

**Image Retrieval Using
Natural Language and
Content-Based Techniques**

Kate Byrne

Master of Science
School of Informatics
University of Edinburgh

2003

Abstract

This MSc thesis deals with the application of Natural Language Processing and Content-Based Image Retrieval to the practical problem of finding database images to answer user queries. The data collection used contains about 50,000 digital images linked to a large text database. In contrast to many otherwise similar image retrieval projects, the text used comes from a range of data fields, not just captions individually linked to images. The data relates to archaeology and architectural history, and contains a lot of domain-specific language.

A number of different techniques are explored through building a retrieval application, named CANTRIP, that explores different methods in combination and in contrasted groups to see their effects. In particular, the aim was to examine the combination of traditional IR techniques with Named Entity Recognition, and then with CBIR and relational database tools. The results show that some methods from different fields combined well and others did not. Integrating normal database joins with external inverted text indexes built on NE-marked text worked well. Combining these techniques with CBIR ones, that look at the physical characteristics of images in terms of colour, shape and so forth, was less successful; although circumstances were identified in which it is useful.

Acknowledgements

A lot of people helped me in various ways with this project, and I'm grateful to all of them. My supervisor, Ewan Klein, provided direction and encouragement throughout, and Claire Grover supported me when he was away. Iadh Ounis and Keith van Rijsbergen of Glasgow University gave very useful advice and pointers on the CBIR and IR work. I'm grateful to James Z Wang of Penn State University for allowing his SIMPLicity software to be used in the project. Several others in the Language Technology Group helped me with discussions and software, in particular James Curran and Yuval Krymolowski.

The data used for the project belongs to the Royal Commission on the Ancient and Historical Monuments of Scotland, and I'm very grateful to my friends and former colleagues there who let me use it, and gave up time for meetings and advice along the way. Clare Sorensen was my chief contact throughout and nagged others on my behalf, but I also want to acknowledge the help of Christine Allan, Peter McKeague, Diana Murray, and all those who sent sample queries for the evaluation. Finally, special thanks to Clare Sorensen, Simon Gilmour and my long-suffering husband Peter Williams, who sat in front of the screen for hours doing the evaluation exercise.

This is also an opportunity to say what a wonderful year it's been, doing this MSc course in language engineering in the School of Informatics. I didn't know it was possible to learn so much, from such a basis of ignorance, in so short a time.

Data and images from the NMRS database are reproduced with the permission of RCAHMS.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(Kate Byrne)

Table of Contents

1	Introduction	1
2	The Nature of the Problem	5
2.1	The Data	5
2.2	The Task and its relevance	7
2.3	Approaches considered	9
3	Related Work	13
3.1	Content-based image retrieval (CBIR)	13
3.2	Text-based image retrieval	14
3.3	Information retrieval (IR)	16
3.4	Question-answering	17
3.5	NER	18
4	Tools and Methods	21
4.1	Database work	21
4.2	Processing the image collection	26
4.3	CBIR work	27

4.4	IR tools	29
4.5	NER and term weighting	31
5	Implementation	37
5.1	Interface design	38
5.2	Version 1: Baseline	41
5.3	Version 2: “Fulltext”	42
5.4	Version 3: Search engine	44
5.5	Version 4: NER	45
5.6	Version 5: Enhanced NER	46
5.7	Version 6: CBIR	48
6	Evaluation	49
6.1	Methodology	49
6.2	Results	51
6.3	Analysis	54
6.4	Contribution of NER	58
7	Scope for Further Work	61
7.1	NE recognition and IDF weighting	61
7.2	Database related issues	62
7.3	CBIR, design, and practicalities	64
8	Conclusions	65
A	Abbreviations used	67

B Database setup, SQL code	69
B.1 Main database tables, with indexes	69
B.2 Location lookup tables	72
B.3 CANTRIP table and index creation	73
C CANTRIP application, sample PHP code	75
C.1 Baseline version	75
C.2 CBIR version	80
D Java programs, for search engine	85
D.1 DocLength.java: calculating “docLength” factor	85
D.2 Loader.java: Adding keyword weights	88
D.3 NEMarker.java: Finding NEs in input text	91
D.4 DbPrepper.java: Marking-up NEs in database text	93
E Sample from generated gazetteer file	97
F Notes for CANTRIP evaluators	101
G Sample evaluation form	103
H Evaluation Results	107
Bibliography	113

List of Figures

2.1	The Lodberrie, Lerwick — View from above	8
4.1	Subset of NMRS Database	22
5.1	CANTRIP application	38
5.2	CANTRIP query interface	39
5.3	CANTRIP results screen	40
6.1	CANTRIP — evaluation version	50

List of Tables

2.1	Sample entry from NMRS database	7
4.1	Functions of core database tables	23
5.1	Query items and corresponding database fields	42
6.1	Overall results summary	53
6.2	Inter-evaluator agreement	54
6.3	Results for non-locational queries	55

Chapter 1

Introduction

This dissertation describes a project on image retrieval, carried out over about three months. The problem to be solved was to return a relevant set of images from a large collection, in response to a user query. This is a practical issue faced by, amongst others, the holders of publicly funded archives, who need to make their material easily accessible over the Internet. The interest of the problem stems from the number of different approaches possible, using tools ranging from physical image content matching to analysis of the descriptive text.

The scope and objectives developed as time went on and the possibilities became clearer. In the event, six separate versions of the retrieval system were built, to compare different approaches to the problem.

The image collection used consisted of around 50,000 digitised images from the archive collection of the National Monuments Record of Scotland (NMRS), held by the Royal Commission on the Ancient and Historical Monuments of Scotland (RCAHMS), a public body based in Edinburgh. The images are linked to a relational database of text describing archaeological sites and historic buildings across Scotland, and this textual data was also made available for the project.

There are two basic categories of image retrieval solutions: those based on the text that describes the picture, and those that compare the actual content of the image — its

colours and shapes and so forth. A key objective here was to try to combine these approaches, to see if content-based image retrieval (CBIR) could be improved by working with the text, and *vice versa*. One of the first issues to be dealt with therefore was to find an effective mechanism for combining these techniques, as a text-based approach obviously starts with a text query from the user, whereas CBIR systems start with a sample image to match.

For text queries, there are a great many well-developed tools available. The areas to draw upon include: database technology, question-answering research, and information retrieval (IR); as well as many specific techniques from the natural language processing (NLP) field. The areas considered and researched are described in Chapter 3. Those employed in the final system were principally IR and NER (named entity recognition), with some ideas drawn from question-answering. Naturally CBIR techniques were also explored, but the focus of this project was on shallow processing language tools.

With a wide-ranging subject like this, impinging on several different fields, there is inevitably something of a conflict between depth-first and breadth-first, when searching the possible solution space. It was decided early on to employ a bias towards a broad-spectrum strategy: trying to combine different tools in a layered approach, to see how they interact and help or hinder each other. This meant that very often particular avenues could not be followed as far as might have been interesting. To continue the metaphor: it would have been impractical to attempt an exhaustive search of the space in the time available. Areas where more investigation would be worthwhile are discussed in Chapter 7.

Chapter 2 describes the task in more detail, looking at aspects that were peculiar to this dataset and at the things that are generally applicable to any work in the same field. It also outlines the approaches considered and chosen. A review of relevant literature follows in Chapter 3. Chapter 4 contains an overview of the practical work done, in terms of the different tools used. Chapter 5 then describes the software package that was built, and which particular ideas each version was intended to test. Samples of the SQL, PHP and Java code are in Appendixes B, C and D. The complete set of programs

is available, but was too bulky to print in full. The evaluation exercise is described and the results analysed in Chapter 6. A full listing of the raw data for the results is given in Appendix H. As mentioned above, Chapter 7 looks at areas where more work could usefully be done, and finally Chapter 8 summarises the conclusions reached. Appendix A contains a list of acronyms used in the text.

Chapter 2

The Nature of the Problem

2.1 The Data

As mentioned in Chapter 1, the dataset used was a subset of the NMRS database. RCAHMS maintain the NMRS database in the Oracle RDBMS, in a set of applications developed in-house. This data was thought particularly suitable for the project for a number of reasons. It has been developed over many years: originally as hand-written records dating from 1908, when the work of systematically recording the buildings and archaeology of Scotland began. At various times the structure has been revised and augmented, and the volume of material has grown steadily. Little has been removed, because the data has historical interest even if theories or recording methods have changed. The Record covers around 220,000 historical sites and buildings in Scotland, with an associated collection of archive material containing about a million items so far.

In the early 1980s the data was computerised, being captured largely by OCR scanning from index cards and paper documents, with all the attendant data errors that brings (though naturally it was cleaned as far as possible). The first application used the IBM STAIRS package, an advanced IR system for the time, but one in which updating the data was extremely awkward. Because the database had to fulfil the dual rôles

of answering user queries and coping with constant record updates, it was transferred into Oracle a year or so later. Previously unstructured text was split into fields at this point, but much remained in various “notes” fields, and the coverage across fields was necessarily sparse as different recording standards had been used over the years. The same core data structure has been in use for over 20 years now, with the usual additions and upgrades of any working system.

Because survey work involves taking photographs, drawing plans and maps and so on, the data incorporates text and graphics, with links to several GIS applications for geo-spatial queries and maintenance of map-based data. The result is an almost organic structure, with apparent surface regularity often masking the richness of content lurking beneath. The text is chock-full of domain specific terminology, and also shows the changes in language and style to be expected in records that span almost a century of recording work. From the point of view of testing the robustness and coverage of the tools, it is an ideal candidate for experiments with language processing techniques.

The images used in this project are that subset of the archive collection that has been translated to digital image files, and the text is the portion of the database describing them.

As noted, the data is sparse in many of the fields, and this had an important impact in one respect. The obvious way to do text-based image retrieval work is by using the image captions, each of which describes the picture it’s attached to. In the NMRS database the caption field is one of the sparse ones: *29% of the digital images have no caption at all*. Where captions are present, a great many are intended to be used in conjunction with the site record, e.g. 10% contain “view from” followed by a compass point, 2% simply note that they are “digital image of” another catalogued archive item, 3% are “general view” or “aerial view” often without further detail, and so on. They were not designed for use alone in retrieval and it was decided early on that this project would need to use the accompanying text from related tables. The sample database entry shown in Table 2.1 illustrates the point. The entry is a typical one, but it should be noted that the length of the free text varies greatly. The report field may be empty, contain a few lines as in this case, or be several pages long. The image that this entry

Description	Field name	Field contents
Site name	nMrsname	LERWICK, 20 COMMERCIAL STREET, THE LODBERRY
Alternative	altname	LERWICK HARBOUR; LODBERRIE
Classification	classgrp	COMMERCIAL; RESIDENTIAL
	class	WAREHOUSING AND STORAGE; COTTAGES
	classsub	COTTAGE; STORES; PIER; CRANE
Free text	report	HU44SE 22 47894 41244 (Location cited as HU 479 413). The Lodberrie, Lerwick, late 18th or early 19th centuries. An attractive compact group of dwelling houses and stores on a pier projecting into the harbour. There is a sea-door with a wall crane, on the sea-front; cf. Stromness, Orkney. J R Hume 1977.
Image caption	desc1	View from above.

Table 2.1: Sample entry from NMRS database

refers to is reproduced in Figure 2.1.

2.2 The Task and its relevance

The specific characteristics of the NMRS data have been stressed, but in many respects it is also typical. Most of our large public archive collections have been built up over many years, and are not quite as regular as one would perhaps like. At some periods there would have been funding available for special cataloguing projects and big advances were possible; at other times funds may have been scarce and the bare minimum was done. The professionals deciding the standards may have disagreed from time to time on recording methods and terminology. (I believe it can happen.) If computerised retrieval systems are to be truly useful they need to be able to cope with irregularities



Figure 2.1: The Lodberrie, Lerwick — View from above ©crown copyright RCAHMS

in the data, as this is the nature of real information. If collection material has to be forced into too tight strait-jackets, some of the less tangible content is simply lost.

Over the last decade or so, the holders of data collections - archives, galleries, museums and libraries — have been quite abruptly faced with demands to make *all* the material available on the Web, *immediately*. Such demands are natural and understandable, given the way the Internet has revolutionised information exchange. They are also, in general, precisely what the collection owners would wish. But, because the data is typically not in the form of rigid, precise data fields, nor on the other hand is it all in readable, coherently-flowing individual documents, the demands for access can be surprisingly difficult to meet. Part of the response has been a push to greater standardisation of terminology and classification, and more splitting of data into fields, so that standard query tools like SQL will succeed on the dataset. Secondly, there has been much work on the “readability” side in producing huge quantities of new text, designed for those creatures so favoured by politicians and grant-givers: “the intelli-

gent layman” and “the life-long learner”. My own interest in language engineering arose from the belief that there is a great deal of scope for a third area of activity, in developing computer tools that can deal intelligently with this type of collection data, extracting the particular nuggets the enquirer wants from the mass of material. This project is a very small exploration of that field, dealing with the practical problem of retrieving relevant images using some pretty heterogeneous text.

In recent years RCAHMS, along with most archive and recording organisations, has made scanning and digitisation of graphic collection material a priority. As the digital image collection grows, the next priority is to make it available over the Internet, so this project has some practical relevance in exploring techniques that may be useful in a production system. However, being a research project, it also had the happy freedom to pursue a range of ideas and investigate their usefulness in an open-minded way, without the constraint of its actually having to work.

Naturally the NMRS data structure had to be followed, and specific field names used in query commands. However, the general applicability of techniques was always a consideration. Some of the findings relate to combining search engine techniques that completely ignore the field structure, with relational queries that exploit it. I believe these methods could be applied to any text-oriented relational database.

2.3 Approaches considered

As mentioned in Chapter 1, one of the first decisions to make was what form of query to use. All the CBIR systems researched (see Veltkamp and Tanase (2000) for a comprehensive survey of current systems) work by matching a seed image provided by the user. However, it’s an approach that works best where the image collection can readily be divided into discrete categories, which also cover all the most frequent types of queries required. Neither of these conditions really holds for the NMRS collection.

There are techniques for clustering images either by their visual content (see Del Bimbo (1999)), or by their descriptive text (see e.g. McCallum (1996)). English Heritage’s

PastScape site (<http://www.english-heritage.org.uk/filestore/pastscape/>) shows a nice example of the sort of interface that could be implemented using the idea of exemplar images. It has a series of cartoon images as part of the query interface, each representing a category of site. Clearly in this case the representative image becomes just a surrogate for a text description such as “church”, “castle”, “stone circle”. This makes the point that whether one used representative cartoon images or picked a real seed image for each category, the results of using *just* a set of exemplar images would be a very restricted set of possible queries, that would always return the same very small subset of the collection. PastScape (which is in fact an entirely text-based search system) allows other search criteria to be entered, in words. Most of the CBIR systems looked at get round the difficulty another way, by allowing the user to pick a new seed image for each in a series of searches, trying to get ever closer to the goal.

This “exemplar image” approach was ruled out early, for two main reasons:

- The types of image in the NMRS collection are not ideal for CBIR-led processing. See Section 4.3 for further explanation, but the big difficulty is the high proportion of monochrome pictures.
- The sort of interface where the user has to pick the best image out of each set returned, in an effort to move closer to what’s actually wanted, would be very frustrating in this domain. (In fact, it’s very frustrating with any diverse image collection, in my experience.) The concept of exemplar images just doesn’t work for many sites; for example, one “castle” may have almost no visual similarity to another “castle”.¹

In any case, this project was supposed to be primarily about language processing! Therefore very early on in the project, the goal was defined as being to work from text-based queries, and use CBIR as a supporting tool where appropriate.

Having decided on a basically text oriented query system, there was a wide range of applicable techniques to choose from. The two main fields researched were question-answering and IR. The outline plan was to use inverted indexing of the free text —

¹There are counter-examples to this argument; see Section 6.3 for details.

which is well-proven and successful — and augment it with named entity recognition (NER), query expansion and CBIR. In particular, the intention was to examine the effect of building an inverted index not just on the raw text but on pre-tokenised entity strings within it. In order to compare the effects of various techniques, several different versions of the application were planned, using off-the-shelf tools and specially written software.

Chapter 3

Related Work

The project touched on a number of related but different fields. This chapter describes some of the relevant work in those areas.

3.1 Content-based image retrieval (CBIR)

A comprehensive technical overview of the state of the visual data retrieval art is given in Del Bimbo (1999). For this project it was only necessary to skim the surface of this subject, which covers video retrieval as well as flat 2-dimensional graphics, and includes automatic indexing and annotation methods. For the purposes of this work an understanding of the basic techniques was all that was needed. Veltkamp and Tanase (2000) gives a very useful overview of current CBIR applications, looking at 39 different systems in terms of the features used, indexing data structures, matching techniques and so forth. Two practical CBIR applications were experimented with — Xie (2001) and Wang et al. (2001) — and Section 4.3 gives more detail on how they were used here. Like all CBIR systems they work by identifying features of images, such as colour distribution, texture, and orientation of shape outlines, and producing metrics by which to compare these. The art is in combining the metrics in such a way as to be able to group visually similar images together.

The notion of “similarity” is hard to pin down of course. When we see images they often conjure up all sorts of associations that have nothing to do with the shape and position of the objects. The human eye and brain can use clues to determine scale that are exceedingly difficult to replicate in a computer, which may, for instance, think that a grey pencil standing on end is very similar to Nelson’s Column. One can think of any number of examples where two pictures may be similar in some ways but not in what we would consider the “obvious” way. The SIMPLIcity system (Wang et al. (2001)) tries to address this problem by using semantic classification methods to categorise images as “indoor”, “outdoor”, “city”, “landscape”, and even “benign”, “objectionable” and such like. As with so many computer tools (alas) one gets better results from it if one understands its underlying working. How it performs on the project data is discussed in Section 5.7.

3.2 Text-based image retrieval

A lot of work has been done on image retrieval by text caption rather than by image content. This task is distinguished from IR on the one hand and question-answering on the other partly by the length of the texts involved: image captions tend to be very short. Several researchers (see Guglielmo and Rowe (1996), Rose et al. (2000), Pastra et al. (2003)) have noted that although NLP has often not been able to improve on IR techniques, it does make a difference in this domain of short documents.

These three systems all use full POS tagging and parsing of captions and queries to produce logical forms. Each is slightly different but they bear similarities to Context Free Grammar (CGF) analysis into phrases with “agent-patient-theme” rôles identified. Guglielmo and Rowe (1996) normalise caption and query by replacing words with standard parent terms from a type hierarchy or thesaurus. Rose et al. (2000) and Pastra et al. (2003) also use a separate lexicon or gazetteer, but for identifying entities or domain-significant phrases in the text. All three find that detecting these multi-word units is helpful, and Guglielmo and Rowe (1996) particularly identify nominal compounds where word order changes meaning. An example from their domain of aircraft

and weapons pictures is “target tank”, which is a tank, and “tank target”, which is not. Simple IR keyword techniques will fail to discriminate between these.

Smeaton and Quigley (1996) also deal with image caption retrieval, but using quite different methods. They point out some more problems that will affect the standard IR “bag of words” approach: **polysemy** (different meanings for the same word), **synonyms** (different words with the same meaning) and **ambiguity** (multiple interpretations of words or phrases). Resolving these requires some extra information on the context, and query expansion is a common approach. They performed a series of experiments, starting with standard IDF matching (see Belew (2000) for a full treatment, and Section 4.4 for a brief description of this) and adding extra components (variations on query expansion and their word-to-word similarity metrics) one by one. This project was influenced by their approach, which seemed very clear and methodical.

Another paper that helped determine my methodology was Flank (1998), which deals with a topic very close to this project’s, and explains it in clear and simple terms. Here a series of NLP elements — tokenising, stemming, query expansion — were added in layers, to build a retrieval system that worked against multiple text databases.

Evaluation tends to be a problem for this type of system. As Sharon Flank points out, it’s particularly difficult to test recall on a real database because nobody knows exactly what’s in it, and hence what items were missed. Testing precision is also subjective because it depends on judgments of relevance of images; and one image can mean different things to different people.¹ The Guglielmo and Rowe (1996) system was compared against an existing system in use by the organisation, instead of attempting to give absolute figures for precision or recall. Evaluation of the ANVIL system (Rose et al. (2000)) was very complicated, but the authors admit there is a degree of subjectivity. The Pastra et al. (2003) system has not yet published evaluation results. In general, one can’t escape the feeling that the results produced depend very much on the design of the experiment, and techniques that work well in some setups are found by others to work badly in other circumstances.

¹The standard definitions of precision and recall are referred to: $precision = tp/(tp + fp)$ or the proportion of selected items that were right; and $recall = tp/(tp + fn)$ or the proportion of the whole target that was found. (tp is true positives, fp is false positives and fn is false negatives.)

3.3 Information retrieval (IR)

Traditional IR is a simple and successful retrieval technique, for which the main source used was Belew (2000). The theory behind TF-IDF weighting is outlined in Section 4.4.

A very interesting paper by two of the pioneers in the field is Lewis and Sparck Jones (1996). They define Information Retrieval as a global term covering “document retrieval” and “text retrieval”, and distinguish these from question answering (or “knowledge retrieval”) by the aim of the query. In IR applications the user is assumed to wish to read about a subject, rather than to receive a short answer to a specific question. The query is quite likely to be ambiguous and/or vague: after all, the user presumably doesn’t yet know very much about the topic. The authors discuss past approaches and suggest that using controlled vocabularies to classify queries (which is rather similar to what was done in Guglielmo and Rowe (1996) for instance) does not generally work well. Statistical term frequency approaches such as the inverted index *do* work well and have the advantage of not involving any prior assumptions about the data or the query types. They cite examples (albeit from some years ago) of syntactic rôle relations, such as were described in Section 3.2 above, producing only very small gains, when they might on theoretical grounds be expected to work much better. They suggest finding compound terms as a promising approach, but argue that they would need separate weighting because they will have lower frequencies than their component words alone. This paper was also a significant influence on my planning.

Another viewpoint that’s slightly different from the mainstream of NLP is given by Kruschwitz (2000), who deals with retrieval of Web pages by search engines. An interesting finding is that automatic query expansion, using related terms, worsens precision dramatically. Experiments on this are discussed in Section 5.6.

3.4 Question-answering

Question-answering is a very big field in its own right, and there is no intention to attempt a comprehensive review here. An excellent source of material is the TREC website, at <http://trec.nist.gov/>. The objective of question-answering is to retrieve a short and specific answer from the source material; typically answering factual queries like “What is the second-highest mountain in the world?” by extracting a snippet of text from one of the source documents.

It became clear during the research for this project that its topic overlaps with different retrieval fields. There are many similarities to the document retrieval work discussed above under the IR heading (Section 3.3), where the objective is to find documents about the query topic. The image caption retrieval work described before that (Section 3.2) is much the same but works with shorter documents. In some respects this project’s subject can also be thought of as question-answering. What is often required in image retrieval is a very small, precisely defined result set; and this is what distinguishes question-answering from more general document retrieval.

Having said this, I have to admit that after researching what seemed the most relevant aspects of QA methods I decided not to use them; not because they are not successful but because it was not possible to integrate them with the other experiments in the time available. Given unlimited time it would be interesting to see if techniques such as surface text patterns could be combined with IR and database techniques, but it was not practical to attempt it, and the methods are so different that it’s not obvious they would combine well. See Ravichandran and Hovy (2002); Greenwood and Gaizauskas (2003); Katz and Lin (2003); Soubbotin and Soubbotin (2003) for various approaches to essentially the same technique. This work is similar in many respects to the CFG-style relations discussed earlier, but the authors just cited go into much more detail on this specific issue and develop the techniques considerably.

3.5 NER

Named entity recognition (NER) is the process of finding content-bearing nouns and noun phrases within a text, such as the names of people, places and organisations. It is a technique from the Information Extraction (IE) field, which is related to but separate from IR. IE is concerned with mapping unstructured or semi-structured text into a structured format, often by identifying significant phrases in text and extracting them to fill a set template layout. Much work has been done on NER, using a combination of two main approaches: rule-based and statistical. See, for example, Bikel et al. (1997), Borthwick et al. (1998) and Mikheev et al. (1999) for different aspects of the subject. The CoNLL Web pages at <http://cnts.uia.ac.be/conll2002/ner/> and <http://cnts.uia.ac.be/conll2003/cfp.html> are also very good sources.

The rule-based method works by defining a set of rules or patterns to identify NERs. These patterns are based on the format of the words (for example, most NERs are capitalised in English text), or on relationships with the rest of the text (e.g. location names often follow prepositions like “in”, “at” or “near”). The NERs are found by working systematically through the text, applying the rules. The TTT package (see Grover et al. (1999)) includes a rule-based NER recogniser. This and other tools from the package were explored as part of the project.

The other approach is statistical, and works by building a model of the target, generally using features of the data similar to the rules just described, and then calculating the most probable alignment between the raw input data and this target. The model is produced from training data, which has all the NERs correctly marked in advance, and which is supposed to be representative of the larger set of unmarked raw data. If the annotated training data is truly representative (which generally means that there has to be a *lot* of it) then the model will be a good match to the raw data and the recogniser will be able to find almost all the entities within it. Named entity recognition has been the shared task at the CoNLL conference for the past two years, and the statistical recogniser this project intended to use (Curran and Clark (2003)) was developed for CoNLL-2003.

For this project NER was an important topic. In the experiments performed it was only necessary to identify the presence or absence of entities, but it is more usual in NER to categorise the entities as well. Thus the entity “John Smith” might be tagged as “person”, while “John Smith & Co” would be “organisation”. The scope for including NE categorisation in this image retrieval work is discussed in Chapter 7.

Chapter 4

Tools and Methods

This chapter outlines the main stages of the project and the tools employed. It is intended to give an overall picture of the work done, and to explain some of the planning decisions.

4.1 Database work

The dataset used was part of the NMRS database described in Chapter 2. The data was loaded into a MySQL database, and Appendix B lists the SQL used in setting this up. A database containing so many free text fields, which contain frequent newline characters, has very specific formatting problems to overcome, so this step was quite time-consuming.

The NMRS database schema contains around fifty tables, but for this project a set of just ten tables was copied, along with some lookup lists for location codes (regions, districts and so on). The ten core tables are related as shown in Figure 4.1. There was also a free-standing thesaurus table, not yet integrated into the NMRS database, but containing a hierarchy of standardised terms that the organisation is in the process of adopting for classifying sites and monuments. The thesaurus contains archaeological and industrial terms, but not architectural ones.

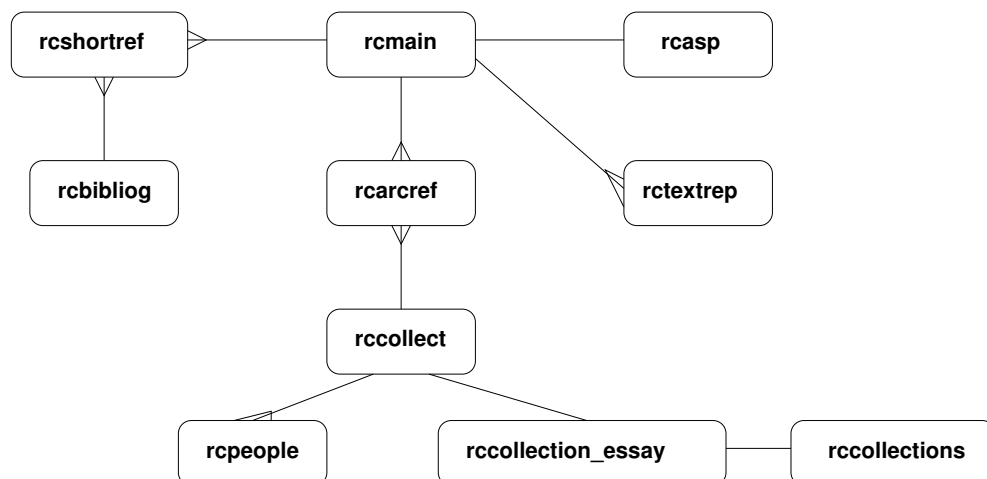


Figure 4.1: Subset of NMRS Database

Table 4.1 lists the functions of each of the core tables and the number of rows they contained. Since this project was only concerned with the digital image data, the number of records was subsequently greatly reduced. So far only around 60,000 items from the archive collection have been digitised, and only the records relating to them were used. There is an argument for retaining all the available text for training statistical recogniser tools, e.g. for NE recognition. From the performance point of view however, there is a big gain from removing the bulk of records that are not required. Roughly a quarter of the whole NMRS database was used for this project. The records selected were all the `rccollect` table entries for digital images, plus all records from `rcarcref`, `rcmain`, `rctextrep`, and `rcasp` that relate to them. Other tables were used for NE selection, explained in Section 4.5.

As Figure 4.1 suggests, the two most important tables are `rcmain` and `rccollect`. The Archaeology Record is completely site-based, i.e. the information always relates to a geographically located archaeological site. The NMRS database was originally designed for this data, so `rcmain` was the driving table with one record per site, and all the other tables hung off it. For example, there is one record in `rcmain` for the “Stones of Stenness” in Orkney, with a page or so of free text notes linked to it in `rctextrep`, and 213 separate archive items (photographs, excavation notes, drawings and so forth) catalogued in `rccollect` and its neighbouring tables.

Table Name	Rows Loaded	Description
rcmain	221,416	One record per archaeological site or building
rctextrep	169,035	Free text: archaeology and/or architecture notes
rcasp	1,100	Essays for the <i>Accessing Scotland's Past</i> project
rcshortref	168,222	Bibliographic references
rcbibliog	51,302	Simple library management table
rcarcref	769,460	Link between site records and archive collection
rccollect	680,082	Archive collection table: photos, maps, etc.
rcpeople	5,779	Biographical details on people mentioned in text
rccollections	193	List of collections deposited with RCAHMS
rccollection_essay	20	Descriptions of particular collections
thesaurus	2,214	Hierarchical thesaurus of terms

Table 4.1: Functions of core database tables

The Architecture Record is less strictly geographically based, and often focuses on collection material such as architects' plans, which may not even have a corresponding physical site. The database was altered in the 1990s to accommodate this, making it equally possible to use `rccollect` as the driving table, and think of the geographical location as dependent on it instead of the other way round. An inevitable result is that views of the data including both archaeology and architecture (which is what users, especially non-specialists, often want) are compromises to some extent: each must either take a site-based or an archive-based approach. With the former, it would be normal to make outer joins to satellite tables starting from `rcmain`, so that all relevant `rcmain` records are retrieved, whether or not there is associated collection or other material. (See, for example, Ramakrishnan and Gehrke (2000) for details of how relational outer joins work.) Similarly, an archive-based view will start at `rccollect` and retrieve all its records that meet the criteria, with associated geographically based material if it exists. This project is about retrieving images, so `rccollect` was used as the driving table. This inevitably means having to repeat text in the presentation, as a group of several images may all link to the same site record.

Relating each image to its text requires a five-table SQL join, driven from `rccol-`

lect. Even with only about 50,000 records this can be a slow operation, and there was clearly a case for pre-joining the tables, in this retrieval-only application. Pre-joining means de-normalising the data: creating a single table from a group of linked ones, by repeating fields as required; e.g. for the “Stones of Stenness” example mentioned earlier, the de-normalised table would have 213 rows corresponding to the 213 archive items, and each row would repeat all the site record data and free text notes. (See Ramakrishnan and Gehrke (2000) for further information on normal form and de-normalising.) MySQL does not support views (virtual pre-joined tables, not instantiated until called), which would be the normal route, so a real cantrip table was created; the details are given in Appendix B. Only those fields believed relevant were included, being the principal text fields from each table. In fact some fields were omitted that the test queries would have benefited from, e.g. collection and copyright. They were left out because of their sparseness — only a very small percentage of the rows have values in these columns. But when the data is present, it is useful. This type of omission would be very easy to correct in a revised version of the application.

How justified is the pre-joining of tables? For a database constantly being updated, as the production NMRS database is, it would be out of the question. Every time a new collection item is catalogued the entire body of site text has to be re-written, wasting space and slowing performance. Much worse, every time an alteration to a word of the site report is needed, it has to be repeated for each of the copies, i.e. 213 times in the Stones of Stenness example. However, there seems a strong argument, in these days of fast data extraction and transfer, for making a separate copy of the data for a query application, updated (say) daily, in which the tables are pre-joined in the manner just described, so that retrieval is very fast because no joins are needed.

As explained in Chapter 2, using only the images that had full captions would have been simpler than joining the five tables, but this would have meant reducing the dataset to beyond the point where the project was of interest or practical use. Given that the full dataset is being used, there is a problem when a site has multiple attached images — as is generally the case with the more important sites in the NMRS database. It is almost impossible to disambiguate the images in CANTRIP, because so few have individual captions. If the query has been answered by site text, then all of that site’s

related images will be returned irrespective of their individual subjects. The images can only be distinguished from one another if they have captions of their own in addition. RCAHMS is well aware of this problem, which arises from the way the collection was originally built up, when researchers were expected to want to start with the site report and look through boxes of related archive material. In this case individual image captions were rarely necessary. Changing the organisation of the material for the present users, who often want to see particular types of pictures across many sites, is a huge task.

The amount of indexing used was lavish. When there are a lot of update transactions multiple indexes will be a real drag on performance, as indexes are constantly having to be updated and re-balanced.¹ But for a query-only application there's no argument against them except on space grounds, which is rarely a major worry now. Standard SQL left-anchored indexes were used on all primary and foreign key fields and, for the baseline application (Section 5.2), on many of the text fields. These indexes are left-anchored in the sense that they can only work if the text at the beginning of the field is supplied to match against, i.e. they can match "B%" but not "%B", where "%" is the SQL multi-character wildcard. MySQL "fulltext" indexes were used on all the free text fields, sometimes individually (as on *rcstextrep*'s report column) and sometimes in a concatenated set, as on the site classification fields in *rcmain*. Details of the index construction are in Appendix B. The fulltext indexes are of particular interest, and are discussed further in Section 5.3 and Chapter 7.

A number of minor data-cleaning jobs were done on the database, such as finding and correcting common misprints against the controlled vocabulary lists. Where there were separate archaeology and architecture text notes these were merged, following what is now becoming standard practice in RCAHMS' presentation of material.

¹Like most RDBMSs, MySQL uses balanced B-Trees for its standard indexes. To retain efficiency these have to be rewritten whenever they become too unbalanced because of additions or deletions in branches of the tree.

4.2 Processing the image collection

Around 60,000 scanned image files were copied from the NMRS collection for this project. They were all in jpeg format, as thumbnails and medium-sized (typically 20–35Kb) screen-resolution files. The thumbnails and larger versions have the same file names, but are stored in separate directories.

Each image that is associated with a specific record in the archive collection is given a numeric filename matching the primary key of the rcollect table, field arcnumlink. Many of the digital images are either not directly linked to the archive, or have been differently named for some other reason. There were also quite a lot of duplicates where the same image had been used for different purposes and separate copies ended up in different directories. The first task, therefore, was to put all the images together and remove those without an obvious database link. This reduced the number of images to be used by the project to 50,173.

The image retrieval software required the image files to be sequentially numbered from zero, with no gaps. This was true of both the systems tried. It was tiresome, but the simplest solution was to rename all the images and add an extra column to the rcollect table to provide the link to the new filenames. This causes a small degradation in performance, but it's insignificant in practice.

Because one rcollect record may relate to several rmain sites, the same image can be returned multiple times with different text attached. It seemed best to allow this repetition of the image as the system is retrieving by text, so arbitrarily discarding text records would be invalid. But this means the image filename cannot be used as a unique key, and hence a new surrogate key was needed in the database. This led to a much more significant performance issue, as the content-based image retrieval (CBIR) software will always retrieve items by image filename, whilst the search engine retrieves by database primary key. If the results are to be merged then one of these keys has to be translated to the other, as is done in the CBIR version of the application, described in Section 5.7. This clearly affects performance, but the only obvious solution (which has several drawbacks) is to duplicate and rename the images that have multiple texts.

4.3 CBIR work

Content-based image retrieval (CBIR) is principally based on physical properties of digital images: the distribution of colour, the texture, the shapes depicted and the relationships between shapes. As mentioned in Section 3.1, techniques also exist for dealing with semantic properties like objects and their rôles, and with meanings associated with pictures, like mood and emotion. See Del Bimbo (1999) for a full treatment of the subject.

A CBIR system may use several features of an image. Two of the fundamental ones are colour histograms and edge histograms. A colour histogram measures the percentage of different shades within the image in order to compare it with other images. Thus a picture of a sunset, having a large yellow sun sinking in a mostly orange sky, will be judged similar to other sunsets with similar colours, but may also, on this measure, be similar to a picture of a bunch of orange and yellow flowers. Edge histograms measure the angle of slope of edges found within an image; an edge being a sharp discontinuity of colour in an image, which can be automatically detected. For example, the Xie software (Xie (2001)) uses a 4-bin system, finding the percentage of edges close to the vertical, to the horizontal, and to $\pm 45^\circ$. Thus images can be compared on the similarity of the shapes they depict. Combining this feature with the colour distribution will rule out the flowers as being closely similar to the sunset, though it will not be able to prevent confusion with a picture of a yellow ball on an orange table.

Most of the digital images in the RCAHMS collection are black and white. The photographic archive as a whole is overwhelmingly monochrome, as it dates back to the early 20th Century. Digitisation projects have tended to concentrate on a higher proportion of colour images, but these still form the minority. As more of the historical collection is scanned the proportion of colour is likely to decrease if anything, even though survey recording is now done in colour. The colour histogram technique does not work at all well on black and white images, and edges are also more difficult to detect within shades of grey. It's remarkable what the human eye and brain can interpret, when you consider it presumably relies — at least in part — on similar techniques; but it's safe to say that CBIR technology struggles to make much of monochrome pictures

at present. To make real use of a simple system using *only* colour and directed edge features, it would have been necessary to restrict the collection to a much smaller set of pre-clustered, colour images.

This project was primarily concerned with language processing and information retrieval, rather than with CBIR techniques. It therefore seemed sensible on the one hand to retain as much of the text data to work with as possible, and on the other to obtain an advanced CBIR package that would work well with a very large and mixed image collection — something that is comparatively rare in research CBIR software. The system chosen was SIMPLIcity (Semantics-Sensitive Integrated Matching for Picture Libraries), developed at Penn State and Stanford (see Wang et al. (2001)), which was kindly made available for this project. SIMPLIcity is a sophisticated system, using semantic classification as well as colour, texture, shape and location features.

There are three steps to running SIMPLIcity:

1. Index all the images, creating a colour map and the various feature distances for each. This took about 21.5 hours for the collection of 50,173 images.
2. Create a database from the summary information on each image. This step is very fast. It produces a very compact index database — less than 400Kb for the entire 2Gb collection — which is all that is needed for the retrieval step.
3. Retrieval: find images that match a “query” image number supplied. This step is done at run-time, within the application. As many images as are required will be returned, ranked in order of increasing distance from the query. The seed image used is always returned at the top of the list, with a distance measure of zero. This step does not involve any reference to the image files themselves, only to the index database. The images are referred to by their filenames, and the actual graphics are only needed for presenting the results to the user.

Chapter 6 discusses the results fully, but from experiments at this stage it was clear that the CBIR software works well with certain types of image and poorly with others. It is best with line drawings and quite good with architectural plans, but, as noted above, it is not suitable for most black and white photographs.

4.4 IR tools

Information retrieval (IR) technology is very well established, and very successful. One of the aims of this project was to learn how its indexing works and to explore ways of improving on the basic inverted index using NLP methods, in particular NER. For a full and very clear treatment of inverted index techniques, see Belew (2000).

The term “search engine” is convenient because the inverted index approach is what underlies Web search engines. It’s a reasonably accurate characterisation of how the relevant CANTRIP application version is designed to work. The basic idea is to index all the significant words in the text, irrespective of structure and fields, and to tabulate how often each word occurs in each document. Here “document” means an image reference along with all its associated text. A “significant” word is one that does not appear on a “stop” list of very common noise words like “a”, “an”, “the”, “if” and so on. Once this index has been constructed, it is “inverted”: instead of recording against each document the words it contains and their frequencies, we record against each word the documents it occurs in, and how often.

To give an example: the un-inverted index has one entry for each document and is of the form:

```
<doc-id> <word-1> <word-1-freq> <word-2> <word-2-freq> ...
```

The inverted index, by contrast, has one entry for every word in the corpus:

```
<word-n> <doc-count> <tot-freq> <DFV-1> <DFV-2> ...
```

where the **document frequency vector** (DFV) is a variable length array of the form:

```
# <term-freq> : <doc-id-1> <doc-id-2> ... #
```

For example, this is the entry for the keyword “adjust”, which is an uncommon word in the NMRS data:

```
adjust 6 7 # 2 : 44718 # 1 : 19658 19879 22877 54133 55574 #
```

This entry shows that “adjust” occurs in 6 different documents, and is in the entire corpus a total of 7 times. It occurs twice in document 44718 and once each in documents 19658, 19879, 22877, 54133 and 55574.

The **term frequency**, f_{kd} , of a keyword k is a count of how often it occurs in document d . This is recorded in the inverted index, for each document the word appears in, as shown above. The **inverse document frequency**, idf_k , of keyword k is inversely proportional to the number of documents that contain k , as a percentage of all the documents in the corpus. At its simplest, this is:

$$idf_k = \log \left(\frac{NDoc}{D_k} \right) \quad (4.1)$$

where $NDoc$ is the number of documents in the corpus, and D_k is the number of documents that contain keyword k .

It is usual (following Robertson and Sparck Jones, quoted in Belew (2000)), to normalise with respect to the number of documents *not* containing the word, and to smooth the formula by adding small constants to moderate extreme values. This gives:

$$idf_k = \log \left(\frac{(NDoc - D_k) + 0.5}{D_k + 0.5} \right) \quad (4.2)$$

Thus a rare word, that appears in few documents of a large corpus, will have a high IDF weight. The intuition on which TF-IDF weighting is based is that words that are best as content-carriers and discriminators are those which are rare in the corpus as a whole, but occur frequently within the documents that are “about” them. (This is a very rough gloss of quite a complicated subject, see Belew (2000), Chapter 3, for a full discussion.) So the TF-IDF weight for a word is the product of its TF and IDF weights:

$$w_{kd} = f_{kd} \times idf_k \quad (4.3)$$

Richard Belew’s book (and website) provides a collection of freeware Java programs intended as teaching aids for students learning to build search engine tools. These were adapted and used as the basis of all the CANTRIP versions that rely on inverted indexes, i.e. of all except the baseline and fulltext versions. There are two main packages in the Belew collection: one for creating the inverted index and one for running the search engine to answer a query. The first package contains 17 related Java programs and the second 11. These needed a significant amount of alteration to run with this project’s data, and in places the architecture was incomplete and extra classes had

to be built. The code that was adapted is not given, but the four new classes that were written are listed at Appendix D.

In particular, the search engine uses normalisation by document length - basically allowing for the fact that short documents will inevitably contain fewer keywords than long ones. It therefore needs an extra factor that isn't provided within the software suite. (It is available as a hard-coded list of weights for the demonstration corpus the software is written for.) Following Belew (2000), the calculation used, in DocLength.java for **document length**, $len(d)$, was:

$$len(d) = \sqrt{\sum_{k \in d} (w_{kd}^2)} \quad (4.4)$$

where w_{kd} is the TF-IDF weight, from equation 4.3 above. This gives a weighted average of term weights across the whole of each document. Experimenting with the smoothed and unsmoothed idf_k (equations 4.1 and 4.2) did not produce measurable differences on my data, but the final version used the smoothed factor.

The DocLength.java program is run as a separate step, after the inverted index has been created on the exported database text. This second pass is inevitable because, as equation 4.4 shows, calculating the document length involves summing over all the documents each keyword occurs in; so it cannot be done until the whole corpus has been processed. Reading the inverted index file itself is the simplest way of collecting the values needed for the calculation.

4.5 NER and term weighting

The initial plan was to use the candc named entity recogniser, described in Curran and Clark (2003). This is a Maximum Entropy tagger, designed for POS tagging, chunking and NER, trained on the MUC-7 conference data. The alternative recogniser was TTT, also developed within the LTG (see Grover et al. (1999)). This is a comprehensive suite of taggers, using a mixture of statistical and rule-based techniques. Several experiments were tried with TTT, on POS tagging, chunking and NE recognition. By

default the system is only set up for finding number and time entities, so *candc* was preferred.

However, the only available model for *candc* was based on data from a completely different domain, and it found very few entities in the NMRS data. The intention had been to use the ready-trained model for an initial pass, and then hand-correct the output, marking missed entities, in order to create an annotated training set to use to train a new model. This would have been a reasonable approach if the initial pass had found even 50% of the entities, but in fact it found far fewer. The NMRS data contains very domain-specific archaeological and architectural terminology, and many of the specialist terms wanted as entities (“chambered cairn”, “cup and ring mark”, “cruck framed”, “paper mill”, and so on) are not capitalised, which will severely hamper most NE recognisers. Of course, standard “person” and “organisation” terms were also wanted, but these only accounted for about half the entities. Because of time constraints it was decided not to persevere with this approach, but to abandon the use of the *candc* recogniser.

A simple gazetteer strategy was used instead. There was a hierarchical thesaurus of terms available, plus three different classification fields on each site record, a table of important people and organisations and a table of bibliographic references. Thus it was possible to extract a comprehensive term-list. Duplicates were removed and the terms converted to lower case. Then a series of transformations was applied, to turn the terms into simple regular expression patterns. (Refer to Jurafsky and Martin (2000), Chapter 2, for an explanation of regular expressions.) Punctuation within terms was made optional and common alternative words were allowed for; for example “Aalen, Whalen & Stout” became “aalen,? whalen (&|and) stout”. As far as could be done without too much hand-editing, alternative forms of names were provided for, eg “Alexander J. Adie” became “a(lexander)?(j\|.?)? adie”.² Even after removing as many as possible of the numerous duplicate forms thus generated, there were over 20,000 terms in the file, and much more hand-editing than there was time for would have been needed to really tidy it up. A short sample from the file is given at Appendix E.

²An important set of alternatives that was overlooked is the “Mc/Mac” issue, so common in Scottish names. This would be easy to include in a revised version.

There is some discussion in Belew (2000) about putting extra weighting on “proxy” fields, such as document title, but this is not implemented in the software. I decided to implement weighting for the NEs, and section 5.6 describes the relevant application version. The theory behind this was that not only are the NEs supposed to be “content carriers” but they also require extra weighting to balance the fact that their frequency counts as compounds will be lower than if the constituents were counted separately. This is similar to what’s put forward in Lewis and Sparck Jones (1996). The program `Loader.java` (see Appendix D), reads in a file of “heavy tokens” and alters their weights in the inverted index file. The input tokens are processed in exactly the same way as raw text is in the initial index creation, so that they will match successfully against the index entries. For each word read the steps are:

1. Clean the word token by converting to lower case and removing attached punctuation.
2. Check the token against the stop list and discard if it’s a noise word.
3. Stem the token using the Porter stemming algorithm (as implemented in the Belew suite by `PorterStemmer.java`).
4. Then add it to the list if it’s not already there.

Once the NE tokens had been weighted, the ranking calculations in the search engine were altered to factor in the “personal term weight” of each keyword, set to 1.0 for standard tokens and some higher value such as 1.5 for the “heavy” ones. Experiments were made with different values, and 1.5 seemed best. Thus equation 4.3 was replaced by:

$$w'_{kd} = f_{kd} \times idf_k \times ptw_k \quad (4.5)$$

where ptw_k is the **personal term weight** for keyword k . The new w' factor was only used in the search engine ranking, not in the calculation of the **document length** given in equation 4.4. Experiments using w' were made (see `DocLength.java` in Appendix D), but it appeared that documents containing many NEs became grossly overweight. On reflection, this tallies with common-sense intuition, since the point of the document length weighting is to smooth out distortions caused by some documents being

“meatier” than others, not to emphasise them. Without such smoothing long repetitive documents would always dominate over short pithy ones.

The so-called “heavy tokens” were produced from the gazetteer file described above. The gazetteer terms were divided into two sets: compound terms (including at least one space in the string), and all the terms (compounds and single words — this was the “heavy” file). Once the data had been exported from MySQL into an XML corpus³, the compound NE terms were marked in it using `DbPrepper.java` and `NEMarker.java` (listings in Appendix D).

The first of these programs reads in the entire corpus, document by document, extracting the free text sections and feeding their text to the `NEMarker`. This searches the text passed to it, comparing it against all the regular expression patterns in the gazetteer file, and replacing each occurrence of a pattern with a version having underscore characters instead of spaces: this effectively turns the matched pattern string into a single token. This tokenisation replaces the standard IR method of using individual words as tokens to build the index over. Here the text is tokenised into strings which may be single words or compound entity tokens, and the inverted index will be built over these tokens.

The gazetteer list was sorted in descending order of term length, to ensure the program always found the longest string available. Since there were 50,173 documents and over 20,000 gazetteer patterns, the `DbPrepper` step was very slow, needing about 40 hours to run on a PC. (In fact it was done in concurrent sections on multiple machines, by splitting the corpus file.)

The `NEMarker` processes a document text passed to it as a single string, marking NEs in it. It can therefore be used unchanged in the relevant `CANTRIP` versions (see Sections 5.5, 5.6 and 5.7), to process the user query string and mark NEs in that.

To summarise, the complete set of steps for preparing the various search engine indexes was:

³The search engine software required XML data as input. The conversion from SQL simply involved turning database fields to XML entities, with a DTD corresponding to the field layout. No extra markup was required within fields.

1. Dump the raw data from MySQL in the XML format required.
2. Create a second version of the XML corpus, with NEs marked by `DbPrepper.java` and `NEMarker.java`.
3. Create a separate inverted index on each of the XML corpora.
4. Create a set of “heavy” tokens, and run `Loader.java` to add weights to these in the appropriate index.⁴
5. Calculate the document lengths for each version of the corpus and produce a free-standing lookup file for each, arranged by document ID. This is needed by the search engine.
6. We now have weighted and unweighted versions of the index file for each input XML corpus, ready for the search engine to run against, given an input query string. This input query string may itself undergo pre-processing that is exactly the same as the data’s.

⁴In the final version only the NEs were weighted, but the software is designed to work with any input data, and initial tests were done by weighting words from the site name field in the “plain” corpus.

Chapter 5

Implementation

This chapter describes the final implementation of CANTRIP. As explained earlier, a layered system was the objective, comparing different techniques separately and in combination. To get a coherent set of comparisons, it turned out that six separate versions were needed. Indeed, it would have been interesting to split some of the layers still further, but if any reasonable user evaluation was to be possible a limit had to be set. Figure 5.1 shows the layout of the application and how its components interact.

The application was written using portable tools, available on all popular operating systems. Open source software was used as far as possible, but the SIMPLIcity CBIR system is on academic research licence to Edinburgh University. The Belew search engine programs are made available freely by the author (see Belew (2000)). The data of course belongs to RCAHMS. Software was written in the most appropriate tool for the particular job: SQL obviously for the database work, PHP (and JavaScript) for the HTML interface to the database, Java for the search engine tools based on the Belew suite, and awk for most of the myriad reformatting jobs. The SQL used for setting up the database is given at Appendix B, and samples of the PHP and Java programs are in Appendixes C and D. The programs were too long to print in full.

The name CANTRIP was chosen as a nod to the existing Web-based query applications developed by RCAHMS: CANMORE and CANMAP. CANMORE is mainly for site-oriented text queries, and CANMAP provides a geo-spatial interface. A “cantrip” is

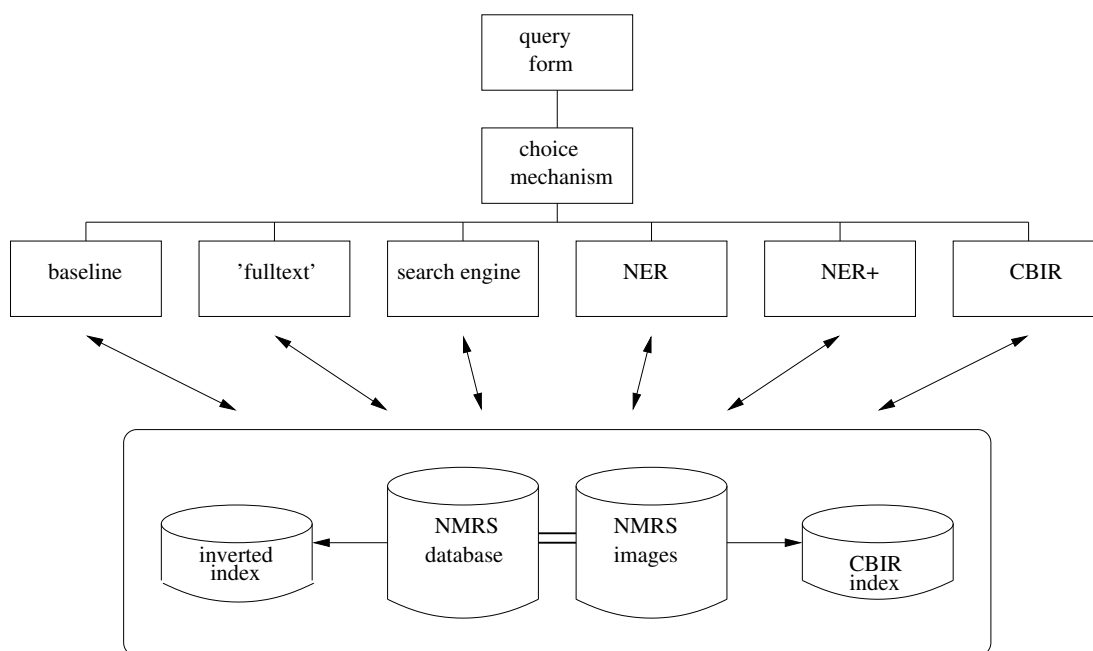


Figure 5.1: CANTRIP application

of course a magic spell or piece of mischief, in Scots dialect, so it seemed appropriate enough.

5.1 Interface design

Various options were considered for the query interface. It could be a single free text box, which is appropriate for the search engine related versions. This is attractive because it is so straightforward for the user, and can look just like any Web search system. It may also make it more likely that the query is a complex string that can usefully be pre-processed, in the manner of a question-answering system. Conversely, there are arguments for an interface with a selection of fields matching the database structure. This will tend to produce better results, as long as the user has some understanding of that structure; otherwise it may be counter-productive.

To enable a fair evaluation, it was important that all six versions could share the same front screen, so a compromise was made. The query screen is illustrated in Figure 5.2.

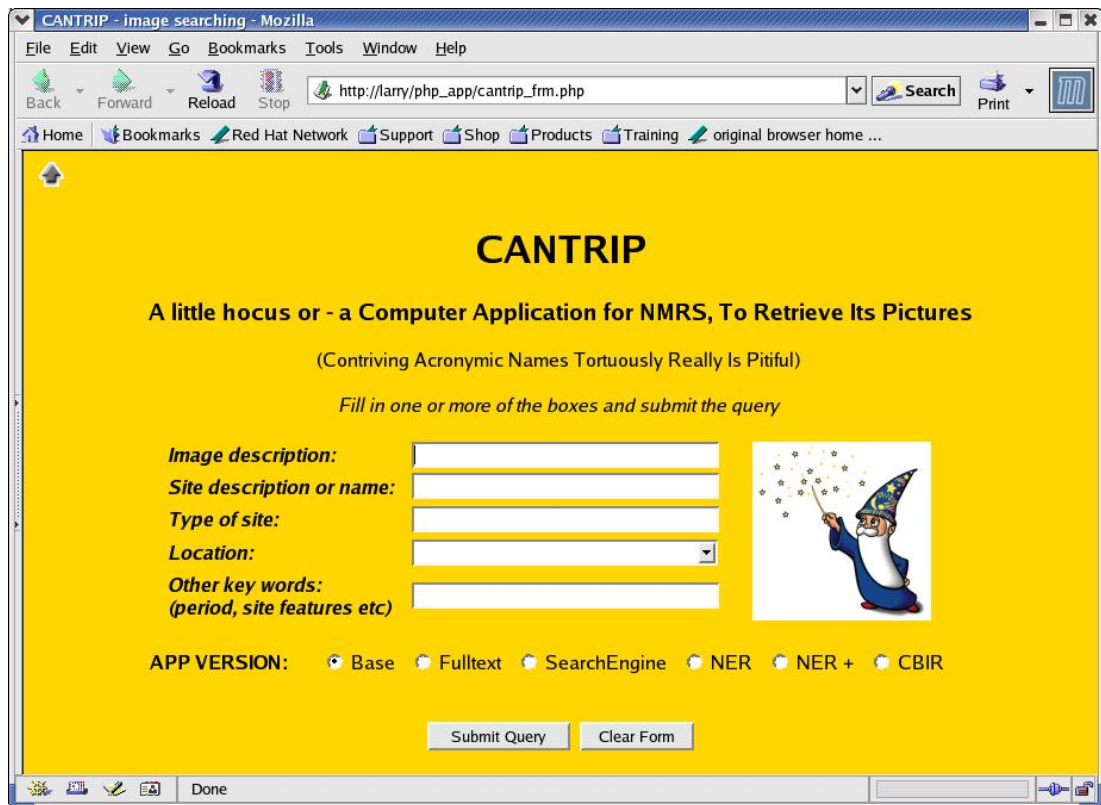


Figure 5.2: CANTRIP query interface

It has only five fields, which are intended to be suggestive of aspects of a typical query, rather than directly linked to particular database fields. An alternative version with just “Who?”, “What?”, “Where?”, “When?” was also considered. This would have been the preferred option except that, for the baseline and fulltext versions, it *was* necessary to have an obvious correlation with database fields, and in fact the present structure doesn’t lend itself to this approach. The “Where?” is pretty well defined, but almost everything else is a “What?”, because the database doesn’t actually use person and period fields very much. If the constraint of having six versions with the same front end were removed, it would be interesting to explore this query interface further: in many respects it would be ideally suited to a mechanism based around named entity recognition.

The “location” field is a bit of an anomaly. The project was deliberately not concentrating on place names, partly because RCAHMS is already very well set up for location

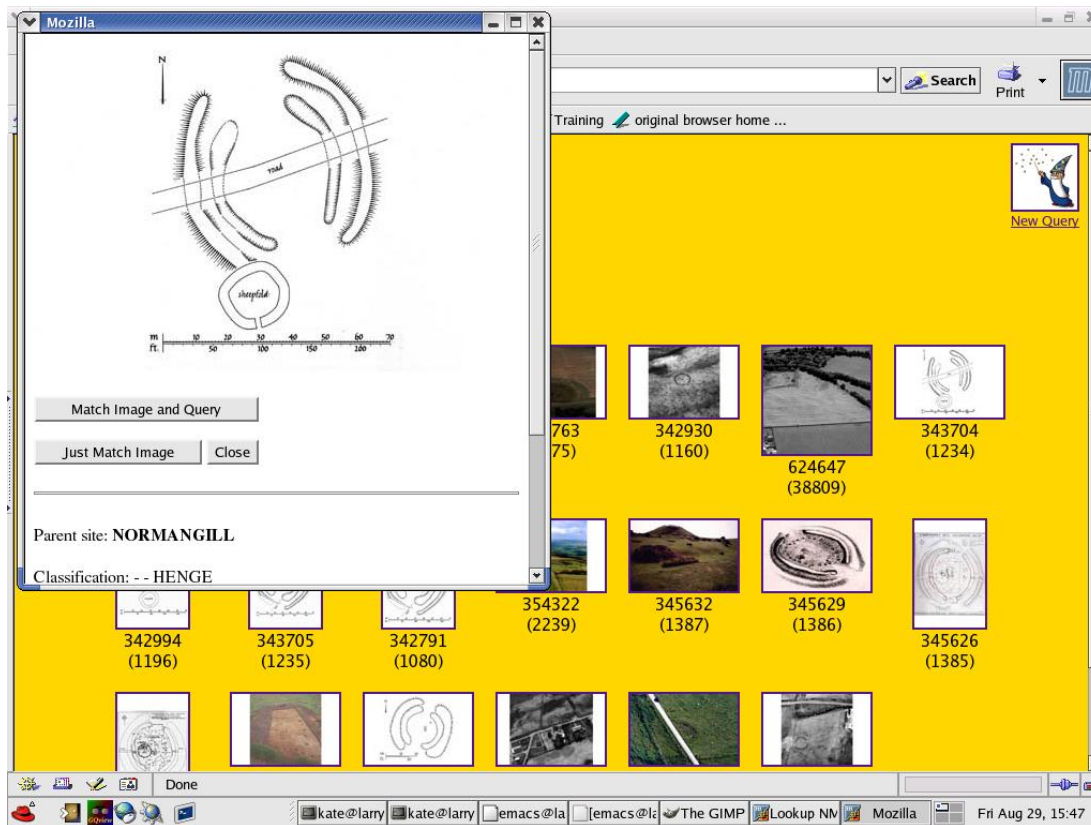


Figure 5.3: CANTRIP results screen

queries and for them this is really a solved problem, and partly because another project on this topic was in prospect for the coming year. On the other hand, the intention was to experiment with intersecting search engine and database results, and for this a simple equality-match field like one of the location code lists was ideal. The list of Scottish Councils was chosen, simply because it is of manageable length (34 entries) to display conveniently in a pick list.

Each version returns a results screen containing up to 20 images, in their thumbnail versions. The limit (which is simply a variable in the code, so could easily be made user selectable) was set for performance reasons and because it seemed about the most that a user would want to examine individually.

When the user clicks on an image a pop-up window appears, with the larger image and its associated database text. Figure 5.3 shows a typical example. The pop-up

window contains either one or two extra matching buttons, depending on the version. In the two simplest versions (see Sections 5.2 and 5.3) there is a single button labelled “Match Image”, which runs a CBIR match and brings back the ten closest content-based matches to the current image. In the next three (Sections 5.4, 5.5 and 5.6) there is another button: “Match Image and Query”. This does a CBIR match, but intersects the results with an extended results list (up to 300 hits if available) from the text query. The matching works well for some images and much less well for others. The final version (Section 5.7) does not include the second button, as it has already performed an intersection of results.

5.2 Version 1: Baseline

The baseline version was implemented very simply. It uses standard left-anchored SQL indexes (see page 25 for the characteristics of left-anchored indexes). It inserts wildcards between and after query terms within each field, so that, for example, “roman camp” as a query would match “roman temporary camp” in the database. Less helpfully, “Edinburgh Castle” will match “Edinburgh, North Castle Street”. Like all the CANTRIP versions, it is case-insensitive.

The wildcards are placed directly between query words, stripping spaces. This sometimes produces slightly odd results, e.g. “stone circle” might return a picture of “stonemasons sitting in a semi-circle”. (Actually it doesn’t. It works quite well; but only because there are rather more of the former than the latter.) This would have been complicated to deal with, as one would have to allow for punctuation, newlines and end of field markers; so fixing it wasn’t appropriate for a baseline version. The other versions behave more as one would expect in this respect.

The correlation between fields on the query form and in the database is as shown in the second column of Table 5.1. (The third column is explained in the next section.) The baseline version does not search the large free-text fields, report and aspnates at all, because it cannot index them so performance would be unacceptably slow.

Query field	Database field(s) — Baseline	Database field(s) — Fulltext
Image description	desc1	desc1
Site description/name	nmrname, alname	nmrname+alname
Type of site	class, classgrp, classsub	class+classgrp+classsub
Location	council	council (not fulltext)
Key words	period, desc1 (front wildcard)	report, asnotes, desc1

Table 5.1: Query items and corresponding database fields

Note that with this version and the fulltext one, it makes a big difference which query fields are used. To return to the earlier example, if the user enters “stone circle” as a query in the “Image description” box, the system returns only four hits, two of which are not stone circles at all. The caption of the first begins “Stone bearing incised cross within a large circle.” However, if the same query is entered as the “Type of site”, then twenty (the maximum) good hits are returned, as this is a standard classification term. An experienced user would get this right, but it is certainly confusing to the novice.

A full listing of the baseline program is given at Appendix C.

5.3 Version 2: “Fulltext”

The MySQL database (version 3.23 and later) allows inverted indexes to be built against individual fields or concatenated groups of fields. These can be searched using SQL of the form:

```
select <fields> from <table>
where match(<fulltext index>) against (<query string>)
```

The “match... against” syntax is peculiar to MySQL fulltext indexes, rather than being standard SQL. However, other RDBMS packages implement their own versions, e.g. Oracle has a product that was originally called “TextRetrieval”, then “ConText” (and presumably still exists under some name), that works in much the same way.

These indexes use the TF-IDF mechanism described in Section 4.4. The only drawback is that it is a “black box”: one cannot adjust the way it works. In future versions this is likely to change, and will be of great interest for this type of work. Of course, MySQL is open source and one could in principle hack the code. This was rejected for the present project as being far too ambitious, but it’s something it would be interesting to pursue.

There is only limited information in the MySQL manual about how the fulltext indexes work. In fact I am merely assuming from the description of term weighting and ranking that it is using standard inverted index techniques, because this is not explained explicitly. The text is tokenised, cleaned of punctuation and normalised to lower case, but it seems that stemming is not done. Words on a compiled-in stop list are discarded, as are all words of fewer than four characters. (So it will not succeed with one of the example queries, looking for “dog bones”.) It’s not explained, but the assumption is that each database record is treated as a “document” for the document frequency counts, but there is no indication that differing document lengths are taken into account. The basic TF-IDF principles apply: words that have high discriminatory power should get the highest weights.

Although the idea of being able to use the strengths of fulltext indexing within the database is very attractive, there is a potentially serious drawback. As the MySQL manual points out, inverted indexes work best when they contain a *lot* of text. On small bodies of text the weighting will not be balanced and results are unpredictable. The NMRS report field on table `rtxtrep` is large enough for this not to be a problem, but it’s not clear that the shorter text fields have enough material to ensure that words get their correct term weights. The third column of table 5.1 shows the database fields that were used in this CANTRIP version. The “+” signs indicate where an index was built on a concatenated set of fields. This was convenient for querying, but was also done in order to give the indexes more bulk. Details of the indexes created for this and the baseline version are included in Appendix B.

5.4 Version 3: Search engine

This version implements plain TF-IDF indexing, as described in Section 4.4. The implementation was done by adapting the Belew suite of Java programs. Here, the database structure was completely ignored. All the relevant fields — the same ones as were used in the fulltext version — were dumped from the database as an XML file, with their unique record identifiers. The inverted index was built by simply working through all the text attached to each image record, irrespective of the field it originated in.

The size of the file thus created was just over 41Mb, which is under 10% of the whole database size. Only about a quarter of the records were used — those attached to digital images — and only a very small set of fields, albeit the longest ones. The number of words in the file (excluding XML tags) was calculated to be approximately 3.9 million. At first glance those figures suggest a surprisingly high percentage of long words(!), but dividing the file size by the number of words (to produce an average word length of over 10 characters) isn't valid, because the file is so bulked out with XML tags.

The resulting index contained just over 25,000 terms, being the distinct word types present in the data. It shows which are the most frequently used terms, and is interesting to browse. This is a fairly average figure for vocabulary size.¹ On the one hand it is likely to be augmented by the high number of proper names, the long period over which the text was collected, and the fact that it has so many domain-specific terms in addition to everyday English words. Counter-balancing this is the fact that the text tends to be written in a fairly fixed, formal style, with a lot of formula phrases.

To run this version, the text from all query fields was simply concatenated and passed as a string to the search engine. This worked against an unweighted inverted index built from the plain XML corpus.

¹Some figures for comparison: a small news-wire corpus contains around 11,000 distinct word types; the Switchboard corpus of spoken American English is roughly the same size as our corpus and has about 20,000 different types. Topping the league is Shakespeare, with a vocabulary of over 29,000 in under 900,000 words in his Complete Works. A typical English dictionary would have around 200,000 entries. Figures taken from Jurafsky and Martin (2000).

There was what amounts to a bug in this and the following application versions. The location query field is a pick-list against a numeric code for Council region. For this and all the other search engine based versions, the corresponding text (e.g. “Perth and Kinross”) was passed in the query string, instead of the numeric code. It was always anticipated that the two SQL-based versions would have a head-start here, because they do a simple yes/no comparison on location code. But what didn’t dawn until quite late on — when it was noticed that the search engine versions were doing *very* badly on location queries — was that the relevant Council names should have been added to the dump file for each XML document, but hadn’t been. By this time it was too late to fix the problem (the killer 40-hour program would have had to be re-run), so these versions are disadvantaged on queries involving location. However, it turned out to permit an interesting extra analysis of the results (see Chapter 6), so no real harm was done.

5.5 Version 4: NER

This version works in the same way as the previous one, but on a different index. In this case the XML corpus mentioned above was processed by DbPrepper and NEMarker to mark the NE phrases. The methodology was explained in Section 4.5. The process involved turning compound NEs into single tokens. Once this was done an inverted index was built on the corpus, in the same way as for the plain text.

This index contained more terms — over 27,300 — but with lower frequency counts. Words from the text were not duplicated, and the longest available entity string was always used. I.e. if a particular record contains the string “Robert Adam”, that will appear in the index just once as a single token, not three times as “Robert”, “Adam” and “Robert Adam”. However, if the document only contains “Adam”, that will be flagged as an entity. This accounts for the higher vocabulary count, as the components of many compounds will occur in their own right. The reason for only using the longest available string was to avoid spurious matches, but with the realisation that sometimes a partial match would be beneficial. The partial matching option is already being

tested by the plain search engine version, but only for single-word components. There is room for more variations on the theme here, by splitting long multi-word compounds that contain multi-word subsets, and indexing on both.

An effort was made to identify variants on the same entity name, but only so much was possible with the way the gazetteer was built (detailed in Section 4.5), and clearly more could be done. For example, the index token for the celebrated architect Playfair is “w(illiam)?_h(enry)?_playfair”, which matches several, but certainly not all, possible variants on the way his name might be represented. It occurs 90 times in the text, in 65 different records. Of course “playfair” alone is also indexed, and is found much more frequently: 225 times across 103 documents. These might be matches on part of the same name (e.g. “William Playfair”), but could also be a different entity; for example his uncle “j(ames)?_playfair” occurs 59 times in 45 documents. These figures show that “w(illiam)?_h(enry)?_playfair” is a good discriminator term, as its total frequency count is relatively low but it tends to occur in clusters (the 90:65 ratio is well above 1.0).

The query string from the user is concatenated, as in the previous version, and then passed to NEMarker to have NEs marked in exactly the same way as in the database, to ensure that any present will match each other. Thus a query on “W H Playfair” will be translated to “w(illiam)?_h(enry)?_playfair”, and will then match occurrences of (say) “William H Playfair” in the text.

5.6 Version 5: Enhanced NER

This version uses an index built on the NE-marked text, but with extra weights added to the NEs, both the single word and compound ones. The reasoning was discussed earlier (see page 33), as was the fact that the initially tried weight of 2.0 seemed to be too high, and the final version used 1.5.

Expansion of the query is also done, if the “Type of site” field is completed. Any user entry in this box is wildcarded and compared against the thesaurus to find the canonical

form for that term plus any entries that exist for **preferred term** or **related term**. If the term is itself a preferred term, there may be a “**use for**” entry, containing non-preferred terms it should be used instead of. For example, the following are the relevant fields for the term “cemetery”:

term	related	preferred	usefor
CEMETERY	BURIAL ENCLOSURE; BURIAL GROUND		Graveyard; Necropolis

Both preferred and non-preferred terms are used in the expansion, on the basis that non-preferred terms may occur within the free text even though they should not be in the classification fields. The expanded string is passed to the NEMarker and thence to the search engine, which uses the weighted version of the NE index. The thesaurus also contains **broader** and **narrower** terms, but these were not used. This is an area that could be explored further.

Finally, this version combines search engine and database results. As mentioned earlier, the location field had been included as a simple pick-list with this experiment in view. So here, the location from the query (if present) is used to sift the search engine results against the database, using the location code which gives a simple yes/no result. This worked well, but a large number of search engine results had to be used (300 in the final version). As mentioned in Section 5.4, this step would probably have been even more effective if the Council names had actually been attached to the records, to give the search engine a fighting chance of returning records for the right location in the first place.

This version adds three components — NE weighting, query expansion and database intersection — that should really have been tested separately. Eight versions of the application was thought impractical for evaluation however.

5.7 Version 6: CBIR

The final version uses all the components just described, but intersects the results with CBIR ones. The problem of course is which particular image to use as the seed for the content-based query. Logically, the user should be invited to choose the image of most interest from the preliminary results, but this wasn't possible if the interface for all six versions was to be identical, to permit a fair blind evaluation. Therefore the top image from the search engine was picked. They are arranged in rank order, so this should make sense. In practice, at least subjectively it often seemed to be the wrong one. Many times, in comparing the output of the last two versions, one felt "If only it had used image 3 instead...", or such like. The pop-up window buttons ("Match Image" and "Match Image *and* Query") were added to the application to deal with this deficiency. They allow the user to pick the most relevant image for themselves. No formal evaluation of these pop-up buttons was possible, but there is discussion of informal findings in the next chapter.

Once again, in order to get any results in the intersected set, a high number of returns was needed: 300 were used here, as for the search engine. Other figures for each set were tried, and this seemed to work best. My inclination was to allow the search engine a much higher number than the CBIR software, but performance became a significant factor; returning more search engine results had an unacceptable impact. The SIMPLiCity software will happily return all 50,000 images in rank order, but by the time one reaches the 300th they are really not very similar to the query image at all. Different types of result combination were considered instead of simple intersection, but none was actually implemented and tested. This is an area where further work would be useful. In the case of the database location used by the enhanced NER version, simple intersection is the obvious method, because it is just a sift of what should be similar results. But where such a different tool as CBIR is being used, there seems a case for adding new results. However, this would be a separate sub-project in its own right.

A full listing of the CBIR program is given at Appendix C.

Chapter 6

Evaluation

The first two sections below describe the formal evaluation and its results. The third and fourth contain wider analysis, including informal judgments of components that couldn't easily be measured.

6.1 Methodology

With six application versions to test, it was clear that each evaluation session would have to be quite lengthy. This meant it was impractical to aim for a large team of testers, and in fact only three people took part: two specialists from RCAHMS (an archaeologist and an architectural historian) and one non-specialist. Each session took about half a day. With so few evaluators, it was important to see how much their opinions differed, so the degree of agreement is also presented in the results.

A set of queries was provided by RCAHMS for the trials, and 16 were used. The queries were not examined¹ until after the application build was complete, and then a set was chosen that contained a balance of archaeology, architecture and industrial topics. The queries were all designed to produce a small result set, so that looking at just 20 hits was valid. An evaluation version of the application package was used, with

¹...any more than was unavoidable, as they were sent by email.

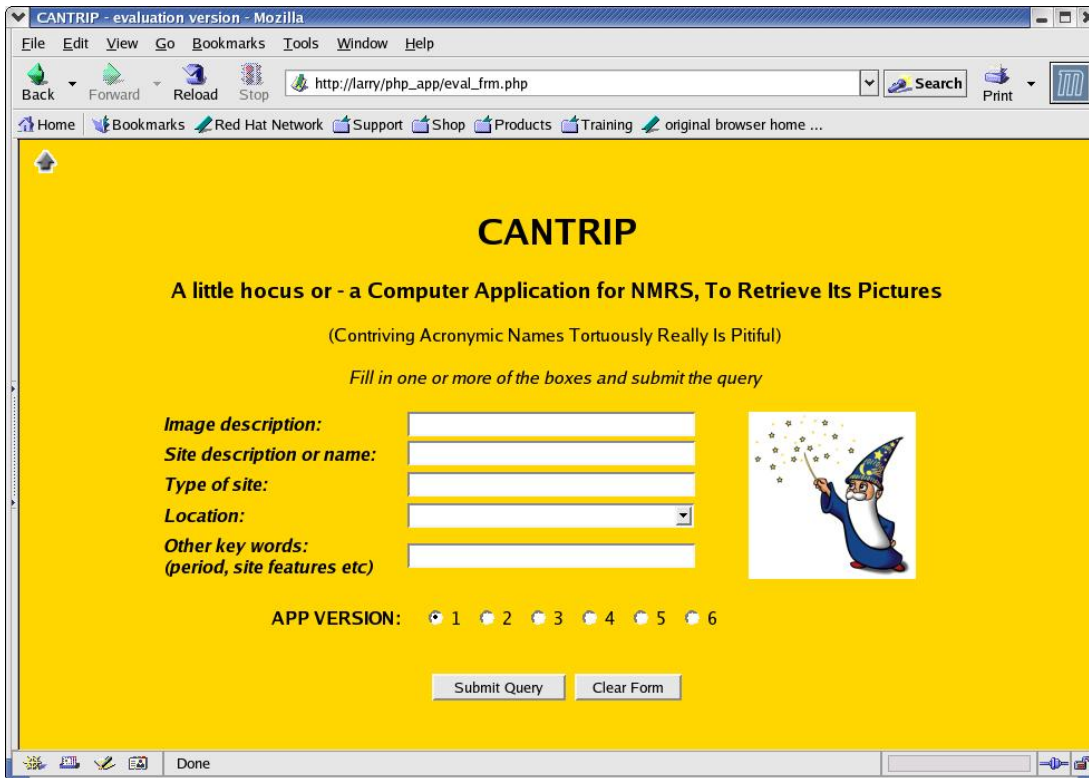


Figure 6.1: CANTRIP — evaluation version

all distinguishing marks removed from the different versions. Selection was made by buttons labelled 1 to 6 on the query screen — illustrated in Figure 6.1 — and each query was run against each version in turn. The order of the versions was mixed up, to avoid either the evaluators or me knowing which was which during the trials.

The instructions for evaluators contain further notes on the methodology, and they are included at Appendix F. An example of the evaluation sheets used is given at Appendix G. The essence of the test was to mark each of the result images as either “relevant” (a tick), “not relevant” (a cross) or, if the result was unclear for some reason, “uncertain” (a question-mark). Response time was also measured, as this would be important in a real application.

6.2 Results

The traditional measures, **precision** and **recall**, are only partially appropriate here. (See footnote on page 15 for the definition of these terms.) Precision, or the percentage of correct answers in the result, is very easy to measure, but with a ranked set one clearly wants to give more credit to a system that returns 10 good hits followed by 10 misses, than one which puts all the misses first. Recall, or the percentage of the whole target that was returned (i.e. a measure of what's *missing* from the result) is notoriously difficult to calculate on large databases. This was discussed in Chapter 3, and there's no really satisfactory answer. Most approaches involve using a tiny subset of the database and making the evaluators compare the results against the whole of it. Certainly in this case, with statistically motivated indexing, results for a very small subset simply wouldn't be comparable with the whole, nor of any particular value, I'd suggest. It's arguable (and indeed is argued in Flank (1998)) that recall is not of great importance to at least the non-specialist user, who wants to see accurate responses to the query but doesn't particularly need an exhaustive list.

In 3 of the 16 queries (numbers 1, 3 and 4), there seemed a good case for believing the successful systems had returned the entire result set available, so recall percentages were calculated here; but in most cases it's impossible to estimate. The results are given in full in Appendix H. For each query the six results are shown, in the same order each time. A completely blank line indicates that that application version returned no hits. The order of the six versions is:

1. plain search engine
2. enhanced NER version
3. baseline, standard SQL indexes
4. CBIR
5. fulltext indexing
6. NER version

Because of the problems with precision and recall estimation, a different measure of my own was used alongside them. This gives credit for placement of correct answers within the ranked set and also makes a distinction between returning no hits and returning misses, with the latter being penalised more. Each image returned in the results set is marked. For every correct answer, a “positional score” is given: 20 points for the first place, 19 for the second and so on down to 1 for the twentieth. For each wrong answer, 1 point is deducted. Where the answer is marked “uncertain” a score of zero is given. Thus the maximum possible score is 210 and the minimum is -20 (see below). Returning no hits at all earns a score of zero.

There may be existing measures that are similar to this one, and that would provide the desired comparison between versions. Most of the work researched quotes precision (and recall where possible) for evaluation, and these figures are included here for general comparison. Some (e.g. Smeaton and Quigley (1996)) also include relevance judgments from evaluators, on a scale of 1 to 5 or similar. Almost all similar evaluations rely on human judgment, with the “yes”, “no”, “maybe” method used here being much commoner than a ranking scale. Flank (1998) uses a metric where partial matches are discounted if the system ranks them higher than full matches. Several (e.g. Ravichandran and Hovy (2002), Greenwood and Gaizauskas (2003)) use the Mean Reciprocal Rank score developed by the Text Retrieval community, but (as is pointed out by Pastra et al. (2003)) this relies on there being a pre-existing set of relevance judgments to compare against, which was not possible here. The scoring method used here served its purpose, which was to compare the six application versions against each other on the criteria the systems were intended to meet; basically giving the best scores to correct answers returned high in the list, whilst penalising over-answering.

Table 6.1 summarises the overall results. The performance of each of the six CANTRIP versions is averaged over the whole query set. The “Time” column gives the average response times in seconds. The scores just described are normalised to “accuracy” percentages by:

$$Accuracy = \frac{score_K + 20}{230}$$

	Time	Precision	Recall^a	$score_K$	Accuracy
NER+	5.77	66.88%	100.00%	100.25	52.28%
fulltext	1.96	48.13%	100.00%	72.69	40.30%
search engine	3.17	29.38%	75.00%	48.63	29.84%
baseline	1.63	37.50%	62.50%	40.13	26.14%
NER	5.00	31.56%	75.00%	39.06	25.68%
CBIR	7.98	60.28%	36.11%	23.63	18.97%

Table 6.1: Overall results summary

^aOver only 3 of the 16 result sets

where $score_K$ varies between

$$Min_{K_s} = 20 \times -1 = -20$$

and

$$Max_{K_s} = \sum_{n=1}^{20} n = 210$$

The three sets of evaluator marks were compared, noting any disagreements between them. Where two out of the three agreed, their mark was used. In the sole case (out of 1,147 marks) where there was a three-way split, “uncertain” was used. Response time was averaged over the 3 runs for each results set. The variation between evaluators is shown in Table 6.2. They disagreed on only 7.85% of the marks, most of the disagreements (56.67% of them) being where two of the three marked an image “wrong” and the third chose “uncertain”. Of the 25 where the majority were “uncertain”, all but one arose on the same query, number 6: “Cranes or bridges connected with the Arrol firm”.² There were only 14 cases, or 1.22% of the total, where the evaluators disagreed about a result being correct; generally it was easy to tell. Five of the 14 are actually mistakes on the part of the evaluator or their scribe (me), because they mark as wrong or uncertain a clearly correct image (e.g. the 3 pictures of Mavisbank House for Query 4) which had been marked correct by the same person in all the other version

²In this case the non-specialist marked items as wrong where Arrol’s was not explicitly mentioned, but both the experts chose “uncertain” when they were fairly sure from the picture that the crane or bridge depicted *was* an Arrol one, even if the text didn’t say so.

	Number	Proportion of Total	Proportion of Disputed
Total	1,147		
Disputed	90	7.85%	
“Wrong” chosen	51	4.45%	56.67%
“Uncertain” chosen	25	2.18%	27.78%
“Right” chosen	14	1.22%	15.56%
Corrected figure for last	9	0.78%	10.00%

Table 6.2: Inter-evaluator agreement

results. If these are taken out of the total, the evaluators disagreed about correct images in only 0.78% of cases. Thus, even with so few evaluators, the results can be considered reliable, as the scoring system overwhelmingly favours correct hits over wrong or uncertain ones.

6.3 Analysis

As Table 6.1 shows, the enhanced NER version performed best overall, by a significant margin. On the accuracy measure described above it doubled the baseline performance.

It was no surprise that the CBIR version did so badly. It had no choice but to work with the first image returned by NER+, and also the simple intersection of results didn’t favour it. This is not the best way to use CBIR tools. There are no formal results to support this contention, but it was clear from using the system that CBIR *can* be a useful aid to text-based retrieval, but *only* where the user actually wants results that are visually similar. The technology works well with plans and maps, and fairly well with colour photographs. To give an example: suppose the user is interested in henges. There are 114 digital image records where “henge” appears in the classification fields and 20 more where henges are mentioned in the text but not in the classification; which is too many to browse through with CANTRIP. Suppose further that the user is particularly interested in line drawings of henges: the NER+ version returns 7 of these amongst the first 20 images. If the user clicks on one of them and chooses the “Match

	Precision	Recall^a	<i>score_K</i>	Accuracy
NER	46.88%	100.00%	61.13	35.27%
NER+	46.25%	100.00%	59.50	34.57%
search engine	30.63%	100.00%	55.13	32.66%
fulltext	26.51%	100.00%	42.38	27.12%
baseline	37.50%	93.75%	26.88	20.38%
CBIR	33.06%	41.67%	16.13	15.71%

Table 6.3: Results for non-locational queries

^aOver only 2 of the 8 result sets

Image and Query” option from the pop-up window, further line drawings of henges will be returned that were not in the original set. In this case the image is suitable for content-based matching, and the text query has a large pool of good hits, so the combination works well. Try the same thing with a dim black and white photograph and you will be unimpressed.

What was disappointing in the overall results was that the basic NER version performed so poorly; slightly worse than the baseline. This seemed surprising because it is closest in methodology to NER+ which did so much better. The answer lies in the location problem described in Section 5.4. This seemed likely to be skewing the results against the four search engine based versions. A second analysis of the results was made, shown in Table 6.3, which excludes the location-based queries. Of the 16 queries, 8 used a location parameter and 8 did not.

This time, all the search engine versions, apart from the unfortunate CBIR, come top. The NER and NER+ versions are very close, but the former is slightly better. What’s immediately noticeable is that the overall performance is much worse. Only NER and the plain search engine actually improve their scores; all the rest plummet. Of course, with such a reduced test set individual query results may unbalance the averages.

It’s worth looking more closely at some of the individual queries (refer to the full results in Appendix H). Queries 1 and 4 (“Frank Matcham” and “Mavisbank House”) are examples of what’s quite a common query type: information on a single entity. As

is to be expected, the two NER versions both get top marks here. Each recognises the query string as a single entity token and simply returns the documents containing it (8 and 3 records respectively) and no others. The search engine also finds all of the records, but fills the rest of the slots up to the limit with irrelevant items, that contain one but not both of the query words. The baseline version also does well on these two queries, but I would suggest that's something of a coincidence; in general it will not find compound term entities easily.

These two queries also highlight a defect in the scoring system: the maximum score available for Query 4 is 57, because there are only three items to return ($57 = 20 + 19 + 18$). Thus although these scores could not be improved on, they fall well short of the 210 maximum score. The option of giving an arbitrary bonus award for 100% recall was considered, but seemed too unsafe. The recall figures are just not sufficiently reliable.

One of the difficulties of objective evaluation is that the methods simply work better for some queries than for others. Query 16 (“cropmarks of Roman camps”) is an example of where spotting an entity string did *not* help. The NER versions both return a lot of records on a colliery called “Roman Camps mine” and on the “Roman Camp Hotel” in Callander. (Neither of these is in the gazetteer in full as an entity in its own right, or of course they would have been ignored.) The plain search engine was the only version to score well here, managing to correctly balance occurrences of “Roman”, “camp” and “cropmark”. Quite why the other versions gave so little weight to the correctly spotted “crop[-_]?marks?” token is unclear, but may relate to problems with handling single-word entities.³

Query 2 (“social housing in Fife by Haxton and Watson”) is an interesting one. At first glance it seemed an ideal query, and it was somewhat disappointing that not a single good hit was returned by any of the systems. Checking the inverted indexes made the reason obvious in the case of the search engines: Haxton and Watson do not feature in it. The RCAHMS staff member who suggested this query knew that this

³It seems likely that single-word entities are not always correctly marked in the inverted index, because there is no easy way of distinguishing them from non-entity words. This would bear further investigation.

organisation name would be recorded in the “collection” field, since that was the source of the material. Unfortunately this field was not one of those included in the search. The “copyright” field is another that was omitted and that, although very sparsely populated, is useful on occasion. It would be easy to alter the package to include them.

Query 9 (“churches dedicated to St Columba”) produced surprising results: any number of churches, but almost no mention of St Columba. The contrast between NER and NER+ is interesting. It was almost certainly this query that allowed NER to pip NER+ in Table 6.3, because NER scores the maximum of 210, whilst NER+ gets the minimum: -20. Both correctly identify “St Columba” as an entity pattern (as “(saint|st|s)?.?_columba’?s?” in fact). However, the query expansion used by NER+ turns “church” (entered as “Type of site”) into a huge string of related terms: “church ritual_buildings? bell_towers? burial[-]?aisles? cathedral; (church|kirk)[-]?yards? collegiate_church lych_gate round_towers? shrine; steeple”. Unsurprisingly, poor old St Columba is hopelessly outnumbered in this collection of terms. The problem is not so much the query expansion itself — we *want* “St Columba’s Cathedral”, even if we may be less sure about the lych gate and the bell tower. The problem is likely to be with the weighting. All the alternative terms count equally in the system. On reflection, it would have been sensible to downgrade the weights on expanded terms. There are various plausible options (e.g. making the sum of the weights come to 1.5), but one would need to experiment to see what worked well.

Finally, there was one query that all the versions did well on: number 8 — “linoleum works in Fife”. None of the systems returned any misses on this, although it’s a location query. This is probably because Fife is where almost all the linoleum factories in Scotland are! The point to note here is that all the search engine versions improved recall, producing 20 good hits against the 3 from the baseline version.

One could pick up particular points on more of the 16 queries, which illustrates one of the problems with this task. The possible queries, and the distributions of the data to answer them, vary greatly. What works well for some queries will not suit others. It would be useful to do a much larger evaluation exercise, with far more queries; but the time needed would be considerable.

6.4 Contribution of NER

Another issue to consider is whether the effort to find NEs in the source text gives sufficient advantage to be worthwhile. For example, if one puts a quoted string like “Thomas Telford” into a Web search engine like Google, this will usually produce good results even though entities have not been marked in the source data. There are a number of points to make here:

1. Only an exact match between query string and source is possible with a quoted string, whereas NER techniques like those used in CANTRIP will code the entity itself and match multiple variants on how that entity is represented in query or text. (See discussion in Section 5.5.)
2. The quoted string technique works in Google, but only because mechanisms have been built to handle it. It was not part of the CANTRIP design. Less pre-processing of the source text is required but the architecture of the search engine is more complex and, more importantly, its performance will be slower than if it were doing a straight comparison of normalised tokens, as is possible if the entities are found in advance. Particularly for a database application, the source data is relatively static and easy to process off-line, so there is an advantage in doing the hard work in advance instead of at query time.
3. Although not used in this project, NER commonly includes classification. This makes it easy to disambiguate entity strings that are in different categories; for example to distinguish “Alexander Hall & Son” the organisation, from “Alexander Hall” the person; or to tell whether a reference to “Fife and Drum” is about places, people or musical instruments.
4. Related to the above, it may be possible to distinguish different entities that are in the same category, but this is more difficult and success rates may not justify the effort. An example in the NMRS database is to try to distinguish “Robert Adam” the architect from the staff member who happens to have the same name. It might be possible to separate these using NLP techniques, working on the context in which they appear. String matching would not tell them apart.

5. Query expansion depends to some extent on detecting NEs in the query. If the system can tell what entities the query refers to it can look them up in other resources such as the thesaurus used in CANTRIP. The user, even if expert enough to enclose significant strings in quotation marks, may not know what the entity terms for the domain actually are. For example, if a user quotes “coal mine” as part of a query in CANTRIP, it will be translated through query expansion to the preferred term “colliery”. The preferred term should be more likely to find a match than the user’s quoted string.
6. The quoted string approach will only work for compound terms, whereas NER methods will deal with single word entity terms as well.
7. Perhaps most importantly, the quoted string method requires special knowledge on the user’s part. It’s surprising how many people one comes across don’t know it works in Google — but why should they? The goal should be to produce a system that can cope with queries from non-expert users, with something at least approaching the competence of a human receiving the same query.

Chapter 7

Scope for Further Work

There are a great many loose threads to tweak in this project, and many have been noted on the way. This chapter draws together what seem the most important ones.

7.1 NE recognition and IDF weighting

The use of IR term weighting on NEs and significant phrases is at the core of the work, and there is a great deal more that could have been done. The main areas are:

1. One could carry on from where the project left off in identifying NEs. The gazetteer could be used to mark text which could then be used to train a recogniser like *candc*. It might be interesting to investigate using automatic collocation detection as part of the NER step, and perhaps to examine relative frequency ratios against different corpora, to find domain-specific terms. See Manning and Schütze (2002) pp 153-7 and 172-6 for discussion of these two ideas.
2. The handling of single word entities needs further work, as there is reason to suspect they are unbalancing the IDF weights at present. They need to be distinguished from non-entity words, which isn't always happening. This is tied up with the way the inverted index creation is done, as it deals with "words" with no embedded spaces.

3. Related to the previous point, the stemming algorithm doesn't cope well with the entity "words" (in which spaces are replaced by underscores). In general proper names and other NEs are not lemmatised, but this is can be a disadvantage.
4. So far, no entity categorisation has been thought necessary. The type of entity is irrelevant in the simple search engine retrieval being done. However, there seems scope for changing the interface to pick up ready-categorised entities from the user, for example through the "Who?", "What?", "Where?", "When?" idea mentioned in Section 5.1.
5. The extra weighting assigned to NEs could not be adequately tested in the evaluation, since it was mixed in with the query expansion and database intersection. This needs more examination. At present the weighting is fairly arbitrary and it's not clear whether it helps or not. Intuitively, it should, as the NEs are obvious content-carriers, but will have artificially low frequencies through being concatenations of other words.

7.2 Database related issues

One of the things the project tried to test was the difference between building inverted indexes on the entire text and on pre-categorised sections of it. This was done by comparing the MySQL fulltext indexes with the hand-built IR ones. However, the methods used to build the indexes were not the same, so the comparison was only indicative. It would be interesting to pursue this further, as using the power of the database structure is attractive, but there are questions about the size of corpus needed to make IDF weighting work properly.

The intersection of search engine results with database querying was another of the ideas under test. It had to be done by using the database to narrow the search engine results instead of the other way round, because clearly a separate inverted index couldn't be built for every possible subset of the database. Common sense suggests this could be a very simple and effective way of narrowing or "cleaning" the broad

search engine results, but at present the search engine often brings back *too* broad a selection. It needs to be refined first, and then more database intersections could be tried, using other fields than location and other join techniques than simple equality. The search engine can quickly, and it seems effectively, provide a selection of candidate documents which standard SQL could not easily find. The power of the database could then be used to rule out the spurious ones.

Some further pre-formatting of the source data might be useful, particularly in providing individual captions for all images. However, this isn't really the business of this project, which aimed to examine techniques that are generally applicable to any body of semi-structured data.

The mention of semi-structured data brings XML to mind — it is becoming the standard for exchanging and sometimes storing such data. Without any prior intention, XML became the medium for transferring blocks of data between tools in the application. The search engine tools required the data to be in XML format; so does TTT. There is nothing surprising about this; it illustrates the way XML¹ tends to turn up as a sort of glue between components. It is mentioned here because of another avenue there wasn't time to explore. There are tools such as SilkRoute (see Fernandez et al. (2002)) that allow a relational database to be presented as XML *in situ*, i.e. without any exporting of data or reformatting. My own slight experience of this software was through a small course-work project (Byrne (2003)). It would be interesting to see if a tool like this could be used to cut down the number of awkward data-dumping and reformatting steps. The project just mentioned showed that it might also provide a neat way of adding XML markup around named entities so they could be highlighted in the browser presentation, through XSL Transformations say. They could then be given hypertext references on the fly, to a stock piece of text describing the particular entity, be it a person such as “Greek Thomson”, or a piece of specialist terminology like “ogam-inscribed standing stone”.

¹...for all its many shortcomings, that will not be dwelt on here...

7.3 CBIR, design, and practicalities

As the project went on it concentrated more on text issues and less on CBIR techniques. This seems perfectly valid given the work was done under the auspices of the Language Technology Group. Clearly there is more that could be done on the image processing side. One route that could have been explored is image clustering; trying to pre-select groups of visually similar images by semantic content, tailored for the semantics of this domain.

Much of the work in the text retrieval field (see Chapter 3) has used logical relation structures to improve matching between query and source text. This was ruled out here because the aim was to use simple tools that didn't have to be hand-crafted to this dataset. (The gazetteer is domain specific, but it was extracted largely automatically.) However, the technique often produces good results and could be tried here.

Performance has barely been commented on, but clearly the more complex techniques take longer, as Table 6.1 shows. Eight seconds feels like a very long time to the average user. What slowed CANTRIP down very often was all the moving of separate indexes into position and comparison of large result sets, which wouldn't be done in a production application. Security, portability and multi-user issues are all important. They were borne in mind throughout but, as it stands, CANTRIP is not a robust enough specimen for the real world, nor was it designed as such.

Chapter 8

Conclusions

Despite the long list of loose ends in the last chapter, the project achieved its goals and produced results. One of the clear findings is that broad-brush IR techniques are effective against this kind of semi-structured data. For someone who has spent many years working with relational databases, this is almost a disappointment. It is clear that *if* the database fields include simple, cut-and-dried classification data items (like the Council code) these will always provide the simplest way to get a definite “in or out” decision about candidates for the result set. The interesting thing about this data, and all similar text-rich datasets that have accreted over many years, is that such clear-cut classifications are often impossible. The combination of different techniques therefore seems promising:

- identify important entities and “concept terms” throughout the text;
- ignore the database structure and use statistical term weighting across the whole text to find candidate answers;
- weed out the spurious ones with the precision of SQL selection against the structured fields.

Lest we get carried away, it must be pointed out that the overall accuracy of the system was quite poor! As discussed in Chapter 6, accuracy is a slippery thing to measure in database retrieval. Nobody can be completely sure what the database contains, and

many queries are open to different interpretations. But against the standard chosen, the best result was only 52% average accuracy. On simple queries with definite answers, like finding the Frank Matcham drawings, the system does much better than this, but so would any application. It's the hard queries that are more interesting, and results on these are still too hit and miss. With more work performance could certainly be improved, but it's not clear it could be made good *enough*. One solution is to restrict the range of queries that can be answered; this is often done, using fixed database fields in the interface. Another is to treat the application the way we are used to treating Internet search engines. Nobody expects 100% precision or recall from them, and frequent users quickly find that the way the query is posed makes all the difference to the response; one learns from experience what works and what doesn't. One day these engines may be able to formulate what the query and data are actually "about". It's an exciting goal to aim towards.

The CANTRIP version that combined CBIR with text methods was not successful. This was not a surprise, for the reasons presented in Chapter 6. However, as that chapter also pointed out, CBIR tools *were* effective when the image and query both met certain conditions. There has been no attempt in this project to improve on the CBIR methodology; this was outside the project's scope. At present, CBIR tools are not straightforwardly applicable to such heterogeneous collections as the NMRS, with such a high proportion of monochrome images.

Despite some imperfections, the project succeeded in improving over the baseline system that used only standard database indexing. It also used real, untailed data rather than an artificially constructed or particularly well-ordered dataset — for which scores would almost certainly have been higher. The practical applicability has been highlighted throughout, as methods that work in the untidy real world seem of more interest than ones which only perform well in controlled conditions.

Appendix A

Abbreviations used

CANTRIP	a Computer Application for the NMRS, To Retrieve Its Pictures
CBIR	Content-based Image Retrieval
CoNLL	Computational Natural Language Learning (conference)
CFG	Context Free Grammar
DFV	Document Frequency Vector
DOM	Document Object Model (in XML)
DTD	Document Type Definition (in XML)
GIS	Geographical Information System
HTML	HyperText Markup Language
IBM	International Business Machines
IE	Information Extraction
IR	Information Retrieval
LTG	Language Technology Group (Edinburgh University)
MUC	Message Understanding Conference
NE	Named Entity
NER	Named Entity Recognition
NLP	Natural Language Processing
NMRS	National Monuments Record of Scotland
OCR	Optical Character Recognition

PHP	PHP: Hypertext Processor [recurse at will!]
POS	Part Of Speech
RCAHMS	Royal Commission on the Ancient and Historical Monuments of Scotland
RDBMS	Relational Database Management System
SEER	Stanford-Edinburgh Entity Recognition project
SQL	Structured Query Language
TF-IDF	Term Frequency — Inverse Document Frequency
TREC	Text REtrieval Conference
TTT	Text Tokenisation Tool
XML	eXtended Markup Language
XSL	eXtensible Stylesheet Language

Appendix B

Database setup, SQL code

B.1 Main database tables, with indexes

```
create table rcmmain (
  NUMLINK NUMERIC (8,0) PRIMARY KEY,
  TEXT_KEY NUMERIC (12,0),
  MAPNO VARCHAR (6) NOT NULL,
  SITE NUMERIC (4,0) NOT NULL,
  SUB NUMERIC (2,0),
  NMRSNAME VARCHAR (250),
  PARISH VARCHAR (41),
  LASTUPDATE DATETIME,
  ENTRYDATE DATETIME,
  NGRE VARCHAR (5),
  NGRN VARCHAR (5),
  COUNCIL NUMERIC (3,0),
  COUNTY VARCHAR (2),
  DISTRICT VARCHAR (2),
  REGION VARCHAR (2),
  LINEAR VARCHAR (8),
  PERIOD VARCHAR (15),
  ARCHAEOLOGY VARCHAR (1),
  ARCHITECTURE VARCHAR (1),
  CLASSGRP VARCHAR (80),
  CLASS VARCHAR (80),
  CLASSSUB VARCHAR (100),
  ALTNAME VARCHAR (150),
  GISIND VARCHAR (1),
  STATUS VARCHAR (4),
  UPD_BY VARCHAR (15),
  ENTER_BY VARCHAR (15),
  RCMSGRADE VARCHAR (2),
  RCGRADATE VARCHAR (4),
  SMRREF VARCHAR (3),
  SMRREF2 VARCHAR (5),
  FORM VARCHAR (7),
  OTHSTAT VARCHAR (40),
  MDESC VARCHAR (50),
  KMSQUARE VARCHAR (6),
  C14DATES VARCHAR (100),
  ARCDATES VARCHAR (100),
```

```

LATHEM          VARCHAR (1),
LONGHEM        VARCHAR (1),
LATDEG         NUMERIC (2,0),
LATMIN         NUMERIC (6,4),
LONGDEG        NUMERIC (2,0),
LONGMIN        NUMERIC (6,4),
HEIGHT         NUMERIC (6,2),
HT_DATUM       VARCHAR (4),
ACCURACY       VARCHAR (1),
X              NUMERIC (11,4),
Y              NUMERIC (11,4),
SL             VARCHAR (2),
COUNTRYIND    VARCHAR (1),
FULLTEXT (nmrsname, altname),
FULLTEXT (class, classgrp, classsub),
INDEX (parish),
INDEX (council),
INDEX (county),
INDEX (region, district),
INDEX (period),
INDEX (class(50)),
INDEX (classgrp(50)),
INDEX (classsub(40)),
INDEX (nmrsname(30)),
INDEX (altname(30))
);

create table rctextrep (
  NUMLINK          NUMERIC (8,0) NOT NULL,
  SECTION         VARCHAR (1) NOT NULL default 1,
  TEXT_KEY        NUMERIC (12,0),
  REPDATE         DATETIME,
  REPORT          LONGTEXT,
  PRIMARY KEY (numlink, section),
  FULLTEXT (report)
);

create table rcasp (
  NUMLINK          NUMERIC (8,0) PRIMARY KEY,
  ASPNOTES        LONGTEXT,
  NOTEDATE        DATETIME,
  FULLTEXT (aspnotes)
);

create table rcshortref (
  BIBNO           NUMERIC (6,0) NOT NULL,
  NUMLINK         NUMERIC (8,0) NOT NULL,
  PAGENOS         VARCHAR (67),
  REFDESC        VARCHAR (40),
  PRIMARY KEY (bibno, numlink),
  INDEX (numlink)
);

create table rcbibliog (
  BIBNO           NUMERIC (6,0) PRIMARY KEY,
  NAME            VARCHAR (69),
  INITIALS        VARCHAR (25),
  BIBDATE        VARCHAR (9),
  SUFFIX          VARCHAR (2),

```

```

JOURNAME          VARCHAR (72),
SERIES            VARCHAR (15),
VOLUME           VARCHAR (14),
PART             VARCHAR (7),
VOLDATE          VARCHAR (10),
BIBDESC          VARCHAR (140),
PAGENOS          VARCHAR (72),
LIBNUM           VARCHAR (20),
LOCATION           VARCHAR (6),
REFRECD          VARCHAR (1),
WHEREPUB         VARCHAR (80),
EDITION          VARCHAR (11),
COLLWORK         TEXT,
EDITOR           VARCHAR (71),
PUBLISHER        VARCHAR (80),
PUBDATE          NUMERIC (4,0),
ISBN             VARCHAR (16),
LASTUPDATE       DATETIME,
SECTION          VARCHAR (1),
CATEGORY         NUMERIC (2,0),
SUBCAT           NUMERIC (2,0),
PAMPHLET         VARCHAR (1),
NOTES            TEXT,
TITLE1           TEXT
);

create table rccollect (
  ARCNUMLINK      NUMERIC (8,0) PRIMARY KEY,
  TEXT_KEY        NUMERIC (12,0),
  PREFIX          VARCHAR (3),
  ARCHNUM         VARCHAR (20),
  SUFFIX          VARCHAR (8),
  NUMITEMS        NUMERIC (6,0),
  ORIGNMRS        NUMERIC (8,0),
  CATEGORY        VARCHAR (1),
  LOCATION        VARCHAR (1),
  SIGNED          VARCHAR (50),
  ARCDATE         VARCHAR (20),
  COPYRIGHT        VARCHAR (80),
  CONDITION        VARCHAR (1),
  COLLECTION      VARCHAR (66),
  SCALE           VARCHAR (15),
  REF             VARCHAR (20),
  PAPSIZE         VARCHAR (15),
  ACCNO           VARCHAR (20),
  PERMISSION      VARCHAR (1),
  STORE           VARCHAR (25),
  COPYDATE        VARCHAR (20),
  DMEDIUM        VARCHAR (50),
  DESC1           TEXT,
  NOTES           TEXT,
  SURNAME         VARCHAR (66),
  INITS           VARCHAR (15),
  ENTER_BY        VARCHAR (15),
  UPD_BY          VARCHAR (15),
  ENTRYDATE       DATETIME,
  LASTUPDATE      DATETIME,
  SURVEY_IND      VARCHAR (1),
  TBNUM           NUMERIC (8,0),
  HLF_IND         VARCHAR (1),
  SAPP_IND        VARCHAR (1),
  FLAT            VARCHAR (1),

```

```

ROLLED                                VARCHAR (1),
CIRCA_DATE                            VARCHAR (1),
ARCHIVE_WRAP_ONLY                     VARCHAR (1),
FROM_DAY                              NUMERIC (2,0),
FROM_MON                              NUMERIC (2,0),
FROM_YEAR                             NUMERIC (4,0),
T_DAY                                 NUMERIC (2,0),
T_MON                                 NUMERIC (2,0),
T_YEAR                               NUMERIC (4,0),
AUDIT_2001                            VARCHAR (1),
SITE_IND                              VARCHAR (1),
INDEX (desc1(30)),
FULLTEXT (desc1)
);

create table rccollection_essay (
COLLECTION                            VARCHAR(66) NOT NULL,
HISTORY                              TEXT,
PROVENANCE                            TEXT,
SCOPE                                 TEXT,
REFS                                  TEXT,
CONSERVATION                          TEXT,
ACC_REF                               VARCHAR(100)
);

create table rcpeople (
PCODE                                NUMERIC(8,0) PRIMARY KEY,
TYPE                                  VARCHAR(1),
FORENAME                             VARCHAR(25),
SURNAME                              VARCHAR(40),
ORGANISATION                          VARCHAR(80),
NOTES                                 TEXT
);

create table rcarcref (
NUMLINK                              NUMERIC(8,0) NOT NULL,
ARCNUMLINK                            NUMERIC(8,0) NOT NULL,
NOTE                                  VARCHAR(25),
PRIMARY KEY (numlink, arcnumlink),
INDEX (arcnumlink)
);

create table rccollections (
COLLECTION                            VARCHAR(66) PRIMARY KEY
);

```

B.2 Location lookup tables

```

create table rccouncil (
COUNCIL                              NUMERIC(3,0) PRIMARY KEY,
COUNAME                              VARCHAR(25)
);

create table rccolkup (
COCODE                               VARCHAR(2) PRIMARY KEY,
COTEXT                               VARCHAR(20)
);

```

```

);

create table rcparkup (
  PARISH          VARCHAR(41) PRIMARY KEY,
  COCODE         VARCHAR(2),
  REGCODE        VARCHAR(2),
  DISTCODE       VARCHAR(2),
  COUNCIL        NUMERIC(3,0)
);

create table rcdistlkup (
  REGION          VARCHAR(2) NOT NULL,
  DISTRICT        VARCHAR(2) NOT NULL,
  DISTTEXT       VARCHAR(25),
  PRIMARY KEY(region, district)
);

create table rcreglkup (
  REGION          VARCHAR(2) PRIMARY KEY,
  REGTEXT        VARCHAR(26)
);

```

B.3 CANTRIP table and index creation

```

create table cantrip (
  canrowid decimal(8,0) not null,
  arcnumlink decimal(8,0) not null,
  numlink decimal(8,0),
  imagefile decimal(8,0) not null,
  descl text,
  nmrsname varchar(250),
  altname varchar(150),
  class varchar(80),
  classgrp varchar(80),
  classsub varchar(100),
  parish varchar(41),
  council decimal(3,0),
  period varchar(15),
  report longtext,
  aspnotes longtext
);

insert into cantrip
select 0,
       c.arcnumlink, m.numlink, imagefile, descl, nmrsname, altname, class,
       classgrp, classsub, parish, council, period, report, aspnotes
from rccollect c
left join rcarcref a on (c.arcnumlink=a.arcnumlink)
left join rcmain m on (a.numlink=m.numlink)
left join rctextrep t on (m.numlink=t.numlink)
left join rcasp on (m.numlink=rcasp.numlink);

alter table cantrip add primary key (canrowid);

alter table cantrip add unique key (arcnumlink, numlink);

alter table cantrip add index (parish);

alter table cantrip add index (council);

```

```
alter table cantrip add index (period);
alter table cantrip add index (class(50));
alter table cantrip add index (classgrp(50));
alter table cantrip add index (classsub(40));
alter table cantrip add index (nmrsname(30));
alter table cantrip add index (altname(30));
alter table cantrip add index (descl(30));
alter table cantrip add index (imagefile);
alter table cantrip add fulltext (nmrsname, altname);
alter table cantrip add fulltext (class, classgrp, classsub);
alter table cantrip add fulltext (descl);
alter table cantrip add fulltext (aspnotes);
alter table cantrip add fulltext (report);
```

Appendix C

CANTRIP application, sample PHP code

The CANTRIP application contains 10 PHP programs:

1. `cantrip_frm`: The user query form.
2. `cantrip_pick`: A version selection mechanism.
3. `cantrip_base`: Baseline version.
4. `cantrip_full`: MySQL fulltext index version
5. `cantrip_se`: Search engine version
6. `cantrip_ner`: NER version
7. `cantrip_nerw`: NER+: entity weighting, query expansion and database intersection.
8. `cantrip_cbir`: CBIR version: as NER+ but intersected with image content results.
9. `cantrip_2_cbir1`: Routine to match an individual image from the pop-up window.
10. `cantrip_2_cbir2`: Matching pop-up item on image *and* query.

The code is too lengthy to print in full (but is available if wanted). Two programs are listed here: the simplest (`cantrip_base`) and most complicated (`cantrip_cbir`) versions of the query mechanism.

C.1 Baseline version

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<HTML>
  <HEAD>
    <TITLE>Lookup NMRS Images - BASELINE version of application</TITLE>
  </HEAD>
  <STYLE TYPE="text/css">
    <!--
      BODY {font-family: sans-serif}
      TD {font-family: sans-serif}
      CAPTION {font-family: sans-serif}
      P {font-family: sans-serif}
    -->
  </STYLE>
```

```

</HEAD>
<BODY BGCOLOR="#ffd700">

<?php

// MAIN PROGRAM
// THIS VERSION USES STANDARD LEFT ANCHORED INDEXES IN MYSQL.
// TO MAKE USE OF THE INDEXES THE USER ENTRIES ARE ONLY FOLLOWED BY
// WILDCARDS, NOT PRECEDED. THE EXCEPTION IS THE KEYWORD SEARCH, WHICH
// THEREFORE WON'T USE AN INDEX.
// Kate Byrne, July 2003.

// TIMING CHECK
$starttime = time();

// FORM ENTRIES FROM USER ALREADY CHECKED, BY cantrip_pick.php

// CONNECT TO DATABASE

$dblink = mysql_pconnect("localhost", "nobody") or die("<br><br><h1>COULD NOT CONNECT TO DATABASE</h1>");
$rcdb = mysql_select_db("rcdb",$dblink) or die("<br><br><h2>PROBLEM SELECTING rcdb DATABASE");

// SET LIMIT ON NUMBER OF ROWS TO BE RETURNED BY SQL QUERY (SIGNIFICANT
// EFFECT ON PERFORMANCE). IF NO LIMIT WANTED, SET $limit = "".
$limit = "limit 20";
$sprintlim = 20; // ONLY USED IN DISPLAYING OUTPUT;
// IGNORED IF $limit = ""

// BUILD THE WHERE CLAUSES

// SPACES ADDED AFTER QUERY WORDS, ALLOWING WHOLE-WORD MATCHES ONLY. THIS
// PRODUCES BETTER RESULTS IN GENERAL, FOR THIS APPLICATION VERSION.

$where1 = "";
$where2 = "";
$where4 = "";
$where8 = "";
$where16 = "";

$where = "";

// DESCR1
if ($descl != "") {
    $qelts = $qelts + 1;

    $bits = split(" ", $descl);
    if (count($bits) == 1) {
        $where1 = "descl like '$descl%'";
    } elseif (count($bits) > 1) {
        $str = "";
        for ($n=0; $n<count($bits); $n++) {
            $str = $str . $bits[$n] . "%";
        }
        $where1 = "descl like '$str%'";
    }
}

// NMRSNAME
if ($nmrsname != "") {
    $qelts = $qelts + 2;

    $bits = split(" ", $nmrsname);
    if (count($bits) == 1) {
        $where2 = "(nmrsname like '$nmrsname%' or altname like '$nmrsname%'";
    } elseif (count($bits) > 1) {
        $str = "";
        for ($n=0; $n<count($bits); $n++) {
            $str = $str . $bits[$n] . "%";
        }
        $where2 = "(nmrsname like '$str' or altname like '$str')";
    }
}

// CLASSIFICATION FIELDS
if ($stype != "") {
    $qelts = $qelts + 4;

    $bits = split(" ", $stype);
    if (count($bits) == 1) {
        $where4 = "(class like '$stype%' or classgrp like '$stype%' or classsub like '$stype%'";
    }
}

```

```

    } elseif (count($bits) > 1) {
        $str = "";
        for ($n=0; $n<count($bits); $n++) {
            $str = $str . $bits[$n] . "%";
        }
        $where4 = "(class like '$str' or classgrp like '$str' or classsub like '$str')";
    }
}

// LOCATION
if ($loc != "") {
    $qelts = $qelts + 8;
    $where8 = "council=$loc";
}

// KEYWORD - DOESN'T SEARCH REPORT AND ASPNOTES (TOO SLOW); DOESN'T USE
// INDEX ON desc1 (WILDCARD AT FRONT).
if ($keyword != "") {
    $qelts = $qelts + 16;

    $bits = split(" ", $keyword);
    if (count($bits) == 1) {
        $where16 = "(desc1 like '%$keyword%' or period like '$keyword%')";
    }
    } elseif (count($bits) > 1) {
        $str = "";
        for ($n=0; $n<count($bits); $n++) {
            $str = $str . $bits[$n] . "%";
        }
        $where16 = "(desc1 like '%$str' or period like '$str')";
    }
}

// BUILD UP THE FINAL QUERY. NEED TO GET 'where' AND 'and' ENTRIES RIGHT IN
// WHERE CLAUSE.
switch ($qelts) {
    case 1:
        $where = "where $where1";
        break;
    case 2:
        $where = "where $where2";
        break;
    case 3:
        $where = "where $where1 and $where2";
        break;
    case 4:
        $where = "where $where4";
        break;
    case 5:
        $where = "where $where1 and $where4";
        break;
    case 6:
        $where = "where $where2 and $where4";
        break;
    case 7:
        $where = "where $where1 and $where2 and $where4";
        break;
    case 8:
        $where = "where $where8";
        break;
    case 9:
        $where = "where $where1 and $where8";
        break;
    case 10:
        $where = "where $where2 and $where8";
        break;
    case 11:
        $where = "where $where1 and $where2 and $where8";
        break;
    case 12:
        $where = "where $where4 and $where8";
        break;
    case 13:
        $where = "where $where1 and $where4 and $where8";
        break;
    case 14:
        $where = "where $where2 and $where4 and $where8";
        break;
    case 15:
        $where = "where $where1 and $where2 and $where4 and $where8";
        break;
    case 16:
        $where = "where $where16";
}

```

```

        break;
    case 17:
        $where = "where $where1 and $where16";
        break;
    case 18:
        $where = "where $where2 and $where16";
        break;
    case 19:
        $where = "where $where1 and $where2 and $where16";
        break;
    case 20:
        $where = "where $where4 and $where16";
        break;
    case 21:
        $where = "where $where1 and $where4 and $where16";
        break;
    case 22:
        $where = "where $where2 and $where4 and $where16";
        break;
    case 23:
        $where = "where $where1 and $where2 and $where4 and $where16";
        break;
    case 24:
        $where = "where $where8 and $where16";
        break;
    case 25:
        $where = "where $where1 and $where8 and $where16";
        break;
    case 26:
        $where = "where $where2 and $where8 and $where16";
        break;
    case 27:
        $where = "where $where1 and $where2 and $where8 and $where16";
        break;
    case 28:
        $where = "where $where4 and $where8 and $where16";
        break;
    case 29:
        $where = "where $where1 and $where4 and $where8 and $where16";
        break;
    case 30:
        $where = "where $where2 and $where4 and $where8 and $where16";
        break;
    case 31:
        $where = "where $where1 and $where2 and $where4 and $where8 and $where16";
    }

    //$sqlquery = "select c.arcnumlink, imagefile, desc1, nmrsname, class, classgrp, classsub, parish, period, report, aspnates from rccollect c
    left join rcarceref a on (c.arcnumlink=a.arcnumlink) left join rmain m on (a.numlink=m.numlink) left join rtextrep t on (m.numlink=t.numlink)
    left join rcasp on (m.numlink=rcasp.numlink) $where $limit";

    $sqlquery = "select canrowid, arcnumlink, imagefile, desc1, nmrsname, class, classgrp, classsub, parish, period, report, aspnates
    from cantrip $where $limit";

    // FOR DEBUGGING
    //print $sqlquery;

    // RUN THE QUERY
    $res = mysql_query($sqlquery);

    if(!$res) {
        echo "<br>Error while querying the database<br>";
    }
    $nr = mysql_num_rows($res);

    // TIMING CHECK
    $endtime = time();
    $diff = $endtime - $starttime;
    if ($diff <= 1) {
        $elapsed = "1 sec";
    } else {
        $elapsed = "$diff secs";
    }

    // OUTPUT RESULT

    //print "<A HREF=\"#\" onclick=\"window.history.go(-2)\">Back</A>";
    print "<P ALIGN=\"right\"><A HREF=\"cantrip_frm.php\"><IMG src=\"wizard60.png\" alt=\"New Query\"><BR><FONT size=2>New Query</FONT></A></P>";
    print "<h2>";
    print $nr;
    if ($nr == 1) {
        print " hit ($elapsed):</h2>";
    } else {

```

```

    print " hits, in order of relevance ($elapsed):</h2>";
}
if ($limit != "") {
    print "(Number of returns is limited to $printlim)";
}

// FIVE IMAGES TO A ROW, IN TABLE.

print "<CENTER><TABLE cellspacing=20 cellpadding=0 border=0><TR>";

// LOOP THROUGH RESULTS ARRAY
for ($n=1; $n<=$nr; $n++) {
    $row = mysql_fetch_array($res);

    // ROW OF 5 IMAGES

    $imgfile = $row['imagefile'];
    $imgsize = GetImageSize("../med_images/$imgfile.jpg");
    $high = $imgsize[1] + 200;
    $wide = $imgsize[0] + 25;

    $canid = $row['canrowid'];
    $caption = $row['descl'];
    $name = $row['nmrsname'];
    $class = $row['class'] . " - " . $row['classgrp'] . " - " . $row['classsub'];
    $per = $row['period'];
    $par = $row['parish'];
    $rep = nl2br($row['report']);
    $asp = nl2br($row['aspnotes']);

    // TEMP FILE NAME: USE canrowid TO ENSURE UNIQUENESS
    $tmpfile = "tmpfiles/$canid.html";

    // PREPARE CONTENTS OF POP-UP WINDOW, INCLUDING CBIR BUTTON

    // NOTE: REQUIRES tmpfiles DIRECTORY THAT nobody CAN WRITE TO -
    $fp = fopen($tmpfile, w);

    fwrite($fp, "<HTML><BODY><IMG src='../med_images/$imgfile.jpg'>
    $caption<br><br>
    <form name=cbl method=post action=../cantrip_2_cbir1.php target=_child>
    <INPUT type=hidden name=seedimg value=$imgfile>
    <INPUT type=hidden name=seedarcnum value=$row[arcnumlink]>
    <INPUT type=submit value='Match Image'>
    <INPUT type=button value='Close' onclick=window.close()>
    </form>
    <br>
    <br>Parent site: <b>$name</b><br><br>Classification: $class
    <br>Period: $per<br>Parish: $par<br><br>Notes: <br>$rep<br>
    <br>ASP notes: <br>$asp<br></BODY></HTML>");

    fclose($fp);

    // BIT OF JAVASCRIPT TO DISPLAY IMAGE AND DESCRIPTION IN POP-UP WINDOW
    print "<TD align=center valign=top><A href='#' onclick='window.open('$tmpfile', 'details', 'height=$high,width=$wide,scrollbars=yes');'>";

    // DISPLAY THUMBNAIL IMAGE
    print "<IMG SRC='../thumbs/$imgfile.jpg'></A><BR>";

    // DISPLAY arcnumlink FOR REFERENCE
    print $row['arcnumlink'];
    // print " ($imgfile)";
    print "<br>($canid)";
    print "</TD>";

    // ROW BREAK AFTER EVERY FIVE ITEMS
    if ($n%5 == 0) {
        print "</TR><TR>";
    }
}

print "</TR></TABLE></CENTER>";

?>

</BODY></HTML>

```

C.2 CBIR version

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">

<HTML>

  <HEAD>
    <TITLE>Lookup NMRS Images - NLP and CBIR combined version of application</TITLE>

  <STYLE TYPE="text/css">
<!--
    BODY {font-family: sans-serif}
    TD {font-family: sans-serif}
    CAPTION {font-family: sans-serif}
    P {font-family: sans-serif}
-->
  </STYLE>
</HEAD>
  <BODY BGCOLOR="#ffd700">

  <?php

  // MAIN PROGRAM
  // THIS VERSION CALLS THE FOA SEARCH ENGINE TO ANSWER THE QUERY, THEN
  // COMBINES ITS RESULTS WITH THOSE FROM CBIR SEARCHING.
  // Kate Byrne, August 2003.

  // TIMING CHECK
  $starttime = time();

  // FORM ENTRIES FROM USER ALREADY CHECKED, BY cantrip_pick.php

  // CONNECT TO DATABASE

  $dblink = mysql_pconnect("localhost","nobody") or die("<br><br><h1>COULD NOT CONNECT TO DATABASE");
  $rcdb = mysql_select_db("rcdb",$dblink) or die("<br><br><h2>PROBLEM SELECTING rcdb DATABASE");

  // NOTE: LIMIT ON NUMBER OF ROWS TO BE RETURNED BY QUERY IS HARD-CODED
  // IN foa.mp3.SearchEngine.outPartialRank() METHOD. TO CHANGE IT, ALTER
  // SearchEngine.java AND RECOMPILE.
  // ALSO NEED TO CHANGE DISPLAY ON RESULTS PAGE, SO USING VARIABLE HERE FOR
  // CONVENIENCE.
  // SET HIGHER IN SearchEngine, BUT LIMIT NUMBER DISPLAYED HERE. (NEED REST
  // FOR CBIR VERSION.)
  $limit = "limit 20";
  $printlim = 20;          // ONLY USED IN DISPLAYING OUTPUT;
                        // IGNORED IF $limit = ""

  // EXPAND type QUERY TERM FROM THESAURUS
  if ($type > "") {
    $typeqry = "select preferred, term, usefor, related from thesaurus where term like '$type%'";
    $res = mysql_query($typeqry);
    for ($i=0; $i < count($res); $i++) {
      $row = mysql_fetch_array($res);
      $newterms = " ". $row['preferred'] . " ". $row['term'] . " ". $row['usefor'] . " ". $row['related'] . " ";
      //print $newterms;
      $expandtype = $expandtype . $newterms;
    }
  }

  // PREPARE THE QUERY FILE USED BY THE SEARCH ENGINE, AND PUT IT
  // IN THE nob DIRECTORY THAT'S WRITABLE BY USER nobody.

  $fpw = fopen("foa_dir/data/nob/cantrip_queryNER", w);
  fwrite($fpw, " $desc1 $nmrsname $type $expandtype $keyword " );
  // (SPACES AT START AND END ARE REQUIRED.)

  fclose($fpw);

  // CALL NEMarker.java TO MARK UP NER STRINGS IN THE QUERY
  exec('/usr/java/j2sdk1.4.0_01/bin/java -cp $CLASSPATH:foa_dir:foa_dir/jar_files foa.ner.NEMarker', $ne_res, $return_val);

  if ($return_val > 0) {
    echo "Error in running NEMarker command<br>";
  }

  // REWRITE THE NEW VERSION OF THE QUERY
  $newquery = "";
  for ($i=0; $i < count($ne_res); $i++) {
    $newquery = $newquery . $ne_res[$i] . " ";
  }

```

```

$fpw = fopen("foa_dir/data/nob/cantrip_queryNER", w);
fwrite($fpw, "{Q1 $newquery}");
fclose($fpw);

// CALL THE SEARCH ENGINE; IT WRITES RESULTS BACK TO A FILE. REMOVE THAT FILE
// FIRST.

if (is_file("foa_dir/data/nob/resultsNER.data")) {
    unlink("foa_dir/data/nob/resultsNER.data");
}

// COPY THE WEIGHTED VERSION OF THE NER INVERTED INDEX FILE INTO POSITION.
// NEEDS TO GO IN THE DIRECTORY nobody CAN WRITE TO
if (!copy("foa_dir/data/invertedIndexNER.weighted",
"foa_dir/data/nob/invertedIndexNER.data")) {
    print "Copying index file failed<br>";
    return;
}

// NOTE THE NEED FOR EXPLICIT PATHS ON EVERYTHING, INCLUDING 'java'!!! WHAT A
// LOT OF TIME IT TOOK TO DISCOVER THAT...
// TRIED USING ENV VARIABLE TO MAINTAIN CODE PORTABILITY, BUT DOESN'T WORK.
// OUGHT TO FIX....

//exec('$JAVA_HOME/bin/java -cp $CLASSPATH:foa_dir:foa_dir/jar_files foa.mp3.SearchEngineNER', $foares, $return_val);
exec('/usr/java/j2sdk1.4.0_01/bin/java -cp $CLASSPATH:foa_dir:foa_dir/jar_files foa.mp3.SearchEngineNER', $foares, $return_val);

if ($return_val > 0) {
    echo "Error in running FOA command<br>";
}

// READ THE SEARCH ENGINE RESULTS FILE INTO ARRAY se_res
// FORMAT OF RESULTS FILE IS: <DOC_ID> <SCORE> PAIRS ON EACH LINE - ONLY
// WANT THE ID (CORRESPONDING TO COLUMN canrowid IN THE DATABASE).
$fpr = fopen("foa_dir/data/nob/resultsNER.data", r);
$i = 0;
while ($result = fgetcsv($fpr,50," ")) {
    $se_res[$i] = $result[0];
    $i++;
}

//print count($se_res)." entries in se_res array<br>";
// CHECK THERE ACTUALLY WERE SOME RESULTS
$gotsome = "y";
if ($i == 0) {
    $gotsome = "n";
}

// IF $loc PRESENT, TURN RESULTS INTO A STRING AND COMPARE WITH DATABASE LOCS
if ($loc > 0 and $gotsome == "y") {
    $se_res_str = "(";
    $last = count($se_res) - 1;
    for ($j=0; $j < $last; $j++) {
        $se_res_str = $se_res_str . $se_res[$j] . ", ";
    }
    $se_res_str = $se_res_str . $se_res[$last] . ")";

    // FIND RESULTS THAT DON'T HAVE CORRECT LOCATION
    $locqry2 = "select canrowid from cantrip where canrowid in $se_res_str and (council != $loc or council is null)";
    //print $locqry2;
    $badlocs = mysql_query($locqry2);
    $countres = mysql_num_rows($badlocs);
    //print "$countres dud locs<br>";

    // IF DUFF RESULTS FOUND, DELETE THEM FROM se_res ARRAY. (DOING IT THIS WAY
    // AS IT'S IMPORTANT NOT TO LOSE THE ORDERING OF THE ARRAY.)
    if ($countres > 0) {
        for ($k=0; $k < $countres; $k++) {
            $row = mysql_fetch_array($badlocs);
            $duffid = $row['canrowid'];
            $found = array_search($duffid, $se_res);
            if ($found >= 0) {
                //print "$se_res[$found]<br>";
                unset($se_res[$found]);
            }
        }
    }
}

// USE THE TOP IMAGE RETURNED BY SEARCH ENGINE AS SEED FOR CBIR SEARCH
// (NEED TO TRANSLATE canrowid TO imagefile FIRST)

// GET THE SEED IMAGE ID (NOT NECESSARILY INDEX 0, BECAUSE OF DELETIONS)

```

```

if ($gotssome == "y") {
    $stopping = current($se_res);
    $sqlquery = "select imagefile from cantrip where canrowid = $stopping";
    $result = mysql_query($sqlquery);
    $row = mysql_fetch_array($result); // QUERY WILL RETURN ONLY ONE ROW
    $seedimg = $row['imagefile'];

    // GET CBIR RESULTS (-n PARAMETER DETERMINES LIMIT ON RETURNS)
    unset($cbres); // MAKE SURE RESULTS ARRAY IS CLEAR
    exec("cbir/bin/retrieve -z cbir/ind/info -q $seedimg -n 300 -o 1 -i cbir/ind/index -p cbir/ind/picture.dat -t cbir/ind/texture.dat",
    $cbres, $return_val);

    if ($return_val > 0) {
        echo "Error in running CBIR command<br>";
    }

    // DISCARD LAST LINE OF CBIR OUTPUT, WHICH IS COMMENT
    $cbre_img = array_slice($cbres, 0, count($cbres)-1);

    // TRANSLATE IMAGE IDs RETURNED BY CBIR INTO CANROWIDS RETURNED BY SEARCH
    // ENGINE. (SIGH. NO OBVIOUS WAY ROUND THIS...)

    $i=0;
    foreach ($cbre_img as $line) {
        // FIRST FIELD IS IMAGE ID
        $currline = split(" ", $line);
        $imgid = $currline[0];

        $sqlquery = "select canrowid from cantrip where imagefile = $imgid";
        $result = mysql_query($sqlquery);
        // IT'S POSSIBLE TO HAVE MULTIPLE VALUES HERE (DIFFERENT TEXT AGAINST SAME
        // IMAGE). NEED TO TAKE ALL OF THEM, AS IT'S THE NUMBERS WE'RE USING TO
        // INTERSECT RESULT SETS.
        $num = mysql_num_rows($result);
        for ($j=0; $j < $num; $j++) {
            $row = mysql_fetch_array($result);
            $cbre_can[$i] = $row['canrowid'];
            $i++;
        }
    }
    //print count($cbre_can)." entries in cbre_can array<br>";

    // INTERSECT THE RESULT ARRAYS FROM SEARCH ENGINE AND CBIR, USING SEARCH
    // ENGINE ORDER. NOTE THIS IS OTHER WAY ROUND FROM cantrip_2_cbir2.php,
    // WHICH IS PRIMARILY CBIR, ASSISTED BY TEXT SEARCH. HERE WE HAVE PRIMARILY
    // A TEXT SEARCH, ASSISTED BY CBIR. IF TAKING SIMPLE INTERSECTION, IT ONLY
    // AFFECTS ORDERING.

    // array_intersect DOESN'T SEEM TO WORK PROPERLY; DOING BY HAND
    // $se = array_intersect($cbre_can, $se_res);
    $j = 0;
    foreach ($se_res as $canid) {
        if (in_array($canid, $cbre_can)) {
            $se[$j] = $canid;
            $j++;
        }
    }

    // DISCARD ENTRIES OVER THE LIMIT
    if (count($se) > $printlim) {
        $se = array_slice($se, 0, $printlim);
    }

    $nr = count($se);

    //print count($se)." entries in se array<br>";
}
else { // THERE WERE NO RESULTS ($gotssome WASN'T "y").
    $nr = 0;
}

// TIMING CHECK
$endtime = time();
$difftime = $endtime - $starttime;
if ($difftime <= 1) {
    $elapsed = "1 sec";
} else {
    $elapsed = "$difftime secs";
}

// OUTPUT INTERSECTED SEARCH ENGINE RESULTS
print "<P ALIGN=\"right\"><A HREF=\"cantrip_frm.php\"><IMG src=\"wizard60.png\" alt=\"New Query\"><BR><FONT size=2>New Query</FONT></A></P>";

```

```

print "<h2>";
print $nr;
if ($nr == 1) {
    print " hit ($elapsed):</h2>";
} else {
    print " hits, in order of relevance ($elapsed):</h2>";
}
if ($limit != "") {
    print "(Number of returns is limited to $printlim)";
}

// FIVE IMAGES TO A ROW, IN TABLE.

print "<CENTER><TABLE cellspacing=20 cellpadding=0 border=0><TR>";

$N = 0;

for ($i=0; $i<$nr; $i++) {

    $canid = $se[$i];

    // GATHER TEXT FOR THIS IMAGE FILE FROM THE DATABASE

    $sqlquery = "select canrowid, arcnumlink, imagefile, desc1, nmrsname, class, classgrp, classsub, parish, period, report, aspnotes
from cantrip where canrowid = $canid";

    $res2 = mysql_query($sqlquery);

    // canrowid IS UNIQUE, SO ONLY ONE ROW RETURNED
    $row = mysql_fetch_array($res2);

    // ROW OF 5 IMAGES

    $imgfile = $row['imagefile'];
    $imgsize = GetImageSize("../med_images/$imgfile.jpg");
    $high = $imgsize[1] + 200;
    $wide = $imgsize[0] + 25;

    $numlink = $row['numlink'];
    $caption = $row['desc1'];
    $name = $row['nmrsname'];
    $class = $row['class'] . " - " . $row['classgrp'] . " - " . $row['classsub'];
    $per = $row['period'];
    $par = $row['parish'];
    $rep = nl2br($row['report']);
    $asp = nl2br($row['aspnotes']);

    // TEMP FILE NAME: USE canrowid TO ENSURE UNIQUENESS
    $tmpfile = "tmpfiles/$canid.html";

    // PREPARE CONTENTS OF POP-UP WINDOW, INCLUDING CBIR BUTTON

    // NOTE: REQUIRES tmpfiles DIRECTORY THAT nobody CAN WRITE TO -
    $fp = fopen($tmpfile, w);

    fwrite($fp, "<HTML><BODY><IMG src=\"../med_images/$imgfile.jpg\">
    $caption<br><br>
    <form name=cbl method=post action=../cantrip_2_cbir1.php target=_child>
    <INPUT type=hidden name=seedimg value=$imgfile>
    <INPUT type=hidden name=seedarcnum value=$row[arcnumlink]>
    <INPUT type=submit value=\"Match Image\">
    <INPUT type=button value=\"Close\" onclick=window.close()>
    </form>
    <hr>
    <br>Parent site: <b>$name</b><br><br>Classification: $class
    <br>Period: $per<br>Parish: $par<br><br>Notes: <br>$rep<br>
    <br>ASP notes: <br>$asp<br></BODY></HTML>");

    fclose($fp);

    // BIT OF JAVASCRIPT TO DISPLAY IMAGE AND DESCRIPTION IN POP-UP WINDOW
    print "<TD align=center valign=top><A href=\"#\#\" onclick=\"window.open('$tmpfile', 'details', 'height=$high,width=$wide,scrollbars=yes');\">";

    // DISPLAY THUMBNAIL IMAGE
    print "<IMG SRC=\"../thumbs/$imgfile.jpg\"></A><BR>";

    // DISPLAY arcnumlink FOR REFERENCE
    print $row['arcnumlink'];
    // print " ($imgfile)";
    print "<br>($canid)";
    print "</TD>";

    // ROW BREAK AFTER EVERY FIVE ITEMS
    $N++;
    if ($N%5 == 0) {
        print "</TR><TR>";
    }
}

```

```
    }  
  }  
  fclose($fpr);  
  print "</TR></TABLE></CENTER>";  
?>  
</BODY></HTML>
```

Appendix D

Java programs, for search engine

D.1 DocLength.java: calculating “docLength” factor

```
package foa;

/** Reads the invertedIndex file created by mp2.CreateInvertedIndex and
    produces the doc length file needed by mp3.SearchEngine. This involves
    doing a pass through the index file, which is arranged by keyword, and
    building up an output array arranged by doc id. The calculation of
    doc length is done according to the description in Belew, section 3.6.

    @author Kate Byrne
    @version 1.0 August 2003.
**/

import java.util.*;
import java.io.*;

public class DocLength extends HashMap {

    static final String INDEX_FILE = "data/invertedIndex.unweighted";
    static final String DOCLen_FILE = "data/docLength.txt";
    static final int NDOCS = 56374;

    public static void main(String[] args) {
/** Builds the doc length array and writes it out to file **/

DocLength docLen = new DocLength(NDOCS);

System.out.println("Reading index file and calculating doc lengths..");
docLen.calcDocLens(INDEX_FILE, NDOCS);
System.out.println("Sorting and writing the docLengths file...");
docLen.writeDocLens(DOCLen_FILE);
System.out.println("Done");
    }

    public DocLength (int size) {
/** Creates a new DocLength HashMap, sized for the corpus.
    **/
    super(size);
    }
}
```

```

    private void calcDocLens(String INDEX_FILE, int NDOCS) {
/** Reads the inverted index file and uses it to build up the
    doc lengths.
**/
BufferedReader br;
String line, keyw, check;
Object value;
StringTokenizer st;
int i, DFVfreq, DFVfreqSum, kwNDocs, kwCorpFreq, docID, dCount;
double kWeight, kwIDF, dLen;
Integer docIDObj;
Double dLenObj;
// The "Obj" versions of docID and dLen are needed because HashMaps
// deal in objects, not primitive types.

i = 0;

try {
    br = new BufferedReader(new FileReader(INDEX_FILE));

    // Discard first line of index file, which contains comments
    line = br.readLine();

    // Read the inverted index file, one line at a time
    for (;;) {
i++;
line = br.readLine();

// Close file reader when end of file reached
if (line == null) {
    br.close();
    return;
}

// Initialise frequency sum for cross-checking on each line
DFVfreqSum = 0;

st = new StringTokenizer(line);

// First field is the keyword
keyw = st.nextToken();

// Second field is keyword weight. Multiply the calculated
// IDF weight by this if desired - but probably not?
kWeight = Double.parseDouble(st.nextToken());

// Third field is number of docs this keyword is in
kwNDocs = Integer.parseInt(st.nextToken());
// Calculate inverse document frequency (IDF) for this keyword
// kwIDF = Math.log(NDOCS/(double)kwNDocs);
kwIDF = Math.log((NDOCS-kwNDocs+0.5)/((double)kwNDocs+0.5));
// Uncomment the following line to factor in keyword's
// "personal" weight. (Prob bad idea.)
// kwIDF = kwIDF * kWeight;

// Fourth field is number of times keyword is in whole corpus;
// we only use this as a cross-check against sum of DFVfreqs.
kwCorpFreq = Integer.parseInt(st.nextToken());

// Now the set of document frequency vectors (DFVs):

```

```

while (st.hasMoreTokens()) {

    // Each DFV starts with a frequency count
    DFVfreq = Integer.parseInt(st.nextToken());

    // Second DFV field is a separator (-1); discard it
    check = st.nextToken();
    if (!check.equals("-1")) {
        System.err.println("Invalid separator; expecting -1 " +
            "\tKeyword: " + keyw +
            "\ntoken is " + check);
        System.exit(1);
    }

    // Next is list of docIDs, until next separator (-2).
    // Create a new HashMap entry for each docID, or update
    // the one already there.

    dCount = 0;

    while((docID = (Integer.parseInt(st.nextToken()))) > 0) {

        docIDObj = new Integer(docID);
        dCount++;

        // Look up current dLen sum for this document, if it's
        // already present (it'll be null if not)
        value = this.get(docIDObj);
        if (value != null) {
            dLen = Double.parseDouble(value.toString());
        } else {
            // If it was null, change it to zero
            dLen = 0.0;
        }

        // Calculate next term and add to running sum
        dLen = dLen + ((DFVfreq * kwIDF) * (DFVfreq * kwIDF));
        dLenObj = new Double(dLen);

        // Add or replace running sum for this document.
        // NOTE: square root is taken just before printing.
        this.put(docIDObj, dLenObj);
    }
    // Last token read should be -2, signalling end of this DFV
    if (docID != -2) {
        System.err.println("Invalid separator; expecting -2 " +
            "\tKeyword: " + keyw +
            "\ntoken is " + docID);
        System.exit(1);
    }
    // At end of DFV update tally of frequency count total
    DFVfreqSum = DFVfreqSum + (DFVfreq * dCount);

}

// At end of the line, do a cross-check on overall corpus
// frequency of this keyword
if (DFVfreqSum != kwCorpFreq) {
    System.err.println("Frequency sum cross-check failed " +
        "for keyword: " + keyw +
        "\n kwCorpFreq = " + kwCorpFreq +
        "\tDFVfreqSum = " + DFVfreqSum);
    System.exit(1);
}
}

```

```

    }
} catch (Exception e) {
    System.err.println("Exception caught in doDocLens(): ");
    e.printStackTrace();
    System.exit(1);
}
}

private void writeDocLens(String DOCLLEN_FILE) {
/** Writes out the HashMap of doc lengths to the output file,
    sorting it by docID first.
    */
    BufferedWriter bw;
    int docID;
    Integer docIDObj;
    double dLen;

    Object[] docIDs = this.keySet().toArray();
    Arrays.sort(docIDs);

    try {
        bw = new BufferedWriter(new FileWriter(DOCLLEN_FILE));
        for (int i=0; i < docIDs.length; i++) {
            docID = Integer.parseInt(docIDs[i].toString());
            docIDObj = new Integer(docID);
            dLen = Double.parseDouble(this.get(docIDObj).toString());

            // Need to take the square root of the dLen summation
            dLen = Math.sqrt(dLen);

            bw.write(docID + " " + dLen);
            bw.newLine();
        }
        bw.close();
    } catch (Exception e) {
        System.err.println("Exception caught in writeDocLens(): ");
        e.printStackTrace();
        System.exit(1);
    }
}
}

```

D.2 Loader.java: Adding keyword weights

```

package foa;

/** Reads the invertedIndex file created by mp2.CreateInvertedIndex and
    updates the keyword weighting factors in it (second field), writing
    out a new version of the file.

    The program reads in a file of tokens that are to be given higher weights,
    eg because they come from important fields in the corpus. The words have
    to be tokenised, cleaned (removing punctuation) and stemmed; and stop
    words are dropped. Then they can be compared with the entries in the
    invertedIndex file. Once a word acquires a higher weight it keeps it,
    whatever unstemmed form it has and whatever context it appears in. The
    new weight is a fixed value - all the input tokens get the same new
    weight (eg 2.0 instead of 1.0). The input file should be just a

```

```

    stream of tokens to process; not XML.

    @author Kate Byrne
    @version 1.0 August 2003.
**/

import foa.WordCleaner;
import foa.PorterStemmer;
import java.util.*;
import java.io.*;

public class Loader {

    static final String INDEX_IN = "data/invertedIndex.unweighted";
    static final String INDEX_OUT = "data/invertedIndex.weighted";
    static final String TOKENS_IN = "data/heavyTokens.txt";
    static final String STOP_WORDS = "data/stop.wrd";
    static final double NEW_WEIGHT = 2.0;

    public static void main(String[] args) {
/** Reads the input tokens in and processes them; then reads the
    index file and updates the weights as necessary.
**/

    ArrayList inputTokens = new ArrayList();

    System.out.println ("Reading token file and processing...");
    inputTokens = processToks(TOKENS_IN, STOP_WORDS);
    System.out.println ("Updating weights in index file...");
    updateWeights(inputTokens, INDEX_IN, INDEX_OUT, NEW_WEIGHT);
    System.out.println ("Done");
    }

    private static ArrayList processToks (String TOKENS_IN,
    String STOP_WORDS) {
/** Reads the input data from file. Drops stop words, then cleans
    the rest and returns them in an ArrayList for easy random access.
**/
    BufferedReader br;
    StringTokenizer st;
    String line, token;
    ArrayList tokens = new ArrayList();

    try {
        br = new BufferedReader(new FileReader(TOKENS_IN));

        // Initialise the stop word file
        WordCleaner.initStopWord(STOP_WORDS);

        // Read the input data file, one line at a time
        while ((line = br.readLine()) != null) {
            // Split each line into tokens
            st = new StringTokenizer(line);

            // Process each token

            while (st.hasMoreTokens()) {

                // Remove punctuation and convert to lower case
                token = WordCleaner.stripPunc(st.nextToken());

                // Skip stop words (and tokens that were punctuation only)
                if (!WordCleaner.stop_Word(token)) {

```

```

// Stem the token
token = PorterStemmer.stem(token);

// Now put it in the ArrayList, if not already there
if (!tokens.contains(token)) {
    tokens.add(token);
}
}
}

br.close();

} catch (Exception e) {
    System.err.println("Exception caught in processToks(): ");
    e.printStackTrace();
    System.exit(1);
}

return tokens;
}

private static void updateWeights (ArrayList tokens, String INDEX_IN,
String INDEX_OUT, double NEW_WEIGHT) {
/** Goes through the invertedIndex file, keyword by keyword,
    updating the word's weight if it matches a word from the
    input list. Writes the updated entries to the output file.
    */
BufferedReader br;
String line, keyw, rest;
double kWeight;
String[] splitLine;
BufferedWriter bw;

int i = 0;

try {

    br = new BufferedReader(new FileReader(INDEX_IN));
    bw = new BufferedWriter(new FileWriter(INDEX_OUT));

    // First line of index file is a comment; write it straight out
    line = br.readLine();
    bw.write(line);
    bw.newLine();

    // Read the inverted index file, one line at a time
    while ((line = br.readLine()) != null) {

// Only need to use the first two fields; but keep rest of
// line for outputting. Split on whitespace.
splitLine = line.split("\\s", 3);

// First field is the keyword
keyw = splitLine[0];

// Second field is the present keyword weight.
kWeight = Double.parseDouble(splitLine[1]);

rest = splitLine[2];

```

```

// Look up list of tokens to see if weight needs updating
if (tokens.contains(keyw)) {
    kWeight = NEW_WEIGHT;
    i++;
}

// Write the line to the output file
bw.write(keyw + "\t" + kWeight + " " + rest);
bw.newLine();

    }

    br.close();
    bw.close();
    System.out.println(i + " keywords updated");

} catch (Exception e) {
    System.err.println("Exception caught in updateWeights(): ");
    e.printStackTrace();
    System.exit(1);
}
}
}

```

D.3 NEMarker.java: Finding NEs in input text

```

package foa.ner;

/** Marks Named Entities within a piece of text. Reads in gazetteer file of
    entity terms to match. Takes text to be marked on the command line.
    Designed to be called from command line (for CANTRIP queries) or from
    foa.ner.ProcessText.java, to deal with the whole corpus.

    @author Kate Byrne
    @version 1.0 August 2003.
    **/

import java.util.*;
import java.io.*;

public class NEMarker {

    static final String TERMS = "foa_dir/data/compoundTerms.byLength";
    static final String QUERY = "foa_dir/data/nob/cantrip_queryNER";
    String textIn;

    public static void main(String[] args) {
        /** Reads the input text and the gazetteer list and looks for
            gazetteer terms occurring in the text. Those found are replaced
            by versions of themselves having all spaces replaced by
            underscores. The gazetteer should be sorted by reverse order
            of line length, to provide greediness: ie the longest possible
            string is always found.
            **/

        String line, textOut;
        String textIn = "";

        try {

```

```

        BufferedReader br = new BufferedReader(new FileReader(QUERY));
        while ((line = br.readLine()) != null) {
textIn = textIn + line;;
        }
        br.close();
    } catch (Exception e) {
        System.err.println("Exception caught while reading query file: ");
        e.printStackTrace();
        System.exit(1);
    }

    LinkedHashMap termList = new LinkedHashMap();
    NEMarker marker = new NEMarker();

    termList = marker.readGaz(TERMS);
    textOut = marker.markNE(textIn, termList);

    System.out.println(textOut);
}

    public LinkedHashMap readGaz (String TERMS) {
/** Reads the gazetteer file and returns it as a map.
**/
    BufferedReader br;
    String line, term, term_;
    int hashPosn;
    LinkedHashMap termList = new LinkedHashMap();

    try {
        br = new BufferedReader(new FileReader(TERMS));

        // Read the gazetteer file, one line at a time. Split line into
        // two fields (term without and with underscores) on "#".
        while ((line = br.readLine()) != null) {
            hashPosn = line.indexOf("#");
            // Wrap term in a reg expr to cope with surrounding text
            term = "[ ('\\"]" + line.substring(0, hashPosn) + "\\W";
            term_ = " " + line.substring(hashPosn+1) + " ";
            termList.put(term, term_);
        }
        br.close();

    } catch (Exception e) {
        System.err.println("Exception caught in readGaz(): ");
        e.printStackTrace();
        System.exit(1);
    }

    return termList;
}

    public String markNE (String textIn, LinkedHashMap termList) {
/** Goes through the input text replacing substrings matched from the
    gazetteer with a version having underscores replacing spaces (so
    the entity becomes a single token). Returns the revised text.
    Converts to lower case, as termlist is in lower case.
**/

    String term, newTerm;
    textIn = textIn.toLowerCase();
    Iterator it = termList.keySet().iterator();

```

```

while (it.hasNext()) {
    term = (String) it.next();
    newTerm = (String) termList.get(term);
    textIn = textIn.replaceAll(term, newTerm);
}
return textIn;
}
}

```

D.4 DbPrepper.java: Marking-up NEs in database text

```

package foa.ner;

/** Marks Named Entities within database text, read from file. Reads in
    gazetteer file of entity terms to match. Writes out a new version of
    the database text file.

    @author Kate Byrne
    @version 1.0 August 2003.
**/

//import NEMarker;
import java.util.*;
import java.io.*;

public class DbPrepper {

    static final String DATA_IN = "data/NER.in";
    static final String DATA_OUT = "data/rcdbNER.xml";
    static final String TERMS = "data/compoundTerms.byLength";

    public static void main(String[] args) {
        /** Reads in the input XML data file and processes the text; then
            writes out the data to a new file.
        **/

        DbPrepper dbPrepper = new DbPrepper();

        System.out.println ("Reading and processing data...");
        dbPrepper.processData(DATA_IN, TERMS, DATA_OUT);

        System.out.println("Done");

    }

    private void processData (String DATA_IN, String TERMS,
        String DATA_OUT) {
        /** Reads the input data from file. Calls NEMarker to find NEs in
            each section of text. Input file in FOA XML format. Writes
            output file as each input section is processed.
        **/

        // For reading input file. (For simple XML tree like this it's
        // simpler not to use the horrid Java XML parsing paraphernalia.)
        BufferedReader br;
        String line;
        boolean inAbstract = false;

```

```

// For processing the NE terms.
LinkedHashMap termList = new LinkedHashMap();
NEMarker neMarker = new NEMarker();
String rawText = "";
String markedText = "";
int endText;

// For writing out the processed data.
BufferedWriter bw;

// Use the NEMarker to read the term list file into map.
termList = neMarker.readGaz(TERMS);

try {
    br = new BufferedReader(new FileReader(DATA_IN));
    bw = new BufferedWriter(new FileWriter(DATA_OUT));

    // Read the input data file, one line at a time
    while ((line = br.readLine()) != null) {

if (line.startsWith(" <ABSTRACT> ") || inAbstract) {
    // This line is part of the text to process.
    // Above conditions can't both be true.

    // Don't want line breaks, but make sure there's at
    // least a space between lines. (Extra spaces are ok.)
    line = line + " ";

    if (line.endsWith(" </ABSTRACT> ")) {
// Final line of ABSTRACT text. Add this line to
// raw text and reset flag. Note that raw text field
// is cleared after use, so doesn't matter if this is
// both start and end of ABSTRACT (ie one line total).
rawText = rawText + line;
inAbstract = false;

// Having assembled the text, cut off XML tags, as
// don't want them converted to lower case with rest.
// Then call the NE marker.
endText = rawText.length() - 12;
rawText = rawText.substring(11, endText);
markedText = neMarker.markNE(rawText, termList);
// Clear the raw text field for next use
rawText = "";

// Write out the marked text as a single long line
bw.write(" <ABSTRACT> " + markedText + " </ABSTRACT>");
bw.newLine();
    }

    else if (inAbstract) {
// Continuation line of ABSTRACT text. Add this line
// to the raw text.
rawText = rawText + line;
    }

    else {
// Starting a new ABSTRACT. Set the raw text to start
// with this line, and set the flag.
rawText = line;
inAbstract = true;

```

```
    }  
  }  
  else {  
    // Write non-Abstract lines straight to output file  
    bw.write(line);  
    bw.newLine();  
  }  
  }  
  
  br.close();  
  bw.close();  
  
} catch (Exception e) {  
  System.err.println("Exception caught in processData(): ");  
  e.printStackTrace();  
  System.exit(1);  
}  
  
return;  
  }  
}
```


Appendix E

Sample from generated gazetteer file

The following is a selection of expressions taken from the gazetteer list, to illustrate the sort of terms it contains. The full list has over 20,000 terms.

```
...
home farm
home improver
home lorimer
homes (&|and) interiors
homestead moats?
homesteads?
homesteads?: palisaded
homesteads?: scooped
honeyman
honeyman,? jack (&|and) robertson
honeyman keppie (&|and) mackintosh
honeymen
honeywill (&|and) stein\.?
honington hall
(honourable )?j(ohn)? elphinstone
hood (&|and) campbell[- ]?jones
hood (&|and) macneill
hood,? anthill (&|and) anthill
hood,? daniels (&|and) miller
hood kinnear
hoogovens aluminium
hooke
hooker
hoole
hooper
hoover (&|and) hoover
```

hope (&|and) wickham[-]?jones
hope,? henry (&|and) sons
hope's metal windows
hope[-]?taylor
hopetoun drawings
hopetoun house
hopetoun lodge
hope works
hopkins
hoppett
hoppus
hoprig house
horatio ross
horder
horizontal mills?
hornby
horne
horne (&|and) hardie
horne (&|and) macleod
horne (&|and) peach
hornel art gallery
horne,? macleod (&|and) oswald
horn house
hornsey
horsburgh
horse[-]?engine[-]?houses?
horse[-]?engine platforms?
horse[-]?engines?
horse[-]?gangs?
horse[-]?gin houses?
horse[-]?gins?
horsehair factory
horsemill house
horse[-]?mills?
horsey
horsey (&|and) muthesius
horsfield a e
horsley
hoscote
hosiery factory
hosiery mill
hosiery works
hospice
hospital blocks?

hospital boards?
hospitalfield house
hospital for sick children
hospitals?
hospitalshields farm
hossack
hoste
hostel
hostelry
h o tarbolton
hotel buildings?
hotels?
hothersall (&|and) tye
houfe
houghton hall
hougue bie
houlder
houndwood house
hourihane
house (&|and) bungalow plans
house (&|and) garden
house (&|and) gardens
house construction
house for an art lover
house furnishers
households?
house of aldie
house of commons
house of craigie
house of daviot
...

Appendix F

Notes for CANTRIP evaluators

1. There are 16 queries, on architecture, archaeology and industrial archaeology. Each query has to be run 6 times, as there are 6 different versions of CANTRIP to compare. There is one double-sided evaluation sheet per query.
2. Fill in the screen query form exactly as shown on the evaluation sheet. It's important that all the evaluators pose the queries in the same way.
3. Select the appropriate "APP VERSION" for each of the 6 runs in turn, using the buttons labeled 1 to 6 at the bottom of the query screen. Then hit Return or click the "Submit Query" button.

NOTE: The version number is displayed in the URL box and the window title bar of the results screen, so you can always check which version you're running.
4. The results screen will show up to 20 hits. Fill in the sheet for the version you're running, entering the number of hits and the response time in the spaces provided. Both these numbers are displayed on the results screen.
5. There are 20 boxes on the sheet for each run, laid out in rows of 7 to match the results screen. In the box corresponding to each image returned, put either a **tick** to indicate the image is relevant to the query, a **cross** if it isn't, or a **question-mark** if you can't decide or it's unclear for some reason.
6. If you have any comments on the run, please put them in the comment box. Comments are not required on every run, but if there's anything particular that strikes you, it would be very helpful to note it so that I can follow up when analysing the results.
7. Similarly, any overall comments at the end would be welcome.
8. To see a larger image and its accompanying text, click on the thumbnail.
9. The easiest way to go back to the query screen for the next run of the **same** query is to select **Go ... CANTRIP evaluation version** from the browser menu. This avoids having

to retype the query — you only need to change the radio button to the next setting. (Unfortunately the “back” button doesn’t work properly with forms.)

10. After all six runs have been done, the easiest way to move to the next query is via the **New Query** button on the results screen.

There are a few images that appear to be incorrectly linked to database records: the text bears no relation to the image. If you encounter any of these, please put a note in the comment box and mark the image on the text not the picture.

Sometimes the same image will appear more than once in the results. This happens when an image is linked to different pieces of text, that are found relevant to the query. Just try and mark each one on its merits.

Thank you very much for helping with the evaluation!

Appendix G

Sample evaluation form

Query 5 - Highland distilleries: processes associated with malting

Image description: malting process

Site description:

Type of site: distillery

Location: HIGHLAND

Other key words:

Version 1

NUMBER OF HITS: _____

--	--	--	--	--	--	--	--

RESPONSE SECS: _____

COMMENTS:

Version 2

NUMBER OF HITS: _____

--	--	--	--	--	--	--	--

RESPONSE SECS: _____

COMMENTS:

Version 3

NUMBER OF HITS: _____

--	--	--	--	--	--	--	--

RESPONSE SECS: _____

COMMENTS:

Version 4

NUMBER OF HITS: _____

RESPONSE SECS: _____

COMMENTS:

Version 5

NUMBER OF HITS: _____

RESPONSE SECS: _____

COMMENTS:

Version 6

NUMBER OF HITS: _____

RESPONSE SECS: _____

COMMENTS:

Appendix H

Evaluation Results

	precision	K-score	recall	secs
Query 1 - people: Frank Matcham				
20 19 18 17 16 15 14 13 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1	40.00%	120	100.00%	3.00
20 19 18 17 16 15 14 13	100.00%	132	100.00%	4.00
20 19 18 17 16 15 14	100.00%	119	87.50%	2.33
20 19 18 17	100.00%	74	50.00%	8.33
-1 -1 -1 -1 -1 -1 -1 -1 -1 11 10 9 8 7 6 5 4 -1 -1	42.11%	49	100.00%	2.00
20 19 18 17 16 15 14 13	100.00%	132	100.00%	4.00
Query 2 - social housing in Fife by Haxton and Watson				
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1	0.00%	-20		3.67
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1	0.00%	-20		8.33
	0.00%	0		2.67
-1	0.00%	-1		11.00
	0.00%	0		1.00
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1	0.00%	-20		5.67

		precision	K-score	recall	secs
Query 3 - masonic halls in Edinburgh					
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 10 -1 -1 -1 -1 -1 -1 -1 -1		5.00%	-9	25.00%	2.67
20 19 18 17		100.00%	74	100.00%	4.33
		0.00%	0	0.00%	3.67
20		100.00%	20	25.00%	9.00
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 10 -1 8 -1 6 5 0 -1 -1 -1		20.00%	14	100.00%	1.00
-1 -1 -1 -1 -1 15 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1		5.00%	-4	25.00%	5.00
Query 4 - Mavisbank House					
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 4 3 2 -1		15.00%	-8	100.00%	3.33
20 19 18		100.00%	57	100.00%	4.33
20 19 18		100.00%	57	100.00%	2.00
20		100.00%	20	33.33%	8.33
20 19 18 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1		15.00%	40	100.00%	1.00
20 19 18		100.00%	57	100.00%	4.33
Query 5 - processes associated with malting, in Highland region distilleries					
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1		0.00%	-20		3.33
20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1		100.00%	210		8.33
		0.00%	0		1.00
20		100.00%	20		9.33
20 19 18		100.00%	57		1.00
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1		0.00%	-20		5.00

		precision	K-score	recall	secs
Query 6 - cranes or bridges associated with the Arrol firm					
-1 -1 0 0 16 0 0 13 12 11 0 0 8 7 -1 -1 4 0 0 1	40.00%	68		2.67	
-1 19 18 17 16 15 14 0 0 11 0 0 0 -1 6 5 4 -1 0 0	50.00%	122		6.33	
	0.00%	0		2.33	
-1 19 -1	33.33%	17		7.33	
20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1	100.00%	210		2.33	
0 0 0 0 0 15 0 0 0 0 0 -1 -1 -1 6 -1 -1 -1 -1 -1	20.00%	13		5.00	

Query 7 - colliery bath houses

-1 -1 -1 -1 -1 -1 -1 13 12 -1 10 9 8 -1 -1 -1 -1 -1 -1 -1	25.00%	37		3.00
-1 -1 -1 -1 16 -1 -1 -1 12 11 10 -1 -1 7 6 5 4 3 2 1	55.00%	68		6.00
	0.00%	0		1.00
-1 -1	0.00%	-2		8.00
-1 -1 18 -1	25.00%	15		4.00
-1 -1 18 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 4 -1 -1 -1	10.00%	4		4.33

Query 8 - linoleum works in Fife

20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1	100.00%	210		2.67
20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1	100.00%	210		4.67
20 19 18 17 16 15 14 13 12	100.00%	144		1.00
20 19 18	100.00%	57		7.00
20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1	100.00%	210		1.00
20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1	100.00%	210		5.00

	precision	K-score	recall	secs
Query 9 - churches dedicated to a particular saint (St Columba)				
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1	0.00%	-20		3.33
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1	0.00%	-20		5.67
20 19	100.00%	39		2.33
-1 -1 -1 -1 -1	0.00%	-5		9.00
	0.00%	0		1.00
20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1	100.00%	210		5.33
Query 10 - animal bones in Orkney excavations				
20 -1 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1	95.00%	190		3.00
20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1	100.00%	210		6.33
	0.00%	0		1.00
20	100.00%	20		8.00
20 19 18 17 16 15 14 13 12 -1	90.00%	143		1.67
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1	0.00%	-20		5.00
Query 11 - oblique aerial views of Edinburgh				
-1 -1 -1 -1 -1 15 14 -1 -1 -1 -1 -1 -1 7 -1 -1 4 -1 -1 1	25.00%	26		4.00
20 19 18 17 16	100.00%	90		7.00
20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2	100.00%	209		1.00
20 19 18 17	100.00%	74		7.33
-1 -1 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1	90.00%	169		5.33
-1 -1 -1 -1 -1 15 14 -1 -1 -1 -1 -1 -1 7 6 -1 -1 3 -1 -1	25.00%	30		5.67

		precision	K-score	recall	secs
Query 12 - oblique aerial views of cropmark 'forts'					
20	19 -1 17 -1 -1 -1 -1 -1 -1 -1 -1 8 -1 -1 5 -1 -1 -1 -1	25.00%	54		3.67
20	19 18 -1 -1 15 14 13 -1 -1 10 -1 -1 7 -1 -1 -1 -1 2 1	50.00%	109		5.67
		0.00%	0		1.00
20	-1 -1 -1 -1 -1 -1 -1 -1	11.11%	12		7.33
-1	-1 -1 -1 -1 -1 15 -1 13 -1 11 -1 -1 -1 -1 6 -1 -1 -1 2 -1	25.00%	32		3.33
20	-1 18 -1 -1 15 14 13 12 -1 -1 -1 -1 7 -1 -1 -1 -1 -1 -1	35.00%	86		5.67
Query 13 - oblique aerial views by John Dewar					
-1	-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1	0.00%	-20		3.67
20	-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1	5.00%	1		5.33
		0.00%	0		1.00
20	-1 -1 -1 -1	20.00%	16		7.00
-1	-1 -1 -1 -1 -1 -1 -1 13 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1	5.00%	-6		3.67
-1	-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1	0.00%	-20		5.33
Query 14 - chambered cairns in the Western Isles					
-1	-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1	0.00%	-20		3.00
20	19 18 17 16 15 14 13 12 11 10 9 8 7	100.00%	189		5.67
20	19 18 17	100.00%	74		1.00
20		100.00%	20		7.33
20	19 18 17 16	100.00%	90		1.00
-1	-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1	0.00%	-20		5.33

																			precision	K-score	recall	secs
Query 15 - hill-forts in Angus																						
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0.00%	-20		2.67
20	19	18	17	16	15	14	13	12	11	10								100.00%	165		5.00	
																			0.00%	0		1.67
20	19																	100.00%	39		6.33	
20	19	18	17	16	15	14	13	12	-1	-1	-1	-1	-1	-1	-1	-1	3	2	57.89%	141		1.00
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0.00%	-20		4.33
Query 16 - cropmarks of Roman camps																						
20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	100.00%	210	3.00
-1	-1	-1	-1	-1	-1	-1	13	12	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	10.00%	7	5.33
																				0.00%	0	1.00
-1	-1	-1																		0.00%	-3	7.00
-1																				0.00%	-1	1.00
-1	-1	-1	-1	-1	-1	-1	13	12	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	10.00%	7	5.00

Bibliography

- Belew, R. K. (2000). *Finding Out About: a Cognitive Perspective on Search Engine Technology*. Cambridge University Press.
- Bikel, D. M., Miller, S., Schwartz, R., and Weischedel, R. (1997). Nymble: a high-performance learning name-finder. In *Proceedings of ANLP-97*, pages 194–201. <http://citeseer.nj.nec.com/bikel97nymble.html>.
- Borthwick, A., Sterling, J., Agichtein, E., and Grishman, R. (1998). Exploiting diverse knowledge sources via maximum entropy in named entity recognition. In *Proceedings of the Sixth Workshop on Very Large Corpora*, New Brunswick, New Jersey. Association for Computational Linguistics. <http://citeseer.nj.nec.com/borthwick98exploiting.html>.
- Byrne, K. (2003). QSX project: Using SilkRoute with text rich data. (MSc coursework project).
- Curran, J. R. and Clark, S. (2003). Language independent NER using a maximum entropy tagger. In Daelemans, W. and Osborne, M., editors, *Proceedings of CoNLL-2003*, Edmonton, Canada.
- Del Bimbo, A. (1999). *Visual Information Retrieval*. Morgan Kaufmann Publishers, Inc.
- Fernandez, M., Kadiyska, Y., Suciu, D., Morishima, A., and Tan, W.-C. (2002). SilkRoute: A framework for publishing relational data in XML. *ACM Transactions on Database Systems*.

- Flank, S. (1998). A layered approach to NLP-based information retrieval. In *Proceedings of the 36th ACL and the 17th COLING Conferences*, pages 397–403, Montreal.
- Greenwood, M. A. and Gaizauskas, R. (2003). Using a named entity tagger to generalise surface matching text patterns for question answering. In *EACL03: 10th Conference of the European Chapter of the Association for Computational Linguistics*, Budapest, Hungary. (Workshop on Natural Language Processing for Question-Answering).
- Grover, C., Matheson, C., and Mikheev, A. (1999). *TTT: Text Tokenisation Tool (Users' Manual)*. Language Technology Group, Human Communication Research Centre, University of Edinburgh. <http://www.ltg.ed.ac.uk/>.
- Guglielmo, E. J. and Rowe, N. C. (1996). Natural-language retrieval of images based on descriptive captions. *ACM Transactions on Information Systems*, 14(3):237–267.
- Jurafsky, D. and Martin, J. H. (2000). *Speech and Language Processing*. Prentice-Hall.
- Katz, B. and Lin, J. (2003). Selectively using relations to improve precision in question answering. In *EACL03: 10th Conference of the European Chapter of the Association for Computational Linguistics*, Budapest, Hungary. (Workshop on Natural Language Processing for Question-Answering).
- Kruschwitz, U. (2000). UKSearch — Web search with knowledge-rich indices. In *Proceedings of the AAAI-2000 Workshop on Artificial Intelligence for Web Search*, Technical Report WS-00-01, pages 41–45, Austin, TX. American Association for Artificial Intelligence, AAAI Press. <http://citeseer.nj.nec.com/kruschwitz00uksearch.html>.
- Lewis, D. D. and Sparck Jones, K. (1996). Natural language processing for information retrieval. *Communications of the ACM*, 39(1):92–101.
- Manning, C. D. and Schütze, H. (2002). *Foundations of Statistical Natural Language Processing*. MIT Press.
- McCallum, A. K. (1996). Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. <http://www.cs.cmu.edu/~mccallum/bow>.

- Mikheev, A., Moens, M., and Grover, C. (1999). Named entity recognition with gazetteers. In *Proceedings of EACL*, Bergen, Norway. <http://citeseer.nj.nec.com/mikheev99named.html>.
- Pastra, K., Saggion, H., and Wilks, Y. (2003). NLP for indexing and retrieval of captioned photographs. In *EACL03: 10th Conference of the European Chapter of the Association for Computational Linguistics*, Budapest, Hungary.
- Ramakrishnan, R. and Gehrke, J. (2000). *Database Management Systems*. McGraw-Hill, 2nd edition.
- Ravichandran, D. and Hovy, E. (2002). Learning surface text patterns for a question answering system. In *Proceedings of the 40th ACL Conference*, pages 41–47, Pennsylvania.
- Rose, T., Elworthy, D., Kotcheff, A., Clare, A., and Tsonis, P. (2000). ANVIL: a system for the retrieval of captioned images using NLP techniques. In *Challenge of Image Retrieval*, Brighton, UK.
- Smeaton, A. F. and Quigley, I. (1996). Experiments on using semantic distances between words in image caption retrieval. In *Proceedings of the 19th International Conference on Research and Development in Information Retrieval*, Zürich.
- Soubbotin, M. M. and Soubbotin, S. M. (2003). Use of patterns for detection of answer strings: a systematic approach. In *Proceedings of the TREC-11 Conference*. NIST.
- Veltkamp, R. C. and Tanase, M. (2000). Content-based image retrieval systems: a survey. Technical Report UU-CS-2000-34, Utrecht University, Department of Computing Science.
- Wang, J. Z., Li, J., and Wiederhold, G. (2001). SIMPLIcity: Semantics-sensitive Integrated Matching for Picture Libraries. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(9).
- Xie, L. (2001). Experiments in Content-Based Image Retrieval. <http://www.ee.columbia.edu/~xlx/courses/vis-hw3/>.