directory can potentially hold entries for many different types of object. However, a white pages DUA is only interested in certain types of entries within an organisation. A DUA can specify which entries it is interested in searching by using object classes in filters. Does specifying the object class(es) of interest in the search filter have much impact on response times?

I tested two different additional filter components that could be used to select entries of the required class. In each case the additional component is ANDed to the original filter. I tested the additional filter components with several filters that we have examined in earlier sections.

Object class "person": This is the normal way of ensuring that entries returned are person entries.

(existingFilter) AND (objectClass=person)

Surname present: Colin Robbins, the principal implementor of Quipu, suggested to me when I was writing DE that an alternative way of testing for person entries was to test for the presence of the surname attribute, as he believed that testing for the presence of an attribute was likely to be faster than testing for an object class value. Note that this technique only works if person entries are the only entries to include the surname attribute.

(existingFilter) AND (sn=*)

The response time of the original filter without the object class filter item is given in each case as a benchmark.

The bad news first: including object class filter items in searches of the Y-NET DSA approximately doubled response times. This is a further example of the Y-NET DSA's response times being inadequate for service use.
APPENDIX G. RESPONSE TIMES

### Table G.7: Comparing the speed of search filters with and without object class filter items

<table>
<thead>
<tr>
<th>DUA-ORG-ATT-FILT</th>
<th>Without person OC</th>
<th>Including person OC</th>
<th>Surname present</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCLb-BATH-CN-MULTI</td>
<td>8.1</td>
<td>8.5</td>
<td></td>
</tr>
<tr>
<td>UCLd-HOLL-CN-SEQ (DIRECT)</td>
<td>7.4</td>
<td>7.2</td>
<td>7.8</td>
</tr>
<tr>
<td>ULCC-BATH-CN-EX</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>ULCC-MICH-CN-EX</td>
<td>1.3</td>
<td>1.9</td>
<td>1.4</td>
</tr>
<tr>
<td>ULCC-YNET-CN-SEQ</td>
<td>32.2</td>
<td>59.3</td>
<td></td>
</tr>
<tr>
<td>ULCC-YNET-CN-MULTI</td>
<td>25.6</td>
<td>57.0</td>
<td></td>
</tr>
</tbody>
</table>

However, the use of object classes in filters for the Bath, Royal Holloway and Michigan DSAs had little or no effect on resolution times. The technique of testing for the presence of the surname attribute was faster in one case, slower in another and the same speed as testing for the object class in the third.

### G.8 The Size of Result Sets

So far our experiments have been somewhat unrealistic in that they have been constructed to only return a single result. In this section, we examine the impact on response times of returning larger result sets. The figures in Table G.8 show the response times of four systems, each for four different sizes of result set. In each case, the DUA requested that four attributes be returned: commonName, surname, telephoneNumber and rfc822Mailbox.

<table>
<thead>
<tr>
<th>DUA-ORG-ATT</th>
<th>Number of entries returned</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>UCLg-BIRM-SN (DIRECT)</td>
<td>6.1</td>
</tr>
<tr>
<td>UCLf-BRIG-SN</td>
<td>5.3</td>
</tr>
<tr>
<td>UCLp-OPEN-SN</td>
<td>5.9</td>
</tr>
<tr>
<td>UCLs-SALF-SN</td>
<td>1.9</td>
</tr>
<tr>
<td>average</td>
<td>4.8</td>
</tr>
</tbody>
</table>

Table G.8: The response time in seconds varying according to the number of entries returned

The tests show that there is a significant cost to returning large result sets. For result sets of up to 25 entries for the DUA-DSA pairs tested, it appears as though each extra result adds about 0.1 second in total processing time. Interestingly, the overhead of extra results was less when the DUA was directly connected to the DSA. Evidently, and this was confirmed by a further simple experiment, some of the cost of handling larger result sets is due to the processing of intermediate
The default mode of distributed operation for Quipu systems is for the local DSA to chain requests to remote DSAs, rather than to pass referrals to the DUA: the strategy is described fully in [KR88] and [BR89]. One advantage of this approach for a DUA is that the DUA only has to handle a single connection to the directory; it does not have to implement code to handle multiple associations, and does not have to worry about looping if more than one referral is received while trying to resolve an operation. Another possible advantage of chaining is that the local DSA might already have a connection open to the remote DSA; it might have just retrieved information from that remote DSA for another user. Do there seem to be any response time advantages to either approach?

<table>
<thead>
<tr>
<th>Experimental Data</th>
<th>Chaining</th>
<th>Referral</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCLp-OPEN-CN-EX</td>
<td>6.4</td>
<td>6.4</td>
</tr>
<tr>
<td>ULCC-BATH-CN-EX</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>ULCC-BATH-CN-EX*11</td>
<td>2.4</td>
<td>2.2</td>
</tr>
</tbody>
</table>

Table G.9: Chaining compared with Referral

The results in Table G.9 show that there is little difference in speed between chaining and referral for an operation with a result set of one entry. The 10th-90th percentile ranges were also similar for all three sets of test data. However, our findings from the previous section suggest that referral may be quicker for larger result sets, as this avoids the result handling overhead imposed by intermediate DSAs.

Benford, however, notes that DUA referral may be inefficient if the user is authenticated as each DSA that is contacted by the DUA will have to connect back to the user's DSA to verify the user's password [Ben88]. I did a further set of experiments connecting a UCL DUA to the Open University DSA using simple authentication on the binds to assess the size of this overhead. The results were 5.6 and 6.4 seconds for a \textit{cnExact} search using chaining and referral respectively, an overhead of less than a second. However, this overhead would be larger if more than one remote DSA had to be contacted to resolve the query.

G.10 Results from Use of DE Service at ULCC

A limitation of the work described in this appendix so far is that the data was gathered against relatively few DSAs; it is possible that these DSAs are not typical of DSAs in general. I was able to partially redress this problem by gathering some data on the response times of the directory.
service run by the PARADISE project at ULCC. I did this by adding some code to DE to measure the time taken to do its various look-up operations. Whereas the aim of the work described earlier in this appendix is to compare the response times of various search techniques, the aim of the work described in this section was to get broadly based, accurate measurements of DE's search strategy. The data provides further insights into some of the issues we have discussed earlier in this appendix.

I gathered data for just over three days service usage: the PARADISE project's DE received over 4000 connections in this period. The response time data falls, with a few exceptions, into two categories:

- queries of the local database for country and organisation entries;
- queries for department and person entries in remote DSAs.

Of the two categories, the queries for country and organisation entries are the less interesting. These look-ups could be resolved within the access point DSA: it was the policy of the PARADISE DSA to hold copies of all country and organisation entries in the top two levels of the DIT. As a consequence, the look-up times for country and organisation entries were all fast, and negligible compared with the response times for the person entry look-ups described in previous sections. They ranged from 23 milliseconds for reading a country entry, to 44 milliseconds for exact matching a country entry, to 72 milliseconds for approximately matching an organisation entry. Such differences are barely perceptible to a human user of the directory.

The data for resolving department and person name queries is more interesting as the searches have to be resolved by a remote DSA. Tables G.10 and G.11 show how long it took to resolve department name and person name look-ups using the DE sequence of searches strategy.

<table>
<thead>
<tr>
<th>No. of searches</th>
<th>Sample size</th>
<th>Median time in msecs</th>
<th>Percentiles</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>10th</td>
</tr>
<tr>
<td>1</td>
<td>103</td>
<td>633</td>
<td>158</td>
</tr>
<tr>
<td>2</td>
<td>72</td>
<td>817</td>
<td>326</td>
</tr>
<tr>
<td>3</td>
<td>95</td>
<td>917</td>
<td>456</td>
</tr>
<tr>
<td>4</td>
<td>38</td>
<td>1885</td>
<td>578</td>
</tr>
<tr>
<td>all</td>
<td>308</td>
<td>901</td>
<td>279</td>
</tr>
</tbody>
</table>

Table G.10: Query resolution times, categorised by the number of search operations required by DE to resolve a department name query

The median query resolution time for department name queries is just under a second, one second faster than for person name queries. Resolution times increased steadily as more filters were tried, but person name look-ups took a median of between three and four seconds even
<table>
<thead>
<tr>
<th>No. of searches</th>
<th>Sample size</th>
<th>Median time in msecs</th>
<th>Percentiles</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>288</td>
<td>1311</td>
<td>343 31621</td>
</tr>
<tr>
<td>2</td>
<td>94</td>
<td>2345</td>
<td>437 24572</td>
</tr>
<tr>
<td>3</td>
<td>125</td>
<td>3528</td>
<td>737 34756</td>
</tr>
<tr>
<td>4</td>
<td>23</td>
<td>3158</td>
<td>791 34973</td>
</tr>
<tr>
<td>all</td>
<td>530</td>
<td>1971</td>
<td>400 31621</td>
</tr>
</tbody>
</table>

Table G.11: Query resolution times, categorised by the number of search operations required by DE to resolve a person name query when three or four searches were required. However, resolution times are heavily skewed, with the worst 10% of person name queries taking upwards of 30 seconds.

G.11 DAP vs LDAP experiment

DE can be compiled to use either the ISODE DAP libraries or the University of Michigan LDAP libraries. I conducted a test to compare the response time of DE for a typical query, using the two different versions of DE. The experiment is interesting as the LDAP DUA and other LDAP system components implement some of the features we cited as being desirable in Section G.3.

Figure G.2: The components used in the DAP versus LDAP test

Figure G.2 illustrates the experimental set-up for the two DUAs. The test was to find an entry for a colleague at the Open University, and to retrieve his telephone number. The query required a bind and three exact match search operations to be resolved: one on the country name, one on the organisation name and one for the person's surname.

The DAP and LDAP DUAs were both run from my workstation. The binaries were built...
in and run from the same directory in my NFS mounted file-store. Both versions of DE used the same two DSAs to resolve the query: the ULCC DSA "{cn=OscelIated Turkey}" and the Open University DSA "{c=GB, cn=Muscovy Duck}". In addition, the LDAP DE first has to communicate with an *ldapd* server; this translates LDAP into DAP queries. I chose the *ldapd* running on the same machine as the ULCC DSA. Thus, LDAP DE queries bear an additional overhead compared to DAP DE queries; LDAP queries must be translated into DAP queries and DAP results must be translated back into LDAP results. There is also a small inter-process communication overhead as the *ldapd* process communicates with the ULCC DSA, although this should be negligible as the communication uses a TCP loop-back connection.

<table>
<thead>
<tr>
<th>Underlying stack</th>
<th>Size of binary in bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAP</td>
<td>1318912</td>
</tr>
<tr>
<td>LDAP</td>
<td>221184</td>
</tr>
</tbody>
</table>

Table G.12: The size of the DE binary built with DAP and LDAP libraries

However, the LDAP configuration has some compensating advantages. First, the LDAP binary is a sixth the size of the DAP binary; the sizes are given in Table G.12. We would expect the LDAP binary to be quicker to load, and to require less paging when in operation. Second, an LDAP DE reads a single DE-specific configuration file, whereas an ISODE DAP DE additionally has to read up to twelve ISODE configuration files. Third, although the *ldapd* process has to read the ISODE configuration files, it can do this at start-up time. When an LDAP DUA makes a bind request, the *ldapd* simply has to bind to the directory. Fourth, the *ldapd* ISODE configuration files are all on local disk, whereas the DAP DE ISODE configuration files are on NFS-mounted file-store.

<table>
<thead>
<tr>
<th>DE configuration</th>
<th>Page faults</th>
<th>I/O operations</th>
<th>Query resolution time in secs</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAP</td>
<td>150</td>
<td>18</td>
<td>9.5</td>
</tr>
<tr>
<td>LDAP</td>
<td>29</td>
<td>4</td>
<td>3.8</td>
</tr>
</tbody>
</table>

Table G.13: A comparison of DAP and LDAP queries using DE

The results of the experiments show a speed advantage of nearly six seconds for the LDAP version of DE. It is clear that protocol conversion is very fast. In fact, Howes has shown in an environment where communication costs are minimised – by co-locating the *ldapd* daemon and the DSA – that the *ldapd* daemon adds very little overhead to bind and search operations [How95a]. Using his figures I would expect the total protocol translation overhead to be less than a tenth
of a second for the bind and three searches.

A likely explanation for some of the speed difference is the number of page faults. The larger binary had over 120 extra page faults during its execution; it also had more I/O operations presumably caused by reading more configuration files.

Another difference between the DAP and LDAP experimental set-ups is the usage of the UCL-ULCC link. The DAP DE uses more bandwidth on this link for two reasons. First, the LDAP client needs only four network packets to bind to ldapd, whereas a DAP client needs six network packets to bind to a DSA [You95]. Second, Howes has shown that LDAP encodings are smaller than DAP encodings; from his figures I estimate that the LDAP bind and searches took 500-600 bytes fewer to encode [How95a]. If bandwidth on this link was limited, the LDAP DE would suffer less than the DAP DE. However, my subjective impression is that the link was always quite fast and cannot explain much of the speed difference.

My conclusion, supported by evidence in previous sections, is that the speed difference is due to a combination of binary size, faster initialisation procedures and use of local file-store.

G.12 Conclusions

In this section, we summarise the results and attempt to draw some conclusions on directory service response times.

A prominent feature of the results is the huge range of response times for the different DUA-DSA combinations. The response times for one of the complex Boolean filters include a low of one second and a high of 57 seconds. Despite the range, the vast majority of response times were below 10 seconds.

I started the work described in this appendix believing that OSI connection establishment was the main performance problem of the PARADISE directory service. I was wrong. The biggest problem is starting and initialising the DUA to the point where the DUA is ready to bind to the directory. I have described several pragmatic solutions to this problem, and been able to demonstrate their efficacy by using an LDAP based version of DE. The key elements of the solution are to minimise the amount of file reading at initialisation, to read configuration information from local file-stores rather than NFS, to start the DUA in the background before it is required, and to use small DUA binaries. This last element can be achieved by splitting the DUA into a small user interface on the user's machine and a protocol engine on a directory server machine, the two parts communicating via a lightweight protocol: the University of Michigan's LDAP release uses this architecture to access X.500 directories.

Unauthenticated binding to the directory is pretty fast. The experiments in Section G.3 show that DUA-DSA bind times are negligible.

The operation response times described in Sections G.4 to G.7 do not show clear differences in response times according to the operation and/or search filter.
The main determinant of the response time for any given test seems to be the DUA-DSA pair, rather than the operation/filter under test. The choice of operation or search filter usually has relatively little impact on response times. Furthermore, there is often contradictory information about which operations/filters are fastest, even when two DSAs use the same implementation.

Some differences do emerge between operation speeds. Read operations averaged just over a second quicker than subtree exact match search operations. Section G.5 shows that the response time cost for sending two successive querying operations to a DSA is much less than twice the cost of a single operation. These figures are confirmed by analysis of the DE algorithm in Section G.10: response times do not increase linearly as the second, third and fourth operations are tried in attempting to resolve a query.

The evidence is mixed as to whether the sequence of searches strategy is generally faster than the alternative multiple strategy of combining all the filter items in a single Boolean OR filter. Since there are no clear response time advantages to either of these search strategies, it means that the preferred strategy should be selected on grounds of matching characteristics alone.

Most of the tests requiring access to data in remote DSAs had the queries chained by the access point DSA to the remote DSA. I compared this strategy with referral to see if there were response time grounds for preferring either strategy. Tests showed that response times were almost identical when binding was unauthenticated, but that referral was almost a second slower when using simple authentication. On the other hand, referral is faster for large result sets.

It is ironic that while DSAs invariably attract attention to improve response times, and the literature concentrates on this aspect of system performance, improving DUAs and their configuration may have a greater impact in practice on directory response times.
Bibliography


BIBLIOGRAPHY


[FMN90a] A Findlay, D Mahl, and S Nahajski. Design document: Xdir - X.500 directory user agent. Copy of document provided by X.500 team at Brunel University - contact x500@brunel.ac.uk, June 1990.


BIBLIOGRAPHY


BIBLIOGRAPHY


[WG93] L. Ward and C. Gedney. The CCSO nameserver (PH), 1993. Published as part of the documentation set available with the University of Illinois at Urbana-Champaign's release of the CCSO software.


