Service selection using trust and semantics: Getting the service you want with strong words and who your friends know

Nicholas Brunwin

Bachelor of Science in Computer Science with Honours
The University of Bath
May 2007
This dissertation may be made available for consultation within the University Library and may be photocopied or lent to other libraries for the purposes of consultation.

Signed:
Service selection using trust and semantics: Getting the service you want with strong words and who your friends know

Submitted by: Nicholas Brunwin

COPYRIGHT

Attention is drawn to the fact that copyright of this dissertation rests with its author. The Intellectual Property Rights of the products produced as part of the project belong to the University of Bath (see http://www.bath.ac.uk/ordinances/#intelprop). This copy of the dissertation has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the dissertation and no information derived from it may be published without the prior written consent of the author.

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Batchelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Signed:
Abstract

Web services are well-defined interfaces for program communication across the internet. Joining these interfaces together is a solved problem but finding the right interfaces is not. (Chapman et al. 2006) Here two services, one utilizing the semantic meaning of web service descriptions and the other utilizing social networks to match users’ preferences, are put forward that with the support of other services aim to make the matchmaking process more accurate. We show that the two services can work within the matchmaking framework. That they can contribute to the choosing of an appropriate web service. Then we show how the algorithms can be tuned and outline areas where they can be improved upon.
Contents

I Introduction 1

1 Problem description 2
  1.1 KNOOGLE 4
  1.1.1 Existing matchers 4
  1.2 Common requirements 5
  1.3 Text semantics 6
  1.4 Trust 6
  1.5 The attached CD 7

II Text Semantics 8

2 Aims and expectations 9
  2.1 Matching on meaning not spelling 9
  2.2 What to look for 11
  2.3 Finding and comparing text semantics 11
  2.4 Other considerations 11

3 Literature Review 13
  3.1 Existing text semantics matchers 13
  3.2 Finding and comparing text semantics 13
    3.2.1 Taxonomies 13
    3.2.2 Ontologies 14
## 3.2.3 Thesauruses

### 4 Design and implementation

4.1 Matchers

4.1.1 Query-to-description semantic scorer

4.1.2 Description-to-query semantic scorer

4.2 Thesauruses

4.2.1 WordNet

4.2.2 Connecting Mathematics thesaurus

4.2.3 Wikisauri

### 5 Testing and discussion

5.1 Thesauruses

5.2 Distance to travel

5.3 Algorithms

5.4 Parts of the NAG library descriptions

5.5 Distribution of results

5.5.1 Specificity of queries

5.5.2 A variety of queries

5.5.3 Unlikely queries

5.6 Anomalous results

5.7 Considerations

5.7.1 Weighting

5.7.2 Fixing over-keenness

5.7.3 Removing polysemy

5.7.4 Term occurrence

5.7.5 Tuning and human computation

5.7.6 Context not meaning
### CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.3.1</td>
<td>Common features</td>
<td>83</td>
</tr>
<tr>
<td>8.3.2</td>
<td>TidalTrust</td>
<td>83</td>
</tr>
<tr>
<td>8.3.3</td>
<td>Single Depth TidalTrust</td>
<td>84</td>
</tr>
<tr>
<td>8.3.4</td>
<td>Multiple Depth TidalTrust</td>
<td>87</td>
</tr>
<tr>
<td>9</td>
<td>Testing and discussion</td>
<td>89</td>
</tr>
<tr>
<td>9.1</td>
<td>Trust network generation</td>
<td>89</td>
</tr>
<tr>
<td>9.2</td>
<td>Caching</td>
<td>90</td>
</tr>
<tr>
<td>9.3</td>
<td>The run-times of the algorithms</td>
<td>94</td>
</tr>
<tr>
<td>9.4</td>
<td>Distribution of scores</td>
<td>99</td>
</tr>
<tr>
<td>9.5</td>
<td>Considerations</td>
<td>101</td>
</tr>
<tr>
<td>9.5.1</td>
<td>Feedback</td>
<td>101</td>
</tr>
<tr>
<td>9.5.2</td>
<td>Are the possible trust values evenly spread?</td>
<td>102</td>
</tr>
<tr>
<td>9.5.3</td>
<td>Filling the gaps</td>
<td>102</td>
</tr>
<tr>
<td>9.5.4</td>
<td>Speed optimisations</td>
<td>103</td>
</tr>
<tr>
<td>9.5.5</td>
<td>Using Automated Collaborative Filtering algorithms</td>
<td>104</td>
</tr>
<tr>
<td>IV</td>
<td>Consolidation</td>
<td>105</td>
</tr>
<tr>
<td>10</td>
<td>Common features</td>
<td>106</td>
</tr>
<tr>
<td>10.1</td>
<td>Web service arguments</td>
<td>106</td>
</tr>
<tr>
<td>11</td>
<td>Critique</td>
<td>108</td>
</tr>
<tr>
<td>11.1</td>
<td>The Text Semantics Matcher</td>
<td>108</td>
</tr>
<tr>
<td>11.2</td>
<td>The Trust Matcher</td>
<td>109</td>
</tr>
<tr>
<td>11.3</td>
<td>Overall</td>
<td>110</td>
</tr>
<tr>
<td>12</td>
<td>Notes on KNOOGLE</td>
<td>111</td>
</tr>
<tr>
<td>12.1</td>
<td>New dimensions</td>
<td>111</td>
</tr>
<tr>
<td>12.2</td>
<td>Repository access</td>
<td>112</td>
</tr>
</tbody>
</table>
12.3 Operation names .............................................. 112

V Appendix ......................................................... 119

A Code ..................................................................... 120

A.1 Semantic Text Matchmaker .................................. 121
  A.1.1 File: Matcher.java ........................................... 121
  A.1.2 File: QueryToDescriptionScorer.java .............. 125
  A.1.3 File: TermToDescriptionScorer.java ............... 127
  A.1.4 File: SearchPath.java .................................... 129
  A.1.5 File: ASearchSynset.java ............................... 130
  A.1.6 File: Weighting.java .................................... 132
  A.1.7 File: DescriptionToQueryScorer.java ............. 133

A.2 Trust Network Matchmaker .................................. 139
  A.2.1 File: SingleDepthTidalTrustScorer.java .......... 139
  A.2.2 File: MultipleDepthTidalTrustScorer.java ....... 143
# List of Figures

1.1 The KNOOGLE system ......................................................... 3

5.1 Thesauruses: The scores in rank order using the query-to-description scorer . . . 34
5.2 Depth: The scores in rank order using the query-to-description scorer ........ 37
5.3 Depth: The average times for using the query-to-description scorer .......... 39
5.4 Algorithms: The scores for each description for the two algorithms .......... 40
5.5 Algorithms: The scores in rank order using both algorithms ................. 41
5.6 The distributions of scores for different amounts of the description .......... 45
5.7 The distributions of scores for queries of different detail ...................... 46
5.8 The distributions of scores for random queries ................................ 47
5.9 The distributions of scores for unlikely queries ................................ 48

8.1 User’s information within a WSTRN FOAF file ............................ 77
8.2 User’s relations in WSTRN FOAF files .................................... 79
8.3 User’s reviews in WSTRN FOAF files ..................................... 81

9.1 The Pajek generated graph from which the trust network was generated .... 91
9.2 The hit-rates of possible user-web service pairs ............................ 95
9.3 The distribution of scores of the three methods ................................ 98
9.4 The distribution of scores the Multiple Depth weighted against expectations method 99
9.5 The distribution of scores the Multiple Depth using weighted averaging method 100
# List of Tables

5.1 Time and score statistics of the algorithms .................................................. 43
5.2 How a score was reached ................................................................................. 51

9.1 Statistics of the sets reviews used in testing .................................................. 92
9.2 Multiple Depth algorithm matcher example (microsecond) times ................... 92
9.3 Page caching Multiple Depth algorithm matcher example (microsecond) times .. 93
9.4 Model caching Multiple Depth algorithm matcher example (microsecond) times . 93
9.5 Results caching Multiple Depth algorithm matcher example (microsecond) times 93
9.6 The worst case running times (in usecs) of the algorithms against number of reviews 96
9.7 The average running times (in usecs) of the algorithms against number of reviews 96
9.8 The “right-average” running times (in usecs) of the algorithms against number of reviews ................................................................. 97
Acknowledgements

I would like to thank Dr Julian Padget and Dr Bill Naylor for answering my many hundred of questions on how things worked and how I could get other things to work. I would like to thank my parents for their correction of my writings and helping me limit the great length that my sentences can sometimes find themselves reaching these days. I would also like to thank Tristan Caulfield and Martin Brain for their advice. Beyond that I would like to hope that people will not wonder why they were not mentioned, because the reason is it is late and I am no good with names.
Part I

Introduction
Chapter 1

Problem description

Web Services are well-defined interfaces for program communication across the internet. Combined with Service Oriented Architectures they allow the construction of distributed data-flow styled programs. As more Web Services become available knowing the best one to put into a work-flow becomes a bigger problem. Joining together the services into work-flows is a solved problem but getting the services you actually need is difficult. (Chapman et al. 2006)

To make the situation slightly better registries of services have been introduced. UDDI (Universal Description Discovery Integration) is an open-standard for registries that has become popular amongst web service users and publishers. UDDI registries allow developers and software to find web services quickly by associating metadata with the web services such as their positions within taxonomies. Services implementing the UDDI standard are described as containing ‘White Pages’ listing business contact details and identifiers; ‘Yellow Pages’ listing categorisations of the business and services; and ‘Green Pages’ listing technical details of web services exposed by businesses.¹ UDDI registry services are of limited use on their own for automatic service discovery because their understanding is only of the relationships between the web services and entities that they contain and not of the meaning of the metadata.

¹See http://www.uddi.org/
Figure 1.1: The KNOOGLE system
1.1 KNOOGLE

KNOOGLE is a matchmaking and brokerage framework that is used to choose a web service for a task based on a user’s input. Users provide the service with a set of queries for selected matchmaking services along with an expression (a query over RDF triples) of how to combine the scores from the matchmakers to produce an overall score for each web service for ranking. The division of the query between matchmaking web services firstly provides the obvious advantage of increased speed through the division of labour across machines. It also allows the matchmakers to be highly specialised in both the types of web services they can consider and the information and techniques used to score the web services.

Figure 1.1 illustrates the use and method of the KNOOGLE broker. The numbered steps can be described as follows:

1. A user or application sends to the KNOOGLE broker:
   - A set of repositories it would like a web service chosen from.
   - A set of matchmakers (matchers) to use to generate scores for each web service.
   - A set of queries, each corresponding to a matcher that should be used to score the web service with.
   - A selection policy that describes how the sets of scores awarded by the matchers (described as RDF triples) corresponding to each web service are to be combined to produce a score for each web service.

2. The KNOOGLE broker gets hold of a list of the web services from each of the selected Grimores UDDI repositories.

3. Each matcher is sent its corresponding query and each web service.

4. The matchers calculate and return a score (between 0 and 1 - 1 being a perfect match) to the given web service.

5. The broker uses the selection policy it was given to calculate the final score for each web service. The highest scoring web service is then returned to the user.

1.1.1 Existing matchers

At present there exists for KNOOGLE five matchmakers all specialising in the domain of mathematical web services. (Chapman et al. 2006)

1. A matcher that matches based on the structure of web services’ capabilities against a given task description.
2. A matcher that matches based on elements and attributes of a web services’ capabilities relationships within a taxonomic structure to elements and attributes in a task description.

3. A matcher that users Racer\(^2\) (a reasoner and inference server for OWL (the Web Ontology Language)) to reason the overlap between task descriptions and the capabilities of web services.

4. A matcher that works out the algebraic equivalence of the functions being computed by a web service.

5. A matcher that substitutes random values into expressions, with common results being evidence of equivalence between web service operations.

It is notable that each of these matchers require an understanding of mathematics and of how to provide a useable query for each matcher. Their specialisation is also a limiting factor in their use.

### 1.2 Common requirements

Paolucci et al. (2002) provides four requirements that web service matching engines should satisfy summarised as follows:

1. The broker should be flexible in the semantic matching it allows between the web service advertisements and the users’ queries based on the ontologies available.

2. False negatives should be minimised.

3. Honesty should be encouraged in the web service advertisements and the queries put forward by the user.

4. Matchmaking should be done quickly and efficiently.

The first of these requirements is not of interest to our matchers as they will not work from ontological information from the web service advertisements. The other three requirements do though make for relevant advice. A more relevant version of the first requirement, to our project, is that the matchers should be flexible and not be designed in such a way that limits the possible domains of their deployment where possible. (Kuokka & Harada 1996)

The matchers created will have to be compatible with the KNOOGIE system and as such generate scores between 0 and 1 to imply how close the query and the web service match. The generated scores should be distributed across the range to allow user’s selection policies to be able to make use of the information to distinguish whether matches are good enough to be considered.

\(^2\)See [http://www.racer-systems.com](http://www.racer-systems.com)
1.3 Text semantics

Searching and matching using meaning, semantics, is increasing. Meaning of previously abstract information providing extra information for computation with to find what is wanted. It is obvious why meaning is increasingly used as better techniques are discovered and computing power increases. Knowing the meaning of something and the meaning’s relation to other concepts allows their similarity to be calculated.

A number of the matchers described in Section 1.1.1 used applications of semantics to interpret expressions or to reason over descriptions of capabilities. But the use of semantics in these cases relied on the use of a small controlled vocabulary and syntax with which to describe concepts to allow them to be interpreted and reasoned with.

What we intend to construct is a matcher that uses the meaning of terms in natural language sentences. The matcher would match user desires such as “finds square roots of a given number” against descriptions given in the documentation of web services. This requires the understanding of an incredibly large vocabulary of terms and their relations and the ability to quickly score the similarity in the meaning of one expression over another. The learning required of the user is minimised as they only need say what they are looking for in their language. Part II describes the aims, implementation and results of implementing such a system.

1.4 Trust

Users preferences for particular services may depend on their experience with them. But at present no matcher exists to take this into account. A matcher that could use ratings of web services given by users to generate the scores it returns would allow personal preferences of users to be taken into account.

A user cannot (or at least ideally should not) review a web service before they have used a web service and users will struggle to review every web service. Using another user’s rating of a web service where the user does not have a rating for it does not work if any arbitrary user is picked as a substitute reviewer because the difference in opinions they have is unknown. But if users were to not only rate web services but also state how much they trust other users where they were able to then users’ ratings for web services they have not rated could be inferred from the opinions of users they trust. The users that a user is able to assign trust values to will have reviews that cover a limited proportion of the web services that the user could be interested in. But the range of web services covered can be extended by inferring the opinions of these users as well, for web services that neither the original user nor that users ‘friends’ (the users the user trusts) have reviews for. Therefore using a network of trust values, users, reviews and web services users’ opinions of web services can be inferred to provide a value for how much they might prefer each web service. The required input from the user would have be their opinions on web services and
the relationships of trust they have with other users which could be constructed using a social networking web site for web service users. Part III covers the aims, implementation and results of implementing such a system.

1.5 The attached CD

The CD attached to this dissertation contains the source code of the programs developed in the process of undertaking this project. A selection of the source code to demonstrate the algorithms developed is included in the appendix of this document but it is recommended that the source code is viewed digitally where the structure of the program code will appear more apparent. Along with the source code is documentation on how the software developed should be deployed. The CD also contains the input and output data of the tests run.
Part II

Text Semantics
Chapter 2

Aims and expectations

In Section 1.3 an outline of a matcher was given. Here and in Chapter 3 we explored in detail what can be used to create the matcher. We outline the design and requirements of a matcher that uses the individual terms of two strings of text to compare their meaning.

2.1 Matching on meaning not spelling

Our intention as introduced in Section 1.3 is to not measure an overlap in strings of text between the user’s query and the description of the web service. But instead to measure the overlap in meaning between the user’s query and the web service’s description.

The exact meaning that a web service’s description or a user’s query is intended to convey, because of the variety of language used, is not information that programs at present can assuredly generate. If it was possible to infer the exact meaning of blocks of text then it would be likely translation between languages by software would be error free and human-machine interfaces would be easier to use. Context is an important part of understanding but is not so much of a problem in controlled cases such as here where the context can be specified as “mathematical web services” for instance, to restrict the possible interpretations of a set of sentences.

Instead of taking on the complexity involved in the understanding of the whole, be it the query or the description, it is best to start by attempting to understand the parts, the terms. Concepts, referred to by terms in sentences each have meanings which are comparable to others using predefined structures such as ontologies. Similar structures do not exist for whole sentences. Working with terms allows the problem to be simplified and the solution sped up.

There then exists the problem of how to handle terms that operate on the meaning of surrounding terms. For instance negatives, such as if a description defines a web service to be not something. False positives would occur if that something was searched for and the “not” was ignored. Gen-
erally though descriptions are written in a positive sense. Businesses publishing web services have an interest in advertising in a registry what they can achieve not what they cannot. User queries are controlled by the user and they also should search for what the user wants to find, although some users may wish to prevent certain results. Because the matcher is to work as part of a group of matchers there is not a need for the matcher to always choose the perfect web service for the user. Ideally it should, but errors, such as ignoring an occasional negative word in a description will not be significant when the matcher is used with the other matchers. Detecting negating terms such as “not”, “neither” and “nor” is not so much a problem if a list of such terms can be decided on beforehand. Knowing on which of the surrounding terms these terms operate on is not so simple and requires multiple passes of the text and for the program to be able to accurately group the terms into concepts to be operated on. The extra complication and computation required to analyse words effects on others has led to our avoiding of it, initially at least.

Working purely with the meaning of terms should require that if just the immediate meaning of terms is used that “connection” will be matched with “connexion” and “realize” will be matched with its English spelling of “realise”. Those examples are of words with a common origin and a matcher with a knowledge of how words can be transformed such as the suffix “ise” being interchangeable with “ize”. Such a technique is referred to as stemming where the root form of a word can be generated by manipulating affixes of a word.

The matcher should be yet more sophisticated and recognise terms with different roots that have common meanings such as “meal” and “repast”. In a more appropriate context the matcher should recognise that “summation”, “addition” and “plus” can all be used in the same context to mean the same thing.

Returning to meals the matcher should recognise that a web service that provides a “meal” or “repast” might be appropriate for a user that wants “breakfast”. But the matcher should recognise that by providing a “meal” the web service might not be providing an appropriate “breakfast”, whereas if the user asked for a “repast” the match should be stronger because the meanings are the same. In reverse if the web service promised “breakfast” but the user was after a “meal” the web service might be appropriate. But not as appropriate as a web service that promised generic a “meal” because the user was not after anything more specific.

The matcher should also realise that “dish” whilst is neither a hypernym (super class) or a hyponym (sub class) of “meal” it can constitute a part of it so “dish” in one and “meal” in the other should contribute something to the overlap in meaning between a query and a description.
2.2 What to look for

Terms here are spoken of as the concepts used in the text of both the queries and the descriptions. “Square” is a term, but so is “square root” and “root mean square”, so terms can overlap and are not restricted to single words. Single words do of course represent concepts but when joined with other words can represent entirely different concepts and generally the longest terms are what the author intended to mean, rather than the individual meanings of the sub-terms. For instance in the sentence “The standard deviation of the class was high.” the term “standard deviation” is intended as a concept that is possessed by the class and although “standard” and “deviation” are individually recognisable concepts they carry less of the intended message separated as they do as a pair.

Detecting terms in both the query and the web service descriptions is not trivial because the English language for one does not have a way of indicating that multiple words make up a whole term, not that is consistently used at least. The query though is the responsibility of the user to compose and as such they can be instructed to group terms with quotation marks (as they seem a natural fit) to reduce the problem to being only with the finding of terms in the web service descriptions.

2.3 Finding and comparing text semantics

Once able to turn a string of text into a set of terms the next step is to be able to find their meaning, and compare the overlap in meaning of each of the terms to the meaning of another set of terms. To do this a mapping must exist between terms and their meanings. These meanings would have to be organised in a graph structure that enables the relationships between terms to be represented. The types and number of relationships between terms could then be used to determine the score given. The types of structure available for mapping from terms to meanings and the relationships between meanings that they support are discussed in Chapter 3.

2.4 Other considerations

The matcher being a web service used by many users using many different queries to score against many web service descriptions means that the web service should avoid maintaining local copies of information as it is unlikely to remain relevant. As the matcher is to be used by the KNOOGLE broker to generate scores for potentially thousands of web services from a selection of repositories it has to be quick to run as the user (and software) using the brokering services will expect a prompt response to the request for a web service.

The matcher is to be initially tested on the descriptions of mathematical web services, because
of their availability, but should be at least extendable to use in other areas such as bioinformatics and be able to use sufficient knowledge of the terms used in such areas to be able to assign and compare meanings of the terms.
Chapter 3

Literature Review

3.1 Existing text semantics matchers

Many semantic matchers already exist that operate over ontology based information provided by
users and web service providers. The use of free text based matchmaking much more limited
despite the extra accessibility it provides to the matchmaking process.

The COINS (COmmon INterest Seeker) was developed for matching using volumes of unstruc-
tured text. It uses the System for the Mechanical Analysis and Retrieval of Text (SMART) to
filter and retrieve information from the free text, using document vectors of to compare text us-
ing an inverse document frequency algorithm. COINS for querying uses the Knowledge Query
and Manipulation Language (KQML) and not free text itself as in our service. It stands as an
example that free text queries can be done in the matchmaking environment.

3.2 Finding and comparing text semantics

To produce the meaning of words or sets of words it is necessary to have a model of conceptual
relationships for defining the word meanings so what follows is a discussion of the types of
structures that can be used in computational linguistics to describe the meaning of terms and the
relationships between these meanings.

3.2.1 Taxonomies

Taxonomies provide a basic form of relationship between concepts classifying them with hy-
pernym and hyponym relationships to one another.(Merriam-Webster 2003) A common example
 CHAPTER 3. LITERATURE REVIEW

of taxonomies is taxonomies of biological classifications of lifeforms. Taxonomies are almost always domain specific, such as the GAMS (Guide to Available Mathematical Software) taxonomy\textsuperscript{1} from the National Institute of Science and technology and the American Mathematical Society’s 2000 Mathematics Subject Classification\textsuperscript{2} both of which classify areas of mathematics.(Boisvert et al. 1991) Taxonomies though often use titles of concepts and not individual strings of words and each class they contain only has the title as words that would identify the concept in a body of text. Taxonomies have no concept of words or terms with the same meaning, making them hard to use to extract meaning from text with. The GAMS taxonomy has been converted into an ontology by the MONET project available in the OWL format.(Consortium 2002, Dewar 2004) OWL is described in Section 7.1.1. Taxonomies lack the mapping from terms to meanings that is required to be used directly with the text, but the concepts and hierarchical relationships they contain are often complete for the area that they cover.

3.2.2 Ontologies

Ontologies were first used by philosophers as structures that describe the relationships entities have to one another and they allow many different types of relationships making them more descriptive than taxonomies (as these just contain super-class, sub-class relationships). As with taxonomies ontologies do not map directly to terms in a language, only to concepts and singular terms at that, making using them for understanding free text difficult. Many ontologies are available for technical subjects such as the Microarray Gene Expression Data ontology for use in genomics.(Stoeckert & Parkinson 2003) Ontologies contain a large amount of semantic information and are relatively simple for machines to reason with. Description logic (subsumption) reasoners are tools that can be used to understand the semantics of ontologies and inferring from them information. Sattler (2006) maintains a list of available description logic reasoners. The relationships held between concepts within ontologies vary but reasoners are able to establish if two concepts are interchangeable for a purpose if an ontology is sufficiently well built. Ontologies like most taxonomies are almost always specific to an area because of the time it takes and complexities of constructing an ontology that encompasses a wide area of knowledge accurately. The quality of the information that can be inferred with ontologies is good but difficulties lie in the translation between their structures and terms and with the coverage of terms that they have.(Arano 2005, Lutz 2006)

3.2.3 Thesauruses

The word thesaurus comes from the Greek word \textit{thesauros} meaning collection but now has come to mean a collection of words and their relationships to other words.(Merriam-Webster 2003)

\textsuperscript{1}See from http://gams.nist.gov/Classes.plain
\textsuperscript{2}See http://www.ams.org/msc/
Where ontologies describe the semantics and relationships of things thesauruses (also pluralised as “thesauri”) describe the relationships between words and as such contain less semantic information. (Arano 2005)

Tsujii & Ananiadou (2005) argues with many biological examples that although ontologies can be seen as important text mining resources, thesauruses can be more useful because of the lack of synonym information within ontologies. But it mentions that thesauruses must be able to deal with term ambiguity where the same word has a different meaning depending on its context.

The storing of context needs to be done along with a thesaurus to disambiguate text. Kilgarriff (2003) talks about word senses (a word’s word sense being its meaning) and that they can be used as the elements of thesauruses but cause the additional work of having to determine the sense of the word before it can be looked up within the thesaurus. Kilgarriff (2003) uses the theoretical argument based on the statement of Wittgenstein (1953) in his book on philosophy and semantics - “don’t ask for the meaning, ask for the use” to emphasise why word senses adds complexity to thesauruses. Ide & Véronis (1998) provides an overview of possible solutions to word sense disambiguation.

For the matcher which of the senses of a term is intended is not a decision that needs to be made. Because the matcher can work with each term individually, by pursuing each interpretation of the term and using the closest matching interpretation to relate to the terms in the other piece of text the closest match should be valid to use. Because queries of single ambiguous terms are unlikely, a false positive match of a term being used out of context will only contribute to a small part of the total contribution from all the terms in the query. There is the possibility though that the highest scoring web service will be one that matches in a context not intended by the user, particularly if the query is ambiguous but no web service matching the description as the user intended it exists in the repositories given. But in this case the use of multiple matchers reduce the severity of the error and if the overlap in interpreted meaning is small the difference between that and the score for no overlap at all will make the error even less significant.

Thesauruses, in mapping from terms to sets of terms with the same meaning, are a solution to the need for a mapping between terms and meanings. The set of meanings covered by a thesaurus depends on its specialization and depth. For the matcher, depth of coverage of terms is important but to be able to use the matcher within a variety of different domains the thesaurus or thesauruses used must cover the terms used in the web service’s descriptions.

Meanings in thesauruses are defined by groups of terms, sometimes referred to as “word senses” or synsets - sets of synonyms, words with the same meaning. Synonymy is the basic relationship that all modern thesauruses have represented for terms. Many digital thesauruses support other relationship types such as antonyms (linking to synsets with an opposing meaning), related concepts, broader concepts (hyponyms) and narrower concept (hyponyms). The limit on the types of relationships are because of the range of concepts that thesauruses cover and because they inherently deal with terms (words). The types of relationships between concepts in these advanced
thesauruses are normally sectioned by the syntactic category or part of speech, meaning whether
the term corresponds to an adverb, a verb, an adjective or a noun. (Miller 1995)

WordNet

WordNet\(^3\) is an online lexical database written by humans (as opposed to machine generated),
bearing a resemblance to a thesaurus combined with a dictionary but containing many more re-
lationships between word senses than is typically found in thesauruses and providing example
sentences for many of the terms in each context they have. WordNet has a large vocabulary (con-
taining over two hundred thousand word-sense pairs) but does not contain many domain specific
words such as medical terms meaning it is a close match to what would be ideal for discover-
ering the semantics of text, but lacks the most specialised on subject terms. (Miller 2007) So it
would make a good general purpose thesaurus for understanding most of the terms encountered
but would need supplementing to cover all the terms, with the most specialised terms perhaps
being the most significant. An example of the use of WordNet can be found in Chai & Bierman

Thesaurus generation

Observations from the study of WordNet and the other available thesauruses found so far reveal
the need for more specialised thesauruses. These are either yet to be acquired or in need of
generation. One such way to generate a thesaurus to meet the requirements arises using bodies
of text from a domain as inputs to the thesauruses creation.

In Lin (1998) words with similar meanings are captured by looking for text that specifies informa-
tion about objects. The similarity of objects is defined to be the fraction of the information
about the objects that is common between them. The experiments described in this paper show
the thesaurus generation method can generate synonym sets comparable to those of what Word-
Net contained at the time.

Hearst (1992) shows how looking for common lexico-syntactic patterns can be used to identify
synonymy and hyponymy relationships between words. It was found that a larger than expected
corpora of text was needed to build the desired relationships between words but that the rela-
tionships were accurately defined although the paper only talks about the relationships between
nouns as extraction other parts of speech would imaginably require an algorithm of greater so-
phestification detecting a larger variety of lexico-syntactic patterns.

There is a preference for the thesauruses being already generated because of the complexities
involved in generating thesauruses, the volume of input they require to achieve accurate relation-
ships and the lack of time within which to test the products of such a process.

\(^3\) See http://wordnet.princeton.edu/
Chapter 4

Design and implementation

Here we describe the two matcher algorithms implemented then the two thesauruses to be used with them. We also describe an investigation carried out into the development of a thesaurus using data from Wikipedia and similar sites.

Java was chosen as the language for implementing the matcher as it was a familiar language, a popular and well supported choice for implementing web services and has many libraries available to provide the needed functionality of the matcher. The languages support of interfaces allows parts of the matcher such as thesauruses to be swapped easily at run-time. The development process was iterative.

4.1 Matchers

Two matchers were developed to score the overlap meaning between the terms of a user’s query and the terms in web service descriptions and are described below. The first, the query-to-description scorer, taking each term in the query and trying to find that term, or related terms in the web service description text. The second, the description-to-query scorer, taking each term it can identify in the web service description and matching them against the terms from the user’s query.

4.1.1 Query-to-description semantic scorer

This algorithm can be found in Section A.1 and the attached CD and is implemented using the uk.ac.bath.stamm.QueryToDescriptionScorer class, with the algorithm starting in the class’s score method.
Getting the words and terms to match with

The first step of the algorithm is to tokenize the web service description text into individual words. In this step stop words are removed from the text because by their nature they are not interesting to the matcher. Stop words are words such as “and”, “a” and “the”. These words encode meaning that is not interesting to the matcher and that both the query and the web service’s description both contain the word “the” should not be relevant to the score as it does not indicate any common meaning between the two. The large stop word list from MySQL 4.0.20\footnote{Available from \url{http://meta.wikimedia.org/wiki/MySQL_4.0.20_stop_word_list}} was used, distributable as it is under the GNU General Public License. The stop word list numbered 545 common English words and some less common such as “whereupon”. The following twenty-five words were removed from the stop word list because they either encoded numbers, ordinals, amounts, negations, subjects or designations and were felt in the context of web service descriptions could have valuable meaning to contribute.

```
Listing 4.1: Words removed from the stop words list
eight, even, fifth, first, five, forth, four, nine, none, not, never, once, one, own, second, self, seven, several, six, third, three, twice, two, value, zero
```

The words that survive the check against the stop word list are turned into their base forms by the object that implements the \texttt{IThesaurus} interface that the \texttt{QueryToDescriptionScorer} object is passed when it is instantiated. Converting the words into their base forms can be a process of stemming as described in Section 2.1, the base forms of a word being the word as it would be found in a dictionary or thesaurus, the root of the word without any tense, plurality or such implied, such as the basic form (in English the imperative mood) of a verb would be its base form. For words that can be more than one part of speech they may have more than one base form. The base forms are then inserted into a hash set. Originally they were instead inserted as keys into a hash table where the object the key maps to is a count of the number of the occurrences of the word in the description, but as is later explained in Section 5.7.4 this information was not to be used.

The query itself is then tokenized but not into individual words, but into terms. In Section 2.2 it was noted that identifying the terms of the query, to use to capture what the user is looking for where their identification is necessary, can be done by the user. The matcher assumes individual words are intended as terms unless they are surrounded by double quotes in which case the contents of the double quotes is taken as a single term. Terms that are in the list of stop words described above are ignored, although it is possible and allowed for stop words to exist within terms such as in the term “\textit{line of best fit}”.
Finding term in the description

Each term of the query is passed to the TermToDescriptionScorer instance’s score method class which aims to find and evaluate how close a meaning a term exists in the description to the term from the query it is given if any exists at all.

The given term is looked up in the thesaurus and the set of synsets it falls into are returned, that being the set of sets of words that could be said to have the same meaning as the term, not knowing its context. For instance for the word “tree” there are the following synsets in the WordNet thesaurus, with the proceeding definitions and uses.

**Part of speech:** Noun

- **tree** (a tall perennial woody plant having a main trunk and branches forming a distinct elevated crown; includes both gymnosperms and angiosperms)
- **tree**, tree diagram (a figure that branches from a single root) “genealogical tree”
- **Tree**, Sir Herbert Beerbohm Tree (English actor and theatrical producer noted for his lavish productions of Shakespeare (1853-1917))

**Part of speech:** Verb

- corner, **tree** (force a person or an animal into a position from which he cannot escape)
- **tree** (plant with trees) “this lot should be treed so that the house will be shaded in summer”
- **tree** (chase an animal up a tree) “the hunters treed the bear with dogs and killed it”; “her dog likes to tree squirrels”
- **tree**, shoetree (stretch (a shoe) on a shoetree)

Each of the terms in each of the synsets is then searched for within the hash set of the words in the description, unless the term has more than one word in it in which case it is searched for within the text of the description. If any of these terms are found within the description it is taken that the meaning of the term in the query is present within the meanings of the terms of the description and the value 1.0 is returned.

Branching from meanings

If the synsets’ terms are not found within the descriptions terms each of the paths from the synsets that are termed allowable are listed and added to an ordered list of paths. A path being a step along a relationship away from the given synset in the thesauruses structure, for instance a synset’s hypernym path is the relationship from the synset to its hypernyms. A path being allowable if it
is a branch to a term with a related meaning. For instance from the original source term’s synset
all following paths are allowable - hypernyms, hyponyms, holonyms, meronyms, derivatives
and “see alsos”. Not all thesauruses will support all forms of branching, but at a minimum the
hypernym, hyponym - broader than, narrower than relationships should be supported. Paths from
a synset are added whenever a search for its terms in the web service description is unsuccessful.
The list of paths is ordered such that the paths are in descending order of the size of their weight-
ing. The weighting of a path being the value that would be returned if a term in one of the synsets
at the end of the path was found in the web service description. The weighting of paths should
obviously drop the further from the source term because the meaning is by definition less related
but the types of relationship that lead to the end of a path affect the rate at which the weight-
ing drops. Finding the most effective way to assign weights is something that is discussed in
Section 5.7.1.
After the initial set of synsets is searched for and if the search is unsuccessful the top most
weighted path is popped from the list and the synsets at the end of that path are searched for. The
paths from these new synsets are added and the process repeated from the popping of the top
path until either the list of paths is empty or a match in the description’s text is found. The top
most weighted path is always chosen because it should point to the synset with the most closely
thought of meaning to the original term and ensures that the term that is found, if one is, is the
one from the highest valued synset. To avoid branching throughout the thesaurus the adding of
new paths is stopped beyond a certain distance or once the top most path has been assigned a
weighting of 0.
There is a problem anticipated in that the following of a chain of hypernyms leads to one of a set
of synsets and thus an unscrupulous business could publish a web service description containing
elements of these root synsets and possibly guarantee themselves getting some score wherever
the query contains terms near enough the root that are not found elsewhere in the description. The
ploys of errant businesses is an issue that the project has concerned itself with and the marginal
benefit to a web services score from having a match with a root synset term is minimised as
the weighting for such a term drops off as its distance grows from the source term. Also any
genuine contender for a good match with the query will have to have terms close to the meaning
of significant proportion of the terms of the query to stand out, particularly after the application
of the selection policy.
The motivation behind the choice of paths that can be taken is explained as follows:

**Hyponyms** Also known as subordinates. X is a hyponym of Y if X is a (kind of) Y. Hyponyms
are thought of as the strongest relationship that is considered because if something does
not mean the same thing as a term, the next best thing it can be is a kind of that term. If
you were searching for a dog a poodle would be a valid match, but would not be such a
good match as a generic dog because a dog is what you asked for. Hyponym paths from
the source term’s synset are allowed to continue up to the maximum path distance because
they only get more specialised with depth and under the presumption that the meaning of a term is composed of the common aspects of the terms hyponyms (to be referred to as the subtyping argument). Asking for a dog does not imply that the dog should be able to pull a sledge across snow and ice, a reasonable query for that might be for a husky.

**Hypernyms** Also known as superordinates. Y is a hypernym of X if X is a hyponym of Y. Hypernyms are considered the second strongest relationship following from the ideas above that if the user is searching for a specific type of something, a more general type of that thing may cover what they are looking for. To return to the canine example a shop that advertises dogs might sell the husky that you are after but makes no definite promise to. It could be argued that the hypernym relationship leads to closer meanings than the hyponym relation because a more specific meaning may not cover everything that is meant by the original term, whereas a broader meaning term is more likely to include what is desired but this can be countered by the subtyping argument. Hypernym paths from the source term’s synset are allowed to continue up to the maximum path distance because the meaning of the original term is always in some sense encompassed.

**Holonyms** X is a holonym of Y if Y is a part of X. The holonym relationship is a weak relationship because it does not imply in any sense that a holonym of a term is substitutable for the original term. In fact it should be taken that the chance of substitution being possible is unlikely but it is included to allow a kind of related term to be included in the search. Related terms have value in that they imply a common subject. A steering wheel is a part of a car and thus anything that mentions cars is at least mildly related so the hint is believed to be worth taking. The weakness of this relationship implies a low weighting and only one step from the source term’s synset is taken along a holonym path, because the term being a part of something that is a part of something would intuitively take its meaning too far from the original term and its intentions.

**Meronyms** X is a meronym of Y if Y is a holonym of X. The meronym relationship is tied to the holonym relationship and is included for the same reasons as an accessor to related terms. It also follows that only one meronym step can be taken and from that set of meronyms no further paths can be taken before too unrelated concepts are reached.

**Derivatives** A weak relationship included for its ability to link to related terms and follows the same usage as the other weak relations. Perhaps in being related through the language’s use the relationship is a stronger relationship than meronymy and holonymy as in the observation that “electric” and “electricity” are clearly more related than “car” and “steering wheel”.

**See also** A relationship that is included in some digital thesauruses that is a weak relationship that works as a related term finder. This follows the same usage as the other weak relationships above.
Some terms found in the query may not exist in the thesaurus and in that case the original term is returned labelled so that no paths are taken from it as they would intuitively not exist in the thesaurus.

The values returned from searching for each (non-stop word) term in the query are averaged and that value is returned as the result for that web service description.

**Pseudo code for the query to description matcher algorithm**

```plaintext
score(query, description, maximumDistance)
wordsInDescription = empty set
for words in description:
    if word is not a stop word then:
        for term in base forms of word:
            add(wordsInDescription, term)

count = 0
total = 0.0
for terms in query:
    if term is not a stop word then:
        count = count + 1
        total = total + search(term, wordsInDescription, description, maximumDistance)
if count > 0 then:
    return total / count
return 0.0

search(term, wordsInDescription, description, maximumDistance)
senses = senses of term
paths = empty sorted list
for synsets in senses:
    for terms in synset:
        if term is more than one word then:
            if term is found in description then:
                return 1.0
        else if wordsInDescription contains term:
            return 1.0
        if maximumDistance > 0:
            addValidPaths(paths, synset, 1)
while paths is not empty:
    path = pop(paths)
    if weighting(path) = 0 then:
        return 0.0
for synsets from following path:
```

**CHAPTER 4. DESIGN AND IMPLEMENTATION**

22
for terms in synset:
    if term is more than one word then:
        if term is found in description then:
            return weighting(path)
    else if wordsInDescription contains term:
        return weighting(path)
    if maximumDistance > depth(path):
        addValidPaths(paths, synset, depth(path) + 1)
return 0.0

4.1.2 Description-to-query semantic scorer

The uk.ac.bath.stamm.DescriptionToQueryScorer class implements this algorithm and can be found in Section A.1, with the algorithm starting in the class’s score method.

As the query-to-description matcher worked by finding terms with the same or similar meaning to the terms in the query in the description, the description-to-query matcher works by finding terms from the description and trying to find terms with the same or related meaning in the terms of the query. A flaw with the query-to-description matcher was that it did not know what the terms in the description were so although it could conveniently store the single word terms of the description in a hash set for fast checks and look up, multiple word terms required the description’s text to be searched each time they were encountered. This leads to going through the description text many times. The description-to-query matcher by working from terms in the description walks the description once.

Extracting terms from the query

As has already been stated in Section 2.2, the query is text controlled by the user and as such the terms in it are known because they are either individual words or words grouped by quotation marks. Three maps are used to store terms from the query. The first of the maps goes from the term from the query to a set containing its base forms obtained from the thesaurus. The second map goes from the term from the query to a count of the occurrences of the term in the query. The third map goes from the base forms of the queries terms to a sorted list of the scores they receive in descending order. These maps are constructed from each term in the query that is not a stop word. A count of the number of terms in the query is also maintained, equal to the total of the values mapped to in the terms to occurrences map.

Extracting terms from the description

The description text is then parsed character by character so that the words in it fill a predetermined number of separate buffers each containing a single word. Once the buffers are filled
or the end of the description is reached the buffers and the words in the buffers including the dividers between them are passed to the search method. This tries to find a compound term composed of the words in the buffers, with the dividers back between them, within the thesaurus. If such a compound term does exist in the thesaurus it is scored against the terms in the query using the same weighting and branching system as in the query-to-description matcher described in Section 4.1.1 but with the weightings reversed. By reversed it is meant that hypernym steps from the description’s terms towards query terms are from the perspective of the query’s terms hyponym steps for example and as such should be weighted as if they are hyponyms because the scoring is done ultimately in terms of the query. The score the matched base term of a query term is given is added to the base term’s sorted list of scores. The terms that are found within the searched for compound term are then also scored, in case a shorter term was in fact what was intended. The single word terms within the compound term are tested to check that they are not stop words before they are scored against the queries terms.

If the compound term does not exist in the thesaurus the last word of the term is removed and placed at the start of a new list of buffers to make up part of the next compound term. The remaining parts of the compound term are then turned into a new compound term and attempted to be found in the thesaurus and the process is repeated as above with non-existent compound terms having their last word added to the start of the new list of buffers. Once the term to be checked is an individual word it is only searched for if it is not a stop word and whether or not it exists in the thesaurus it is not added to the list of buffers. To avoid searching for the same term twice in the query a hashset of tried terms is maintained, so that only previously not searched for terms are searched for.

The parsing of the description text then continues adding words to the buffers left over from the call to the search method, repeating the process until the end of the description text is reached. Then the top score from the base terms of each query word (multiplied by their occurrence) are summed together and divided by the count to produce a final score.

Comparing the matchers

The description-to-query matcher was designed to avoid the repeated searching of the description for each multiple word term related to the query. It also avoids the cost of detecting if a term linked to a query term is more than one word but at the price that instead of traversing the thesaurus for each term in the query the thesaurus has to be searched for each potential term in the description and traversed for each of the actual terms that could be found in the query. Descriptions are often much longer than queries and without the specification of what terms are in descriptions, as they are specified in the query, the number of searches in the description-to-query matcher algorithm is larger and hence means much more traversing of the thesaurus structure. As an offset to this cost the description-to-query matcher ensures the same search is never repeated, but the query-to-description algorithm could be enhanced to store the scores of
previous searches so that it too does not have to repeat searches unnecessarily. This would imply that the original query-to-description matcher is in fact the faster of the two and this is explored in the testing Section 5.3.

There is another cost in the use of the description-to-query matcher and that exists where one term in the query blocks another one from receiving a score. In the query-to-description algorithm each of the terms in the query would be searched for and as such each would be assigned a score that would be averaged to produce the final score. In the description-to-query algorithm the same cannot be said to be true. Take a query containing the terms “car” and “transport” and a web service description containing the term “car” and all its other terms being irrelevant. With the query-to-description algorithm the term “car” would get a score of 1.0 and the term “transport” would get some score from cars being a kind of transport. With the description-to-query algorithm the term “car” would get a score of 1.0 just as before, but the term “transport” would receive no score. This flaw means that the way in which queries are written for the description-to-query matcher will differ from the way they are written for the query-to-description matcher, avoiding terms that might be blocked. Users would not know this difference intuitively and explaining it to them is another cost to consider.

4.2 Thesauruses

The need was illustrated in Section 3.2.3 for a thesaurus that covered the terms used in the web service description. It was noted that WordNet provided good coverage of the English language as a whole, but not of terms used in particular fields, such as for instance the names of proteins. So whilst searching for sources of digital thesauruses that could cover the specialist terms used to describe web services an implementation of the semantic matcher that used the WordNet thesaurus data was implemented. It being unlikely that any thesaurus found that covered an area in great detail would also cover the same breadth of terms that is covered by WordNet it becomes likely that joining the domain specific thesauruses to the WordNet thesaurus will be a necessary solution for best coverage.

The requirements so far as usage and included relationships are concerned were mentioned in Section 4.1 but for clarity they will be specified here. Ideally thesauruses should be able to recognise terms that are not in one of their base forms and therefore return a list of the possible base forms of a term. Thesauruses should intuitively be able to return lists of terms with the same meanings as a given term, the term’s synsets. Thesauruses should be able to identify if a term that it has returned consists of more than one word. Thesauruses should at a minimum include hypernym and hyponym relationships between synsets and ideally should contain derivative, see also, holonym and meronym relations, or some subset. Most trivially a thesaurus should be able to identify that a term exists within it.
CHAPTER 4. DESIGN AND IMPLEMENTATION

4.2.1 WordNet

The database for the thesaurus behind the online WordNet service is freely available under a BSD-like license\(^2\) and is used in a wide range of language processing applications. The database is made available as text files of sense mappings and ANSI prolog code. For making sense of and accessing the data there are libraries available that interface with a range of languages.

There are three Java APIs for WordNet promoted by the WordNet website\(^3\). JWord\(^4\) by Kunal Johar, a program and library that provided a generic interface to lexical information of which WordNet was an available driver for it, was found to be unavailable at the time of development so despite its extendable architecture which may have served the requirements well it could not be used. WNJN\(^5\) by Bernard Bou, is a Java wrapper around a C++ WordNet library that uses the Java Native Interface. It had the requirement of C++ compilers and admitted that it was not built in an object oriented fashion requiring additional coding to make it as easy to use as the third of the three libraries. JWNL\(^6\) (Java WordNet Library) is an open source library for querying the WordNet database.

The JWNL allows the storage of the database at run-time to be done in a number of different ways, configured via properties files. Originally, when building and testing the program, the file-based method of database storage was preferred as it was described as being slower but having far lower memory requirements from not storing the database in memory. But after some load testing the amount of memory used by the library rose overtime to be just as large as when the database was stored in a map backed system. So the map backed data storage memory was chosen because of its faster access times and included with the library was a utility to generate the files the library required from the original WordNet data files prior to the running of the program. Storing the data in a database, either in memory using a system such as Axion or a client-server database such as MySQL is a lesser known feature of the library. But in-memory databases are reportedly no better than map-backed systems in terms of speed and memory usage according to the JWNL documentation and client-server databases are noted as slower and make the system harder to deploy.

A class to implement the requirements for a thesaurus was constructed as an interface into the JWNL. Because WordNet includes the hypernyms, hyponyms, holonyms, meronyms, derivatives and “see alsos” relationships the thesaurus created from it could implement all the relationships talked about in the requirements and Section 4.1.1.

The flaw, noted with the implemented WordNet thesaurus whilst doing unit testing using it with the description-to-query matcher, was that the JWNL came with its own set of functions for

\(^2\)See http://wordnet.princeton.edu/license
\(^3\)See http://wordnet.princeton.edu/links#Java
\(^4\)See http://www.seas.gwu.edu/~simhaweb/software/jword/index.html
\(^5\)See http://wnjn.sourceforge.net/
\(^6\)http://sourceforge.net/projects/jwordnet
finding the base form of terms and naturally used them to find terms in the database. The function that it used to do this though turned out to be over-keen to identify terms in the thesaurus from sets of words given to it. This meant it would say a given term was a valid term when in reality only part of it was and it had ignored the parts that were not part of the valid term. This over-keen identification of terms causes problems with the algorithm where terms given to it should have been flagged as not existing and their last word removed and added to the next term but instead the whole term was accepted and the last term was lost. This problem remains an issue with the code but the words that are ignored are more often than not stop words and would have been ignored anyway and the testing of the description-to-query matcher with the JWNLI dictionary in Section 5.1 shows how much this effects the algorithm’s results.

4.2.2 Connecting Mathematics thesaurus

The only domain specific thesaurus found after investigation was the Connecting Mathematics thesaurus\(^7\) provided by the Cambridge University Press under a license that allowed modification, distribution and publication of the software, the data and its results so long as a specified copyright notice and permission notice is included with the product. The source code provided with the thesaurus data is for running the Connecting Mathematics website and the website uses the data to publish and link mathematical terms and concepts in a selection of European languages. Whilst new languages are being added to the site and the provided data there is no evidence that, like WordNet, the number of concepts covered is planned to increase.

The thesaurus includes 4035 concepts (previously referred to as synsets) and with a surjective mapping from 3892 distinct English terms to those concepts (there is 15407 terms in all, referred to in the Connecting Mathematics code as “names”). These concepts are linked with 13975 relationships and there are five types of relationships. The relationships are labelled as broader, narrower, see also, references and referenced by. The first three relationships are equivalent to the hypernym, hyponym and see also relationships of the requirements for a thesaurus respectively. The final two relationships are not used as they appear to represent too large a jump in meaning in traversing along them as many concepts would reference other concepts that had been mentioned in the concepts definition but without enough of a reason to consistently trust these connections.

The database was provided as a 20 megabyte SQL file. Only a small proportion of the database was required though, the terms and what synset they were in and the relationships between the synsets. So just that information was converted into two text files, one of relationships and one of terms and their synsets. Not running a database behind the thesaurus and instead using in-memory maps built from these files allowed the thesaurus to be easier to deploy, faster and with the thesaurus only small the memory cost of using such a map is easily allowable.

Two thesauruses were constructed from the Connecting Mathematics data. One using just the

\(^7\)See http://thesaurus.maths.org
Connecting Mathematics data and the other combining that thesaurus with the WordNet thesaurus with the mathematics specific data being used whenever a term is not found in the WordNet thesaurus. Many of the simpler mathematical terms in the Connecting Mathematics thesaurus are already covered by WordNet, but tests show that the mathematical thesaurus has enough unique terms to effect the score return for a noticeable number of queries using it when compared with the WordNet thesaurus alone. The terms covered by the mathematical thesaurus cover some deeper mathematical terms than the WordNet thesaurus and therefore the combined thesaurus is an enhancement to the WordNet thesaurus. The mathematical terms covered even by the combined thesaurus are still not exhaustive. An example mathematical term it misses is “quadrature” and this lack of coverage may be down to the age level that the thesaurus is aimed at which appears to be for high-school students.

4.2.3 Wikisauri

One area that was studied with a view to creating a thesaurus with a deep knowledge of subjects was the structure and formalisms of the online encyclopedia Wikipedia. Wikipedia is an encyclopedia on the world wide web where articles can be added, written and edited by anyone (with the exception of users from banned IP addresses). This means that the content of Wikipedia covers the areas that individuals feel should be covered and uses terms based on their use, as opposed to their formal definitions. There are different levels of complexity with which thesauri (often referred to as wikisauri) can be generated from Wikipedia data with the data talked about in the following three sections that could be used, with increasing complexity.

Links and redirects

Wikipedia’s structure is that each topic that it is felt should have a page has a page that describes it in as much detail as has been contributed the topic Each topic can be said to have a unique meaning, representing a concept and forms the basis of a synset. The content of pages, describing the topic, link many of the terms used in the description to the pages that describe them. These links are stored in a database. Although all the links from a page are not necessarily to a page on a related topic For instance the page on frequency modulation links to the pages adjective and attributively. Related concepts can be found by using only the links that are mutual. Related topics can be identified as narrower topics if they contain significantly fewer mutual links to other articles because broader terms will, in having more generality, be linked with more other pages, although this relationship is not perfect. Lexical expansion can also be used to identify narrower topics.

Where more than one term means the same thing redirects are used to send users to the page with

---

8See http://www.wikipedia.org
the content about the topic. These redirects can be used to identify terms within the same synset. Redirects are also used from common spelling errors and this provides a feature not commonly found in thesauruses of identifying misspelt terms. Where a misspelt term could be corrected to more than one term the most likely is chosen. Wikipedia has recently moved to using a separate table for storing redirect information in its database, but this table is not at present included in the database dumps provided by Wikipedia but requests for the dumps to be included can be made. At present the redirect information can be found by looking at mutual links from pages labelled as redirects to other pages, although investigation shows that some redirect pages have more than one link from them on the scale of approximately 1300 pages out of more than 5,300,000 articles which could introduce some error. Thirteen hundred errors though being identifiable can be fixed with manual attenuation after a program has constructed the rest of the thesaurus.

Namespaces and categories

Pages on Wikipedia each have a single namespace that they fall into identifying the type of content they represent and used in resolution of links. Regular articles are in the main namespace 0 and the majority of the current 20 namespaces are used to label media and discussion pages. Namespaces can be seen on Wikipedia as the text before the colon in the last part of a page’s URL, for instance http://en.wikipedia.org/wiki/Wikipedia:Namespace is in the Wikipedia namespace which is namespace 14. Only two of the namespaces are of immediate use in generating a thesaurus. Namespace 0 (the regular articles) and namespace 14 categories.

Categories in Wikipedia form a taxonomy of concepts and types of pages and as such can be used to build type of relationships (hyponym and hypernym relationships). Inter-category links are separated from normal inter-page links. Pages are assigned categories to state points they fall within this taxonomy. For instance the Bacon page falls into the Cuts of pork, Breakfast foods, British cuisine and the Irish cuisine categories. Not all of the categories are of use to a thesaurus. The Wikipedia maintenance category and its subcategories do not relate pages to a meaning but flag the pages for requiring cleaning up or correction and the Wikipedia problems category and its subcategories identify controversial pages.

The Wikipedia administration category Disambiguation is a category that needs to be treated as a separate case. Page and categories which fall into this category represent terms that belong in more than one synset with the mutual links from the disambiguation page indicating the synsets the page falls into.
The use of the text of Wikipedia pages would bring the process of generating wikisauri closer to traditional generation of thesauruses from large bodies of text. But the style and formatting of Wikipedia articles is, as it is an encyclopedia, fairly consistent. A simple observation is that many articles begin with saying that the page title is a type of something. For instance the page on Creatine begins “Creatine is a NITROGENOUS ORGANIC ACID that naturally occurs in VERTEBRATES and helps supply energy to MUSCLE cells.” where the terms that are linked are shown here in small capitals.(Wikipedia 2007c) The highlighting of important terms by them being made into links also makes the articles easier to extract information from than typical sources of text and the words chosen to be turned into links can be used in some cases to discover more meaning. The ease at which information can be extracted from articles appears to decrease the further down the article is looked as the style varies more.

Coverage and maturity

Wikipedia by being contributed to by its users and its user base being in the millions has more than 75,000 active contributors. Wikipedia has rapid growth by allowing the users to be the contributors.(Wikipedia 2007e) This also means the terms and relationships used between pages, although fitting within Wikipedia’s style requirements, are based on how the concepts are organised from users perspectives providing a natural fit with users’ ideas, moving with the times and covering topics to the depth required by users.

Anecdotal evidence of Wikipedia’s growth came when writing this section on wikisauri in the time this section was written the size of the Wikipedia article on Creatine grew to be three times its original size.(Wikipedia 2007c) Pages typically begin as stub articles and if a user that feels they want to flesh out the article encounters it they will do. Pages do not grow indefinitely and reach a limit where changes are only in the form of updates or corrections with closely related concepts being encouraged to have their own articles for clarity. Mature pages are preferential for analysis because of the volume and accuracy of the information they contain, but stub pages still contribute value to the generation of a thesaurus as the head of articles often contain the most relevant information.

Of the approximately 5,300,000 articles in Wikipedia more than 1,700,000 are English articles and the other articles cover more than 100 languages. Many pages contain links to the same article in other languages providing a good basis for use of a wikisaurus as a device for translation and for semantic match making without concern for the language involved.(Wikipedia 2007e)

Milne et al. (2006) makes comparisons between a potential wikisaurus and Agrovoc, an agriculture thesaurus. The study found that Wikipedia covers 50% of the terms of Agrovoc and 72% of the terms actually used in agricultural articles that were also in Agrovoc. Although it can be said that Agrovoc’s coverage may grow over time Wikipedia’s coverage will likely grow much
faster and Milne et al. (2006) does not consider the use of the text within articles as a source for information which may uncover many of the terms considered missing. Milne et al. (2006) does argue that useable domain specific wikisauri can be constructed from Wikipedia. Coverage of technological and mathematical terms on the web are imaginably greater than agricultural terms because users familiar with these areas tend to be more able to make contributions in the form of pages and, relative here, Wikipedia articles. It is therefore a consideration that wikisauri may show better coverage of these areas than of agriculture. Domain specific versions of Wikipedia also exist that may provide better coverage still of the domains they specialise in and can potentially be linked with the original Wikipedia articles to make an large thesaurus that has both depth and breadth of coverage.

Abuse and errors

Wikipedia abuse is an issue that needs to be addressed in the development of wikisauri. Not all abused articles will be categorised as such as the authors of the abuse of articles obviously cannot be trusted to categorise the articles as abused themselves. Some abuse can be detected by looking for obscenity and punctuation common to abused articles, but some articles will genuinely have a reason to include these words, such as articles about obscenity. So a large portion of the abusive articles can be removed leaving a proportion of abused pages. Where abused articles can be detected Wikipedia does maintain the previous versions of the page to fall back on. Abuse and errors in popular Wikipedia pages should be quickly removed because of the pages traffic increasing the number of checks the page goes under. Controversial pages on, for example, political figures are less likely to be changed into accurate articles and instead alternate between the different extreme opinions of the controversy. Although the most controversial articles are sometimes frozen by Wikipedia staff to avoid too much abuse. Subtle errors and abuse in Wikipedia are at present impossible to detect but there is reason to believe that Wikipedia is little more prone to errors (which in this case includes abuse) than Britannica. (Giles 2005)

Technical issues with wikisauri creation

The current Wikipedia database dumps do not currently include all the tables that would be ideal for building a wikisaurus but the table data can be made available. Crawling the Wikipedia website to extract thesaurus information would take a long time and would be blocked before it was completed by the site owners. The database information is available under the GNU Free Documentation License and is far quicker to access and manipulate but is larger than 40GB in size as compressed SQL files, although just the link, category and page tables are under 1GB compressed. (Milne et al. 2006, Wikipedia 2007)

There are over 96 million links in the link table and importing this information, indexed, into a table takes approximately one week on a test machine that was new just over a year ago. Certain
optimisations can be made to speed this process up such as only indexing the tables after the data has been imported and in MySQL using a MyISAM storage engine, as opposed to the InnoDB storage engine prompted for in the SQL files. Large tables such as these are clearly costly to search and links do not refer to pages by their index in the pages table but by their namespace and their page title. This means for each link from a page more than 5 million string comparisons may have to take place to find the referred to page. When building a wikisaurus it is best to minimise the amount of searching for page titles that needs to be done. Better than working with the data in a database would be to store information like maps from page titles and namespaces to indexes in memory in hash maps. Because there is no set order for the data in the tables (except perhaps the newer items in tables being the least linked to) there is no way to determine which part of a table will be needed next. So it would appear a large amount of memory is necessary to build a wikisaurus without the process taking centuries. The centuries figure is approximately arrived at because after installing the database a test look up of a page took more than 5 minutes. Optimistically, if it could be said to have taken 1 minute there is 96 million links to look up the destination of meaning to look up the destination of each link would taken more than 182 years. Being able to quickly look up the pages by title would speed up the process dramatically.
Chapter 5

Testing and discussion

Here various choices and ideas are investigated and explored. The performance of thesauruses is observed, as well as the optimum distance over which to search for terms and which of the two algorithms performs best. We look at the design of an individual handler for extracting text from NAG library documentation. Then observe the effectiveness of the algorithms with a variety of queries. Then we explore a range of areas where the algorithms could be improved in the future.

In the tests performed here the accuracy of the results cannot be verified because how closely a query and each of the 848 NAG library description texts are in terms of meaning is not known. In each of the tests of the algorithms the descriptions that are expected to score well are checked that they do indeed score well. Some checks on random descriptions are performed to find insight into the reason they received the score they did against the given query, particularly those that scored well. At no point in the testing did the web service description that was expected to score well not score in the upper-quartile of results. Samples of the test results are included here with the remainder of the test results included on the accompanying CD.

5.1 Thesauruses

With three thesaurus classes being developed for the project, which to deploy in the matcher needs to be decided. It is already clear that WordNet has a much larger vocabulary of terms than the Connecting Mathematics thesaurus but does their combination offer anything more. Of the 3892 English terms in the Connecting Mathematics thesaurus 359 were found to be missing from WordNet. These terms contained five stop words that would have been filtered out but the remainder of the missing terms are too specialised for the more general WordNet to cover such
Figure 5.1: Thesauruses: The scores in rank order using the query-to-description scorer
as “vigintiangular”\(^1\) and “contrapositive”.

When running the algorithms using each of the thesauruses the degree of difference between the three is indicated. Some queries that use terms that are not so related to maths such as “sort an array” produce almost no results for the Connecting Mathematics thesaurus whereas with the query “calculate all the eigenvalues and eigenvectors of a real symmetric matrix” the Connecting Mathematics thesaurus performs much better. The ranked results from running that query using the query-to-description scorer with a depth of 7 is shown in Figure 5.1. Here it can be seen that the only matching done by the Connecting Mathematics thesaurus is with the synsets that contain the terms in the query itself, rather than a synset reached by taking a step away towards hypernyms for example. It is clear from the scores being shown that hypernyms and/or hyponyms of terms are being found in descriptions using the two WordNet based thesauruses.

The two WordNet based thesauruses results are so close that in the graph most of the points for the solely WordNet thesaurus are obscured. The graph shows around rank 300 and around rank 630 descriptions that score higher with the WordNet thesaurus than with the WordNet and Connecting Mathematics (WNCM) thesaurus. This is unexpected because the WNCM contains all the terms and relations of the Wordnet thesaurus and only uses the Connecting Mathematics thesaurus when WordNet cannot find a term in the query. This unexpected result is not understood and attempts to debug the code show no reason for this to happen. Other than this unexplained discrepancy the results for the two WordNet based thesauruses are identical. Based on tests run either of the WordNet based thesauruses seem useable. The WordNet data provides a strong general grounding and a very broad vocabulary that is missing from the single domain-specific thesaurus. In theory the WNCM thesaurus should provide a greater coverage of terms (an extra 359) but evidence seems to show that WordNet on its own seems to find terms that WNCM does not and this should be further investigated. The difference between the two seems almost insignificant so we have decided to use the combined WordNet and Connecting Mathematics thesaurus for all the subsequent tests.

A change to the thesaurus used for the project would be an obvious area for improvement. One that allowed filtering of terms by sets of contexts. This could allow the memory footprint of using the thesaurus to be reduced, if terms in unused contexts could be selected and removed. The algorithms would then need adapting to only search for terms in particular contexts (and a general context) either specified by the user or calculated by finding the contexts shared by the terms in the query. Section 5.7.3 covers this area in some more detail.

A thesaurus that has more information in the perspective of the terms covered has always been a desire of this project to obtain or create. The current solution provides a very general coverage of terms using WordNet but the Connecting Mathematics thesaurus does not add as much specialised information as would be liked to it. It is likely the case that many of the terms within WordNet fall into a context that are never going to be used by the matcher. The example of the

\(^1\)Something which has 20 angles
entries for the term “tree” in WordNet in Section 4.1.1 illustrates this as web service descriptions and user’s queries are not imaginably going to search for “Sir Herbert Beerbohm Tree”. Instead of a great volume of terms, more desirable is good coverage of specialised domains.

Our investigation of the wikisauri in Section 4.2.3 revealed one way in which a thesaurus could be generated that would be based on the usage of terms by people in the field although the coverage has been shown to be not quite that of specialised thesauruses. Wikipedia’s category system also provides contexts that terms may be classified by but it should be noted that wikisauri generation is not something that has been successfully completed as of yet and work is on going in this area. So although we have considered an outline of how to generate a thesaurus from Wikipedia the work is at least a project in itself and requires much in the way of computing resources to be achieved in a reasonable time period.

The Connecting Mathematics thesaurus supplements the WordNet thesaurus and it is conceivable that the WordNet thesaurus would be useful for terms that are not domain specific. Thesauruses from a variety of subjects could be used alongside WordNet to provide the best coverage of terms. But the current combination of the Connecting Mathematics thesaurus and the WordNet thesaurus does not provide enough extra coverage of terms to make the matcher anything more than a general matcher without much in the way of specialised mathematics knowledge. Care needs to be taken in the combination of thesauruses in how the overlap between two thesauruses is handled. Ideally thesauri could be combined into a single index of terms, synsets and their relationships, otherwise on reaching the end of the information in one thesaurus a transition into the other thesauruses needs to take place. It is expensive for each term in each synset in the starting thesaurus to search for overlap in the other thesaurus but the overlap must be detected as soon as it occurs to avoid the missing of terms within synsets that the second thesaurus has better coverage of. Therefore precomputing the overlap and creating a single combined thesaurus provides the best solution. But this has not been done in our combination of the two thesauruses. At present the implementation searches for terms in a single one of the two thesauruses, falling back on the Connecting Mathematics thesaurus if the WordNet thesaurus does not contain the given base term.

5.2 Distance to travel

The distance to travel, or depth, from the source term to search with the hypernym and hyponym relationships is a property that effects the run-time of the matcher and the results it provides. If the hypernym relationships are travelled along too far the same terms, those at the top of the hypernym relationships within the thesaurus, will be reached for every given term.
Figure 5.2: Depth: The scores in rank order using the query-to-description scorer
The weighting system used in the two scorers, where different relationships multiply a number by the score of the synset that they are from and that score is given to the synsets reached, seems a natural fit for a generalised view of the relationships. Using multiplication means a negative number will never be reached as a score for finding a term and the scores of synsets are scaled relative to each other. This also means that after a number of multiplications, with no weighting being greater or equal to 1 the score given to an term will be so small as to be insignificant, limiting the depth. There is a strong case for assigning different depth values to hypernym relationships than hyponym relationships but simplicity dictated that this has not been done in the scorers worked with here, although it appears that gains can be made from such fine tuning. The two algorithms require the depth to be the same for hypernyms as it is for hyponyms because one goes away from the terms in the query and the other goes towards the terms in the query from perceived terms in the description and as such the traversal of the relationships is reversed.

The difference in meaning experience between two synsets at the end of relationships varies within the relationship type as well as between relationship types. There tends to be only one hypernym relationship from synsets because of the tree-like nature of meaning from this perspective but none the less language does not impose a set amount of difference in meaning between synsets separated by given relationships. So for some synsets ten steps away from them will still be a synset with similar enough (for the user’s purposes) meaning, whereas for others any steps at all go too far.

It has been presumed that users choose terms based on that being what they want and as such it can be argued that stepping many steps away is an increasing source of inaccuracy and wasted effort. The results in Figure 5.2 show the expected clustering of results for different weightings after about three or four steps. The graph is based on running the query-to-description scorer with the WNCM thesaurus with the query “sort a vector of objects into ascending order”. The grouping of the graph points occurs earlier than expected because with hyponyms weighted with a multiplier of 0.85 a term found at the end of a 5 deep walk along hyponym relationships should provide a score for a term found of 0.4437 and as such would be noticeable if a term was found at such a distance.

As the distance travelled has an influence on the time taken by the matcher a graph of times is shown in Figure 5.3 using the same queries as with the previous graph. The graph shows the average time taken to run the scorer with the given depth and the average time taken when no matches of terms were found, this being a worst case as searches will have to run to their limit, as set by the depth. Occasionally and without following a pattern anomalous large times would appear in the results output. The occurrences of these anomalies was approximately evenly distributed across each of the depths tested. But as the time taken and depth travelled increased the number of worst cases dropped so the impact of anomalous times on the worst case average increases. This is why a version of the worst case times where the anomalies are ignored is also provided. The reason behind these anomalous times is not known and could be a process in the
Java virtual machine or in the WordNet thesaurus library but its inconsistent timing when running tests appears to indicate it is not directly tied to any program code. The times seem to become noticeably much worst at a depth of around 5 or 4 and strangely the overall averages appear to almost flatten out after the period of acceleration in time added per unit distance travelled. This could be because at such points in the thesaurus at a distance from the, perhaps, general terms of the query the number of synsets to travel to is low.

From both the times and the graphs of scores created it appears that using a depth of 3 or 4 will provide access to enough of a spread of meaning and in a short period of time. A depth of 3 has been chosen to be used in the remaining tests.
Figure 5.4: Algorithms: The scores for each description for the two algorithms
Figure 5.5: Algorithms: The scores in rank order using both algorithms
5.3 Algorithms

The two algorithms developed are very similar but approach the term acquiring process from different ends. Figure 5.4 illustrates where the scores differ. This graph was produced with the query “calculate all the eigenvalues and eigenvectors of a real symmetric matrix” and the WNCM thesaurus using a depth of 3. Differences in the scores are visible and are artefacts of the differences in how the searching within the description for terms is done. The most the two scorers differ with this query is 0.28708. The average difference between the two is under 1.5%.

Figure 5.5 uses the same data as Figure 5.4 and shows that the description-to-query algorithm appears to find more relationships between possible terms in the description and terms in the query. This appears to come from better handling of characters separating words in terms in the description and possibly because of the WordNet thesaurus sometimes being overly keen with its identification of terms. Testing showed that the checks to see whether sets of words in the description were in fact terms would often find terms where terms did not exist because the WordNet library would strip away words such as stop words or match to the closest term it had. This over-keenness does not necessarily cause relationships between the query and description to be missed though because terms found in the thesaurus should be found with some basis from the input words and not just at random. Relationships found between terms can be presumed correct if they exist. Incorrectly identified terms are not statistically significant so relationships found between them can be seen as appropriate connections of meaning. Although the meaning may be inappropriate for the context of the term and the intended meaning of the term in both the query and the description, any description that scores well for each of the terms in the query will for a query of more than a few words have a significant overlap in context with the query.

The cases where the query-to-description algorithm is able to better find terms with relationships in the query and description are thought to be cases where the over-keenness of the description-to-query scorer causes it to identify the first half of a term as part of another term and as such it does not manage to match it against anything in the query. Had the term intended by the description’s author been reached and searched for the match would have been made as it is done in the query-to-description scorer and this is where the over-keenness causes problems. An example of where the query-to-description scorer outperforms the description-to-query scorer can be seen in the average scores for the second query in Table 5.1. The description-to-query scorer can also suffer from a problem referred to as blocking and discussed in Section 4.1.2.

By its design the description-to-query algorithm should take much longer because it branches from many more terms than are intended and the web service descriptions tend to be longer than the users’ query so more terms have to be branched from. Sample averages from three queries using each of the matchers is shown in Table 5.1. The three queries are the query used for the previous two figures, “rearrange a vector of arbitrary type objects into ascending order” and “The basic tool for the manipulation of reality is the manipulation of words. If you can control the meaning of words, you can control the people who must use the words”, a Philip K. Dick
CHAPTER 5. TESTING AND DISCUSSION

quote. The Philip K. Dick quote is included because it is not a query that is expected and should not score very highly against the descriptions and it is also a reasonably long query string.

Table 5.1: Time and score statistics of the algorithms

<table>
<thead>
<tr>
<th>Query-to-description scorer</th>
<th></th>
<th></th>
<th></th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query number</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>Combined</td>
</tr>
<tr>
<td>Terms in query</td>
<td>6</td>
<td>7</td>
<td>12</td>
<td>8.3</td>
</tr>
<tr>
<td>Average score</td>
<td>0.33818</td>
<td>0.32053</td>
<td>0.25319</td>
<td>0.30397</td>
</tr>
<tr>
<td>Average time (usecs)</td>
<td>4305.4</td>
<td>20365.1</td>
<td>36924.3</td>
<td>20531.6</td>
</tr>
<tr>
<td>Average miss time without anomalies (usecs)</td>
<td>1738.3</td>
<td>25185.1</td>
<td>29350.9</td>
<td>18758.1</td>
</tr>
</tbody>
</table>

| Description-to-query scorer |         |         |         | Comb | m | | e | e |
|----------------------------|---------|---------|---------|------|------|------|
| Query number               | 1       | 2       | 3       | Combined |
| Terms in query             | 6       | 7       | 12      | 8.3 |
| Average score              | 0.35425 | 0.24635 | 0.34823 | 0.31628 |
| Average time (usecs)       | 219312.9 | 233171.0 | 366275 | 272919.9 |
| Average miss time without anomalies (usecs) | 30754.3 | 20702.1 | 25728.2 |

The difference in time taken between the two algorithms can be clearly seen with the description-to-query scorer taking more than ten times longer on average. The average time of the query-to-description algorithm appears to increase as the number of non-stop word terms in the query increase but not proportionally so although a relationship may be present from the data it appears the number of terms is not the only factor playing a role. The times for the description-to-query scorer appear to increase with the terms in the query as well and part of the algorithm will depend on the number of terms in the query. But it is a less significant part of the algorithm than the part dependent on the number of terms in the query-to-description scorer. Times from testing show that the average times for the description-to-query scorer are consistently around the same 0.3 of a second mark as this is more dependent on the length of the descriptions rather than the query. There is no third average miss time for the description-to-query scorer because despite the intended extraneousness of the third query each description was found by the matcher to have some overlap in meaning with it. This may show evidence of problems with the over-keenness in the description-to-query scorer.

Both algorithms are reasonably similar but both have flaws. The query-to-description algorithm is relatively quick but misses some connections because of how it searches for terms in the description. The description-to-query algorithm is much slower and as much as it searches for many more terms for each query and description pair it manages to miss terms through its over-keenness to identify terms. The over-keenness is a flaw in the thesaurus library being used and the algorithm because the thesaurus is responsible for identifying single terms where pairs of
terms may exist and the algorithm, in not checking for overlapping terms with the terms it finds in the thesaurus, misses terms. The time taken by the description-to-query scorer is significant and foreseeably more changes need to be made to that algorithm to improve it than the other, which needs to perhaps just use better string searching techniques but aside from that it is not so flawed. So the query-to-description algorithm will be the algorithm used for illustration in the subsequent tests.

5.4 Parts of the NAG library descriptions

The matcher allows handlers to be written for different types of web service descriptions that it is supplied with. The test data for the text semantics matcher is the NAG library descriptions which are XML documents containing nine sections describing functions that have been converted into 848 web services. The nine sections are labelled “Purpose”, “Specification”, “Description”, “References”, “Parameters”, “Errors”, “Accuracy”, “Further Comments” and “Example”. Manual observation of the nine sections indicated that the first, third and last sections had the closest and best formatted meaning for the web services they represented. The sections used have the contents of their <maths> and <formula> elements removed because the contents of these elements is mathematical expressions and it is not the intent of the matcher to understand this. Characters representing variables in formulae that are within these tags are too numerous and interchangeable to pass onto the scoring algorithm.

Figure 5.6 shows the scores for each of three different sets of sections of the NAG documentation being used on query 2 from Section 5.3 by the query-to-description algorithm. It shows not much is to be gained from the use of the of the “Example” section’s text but the “Description” section has many terms that add to those from the “Purpose” section increasing the score the web service receives. The “Purpose” section is a relatively short section that summarises the operation of the web service. It can be seen from the graph that a number of web services are still recognised by the matcher with just the “Purpose” provided. But the “Description” is a much longer section that provides, as the name implies, a description of what exactly the web service does. Using the extra information available in the “Description” carries a risk that the extra information could contain a list of things the web service cannot do. As the matcher does not account for terms that operate on other words in its current design this would create false positives. But the matches appear to be grounded in terms used in a positive sense from looking at samples of results. The web services that closely match the query though should often only need the “Purpose” to score highly as the matcher seems to miss some terms in the way it searches within the description the inclusion of the “Description” information appears to offset this flaw in the algorithm. So, presently we can see a case for the use of both the “Purpose” and “Description” sections as the third considered section appears to add little new terms. A matcher could be considered very effective if it could reveal the closest matching web services using just the “Purpose” section.
Figure 5.6: The distributions of scores for different amounts of the description
Figure 5.7: The distributions of scores for queries of different detail
Figure 5.8: The distributions of scores for random queries
Figure 5.9: The distributions of scores for unlikely queries
5.5 Distribution of results

The scores illustrated in the figures discussed here are generated using the “Purpose” and “Description” sections of the NAG web service descriptions being scored against the given queries with a distance to travel of four using the query-to-description scorer.

5.5.1 Specificity of queries

In Figure 5.7 the scores from a series of queries are shown where the queries go from describing a precise desire about what the web service should do to a much more vague notion within which the more specific notions are encompassed. This illustrates how the more specific a query is the easier it is to select the web services that match the query using this method. So users should be made aware that the better they define what they want in their query the better results they should receive. The fewer terms that exist in the query the more clustering exists in the results because of the fewer possible scores that a web service can receive.

5.5.2 A variety of queries

Figure 5.8 indicates that the matcher works with variety of different queries. The highest scoring twenty web service descriptions for each query all score as expected. The web services descriptions that the queries were based on were found to be amongst the highest scorers for each of the tests.

5.5.3 Unlikely queries

Figure 5.9 shows the results of submitting five unlikely queries. The queries used are the following quotes:

1. “There is much pleasure to be gained from useless knowledge.” - Bertrand Russell
2. “The word ‘meaningful’ when used today is nearly always meaningless.” - Paul Johnson
3. “The basic tool for the manipulation of reality is the manipulation of words. If you can control the meaning of words, you can control the people who must use the words.” - Philip K. Dick, 1978, How To Build A Universe That Doesn’t Fall Apart Two Days Later
4. “I have no objection on principle to make to the guillotine. Nature, my only mistress and my only instructress, certainly offers me no suggestion to the effect that a man’s life is of any value; on the contrary, she teaches in all kinds of ways that it is of none. The sole end
and object of living beings seems to be to serve as food for other beings destined to the same end. Murder is of natural right; therefore, the penalty of death is lawful, on condition it is exercised from no motives either of virtue or of justice, but by necessity or to gain some profit thereby. However, I must have perverse instincts, for I sicken to see blood flow, and this defect of character all my philosophy has failed so far to correct.” - Anatole France, 1912, The Gods are Thirsty

5. “A military operation involves deception. Even though you are competent, appear to be incompetent. Though effective, appear to be ineffective.” - Sun Tzu, 6th century BC, The Art of War

This set of queries provides us with an idea about how high a score can be achieved unintentionally. The Philip K. Dick quote (Query 3) is the highest scoring of the five quotes with a maximum score of 0.8914583. The highest the other queries reach is just over 0.6. The results show that the cut off point for a close match on meaning has to be over 0.8 to avoid unintended matches, but the results in Figure 5.8 indicate with far more likely queries that a potential cut off point for good results could lie between somewhere between 0.6 and 0.8. A cut off point is hard to advise for matchers such as the text semantics matcher where how spread the similarity in the meaning of web service descriptions and the users’ queries varies with both the set of web service descriptions chosen and the queries compared with them.

Why Query 3 got such a high score is worth investigating. The text of the “Purpose” and “Description” section of the high scoring web service description contains 7033 characters and more than 1264 words. The longer the query is the more precise a concept is often described and the harder it is for a description to get a score for each of the query’s terms. But the longer a description is the more terms it has with which to find the terms in the query and the broader terms the coverage of the description is likely to have. Table 5.2 shows how the score achieved by each of the terms in the query was reached. The matched terms are shown in bold. Here it is worth knowing that the following weightings combined using multiplication are used: hyponyms 0.85, holonyms 0.3, meronyms 0.3, derivatives 0.3, “see alsos” 0.2.

Many of the words in the query have a synonym in the text of the web service description. The matching of “people” appears to be because the thesaurus identified that the intended term could be “free people” which is clearly a problem that needs to be resolved. The match with ”words” is as in ”they had words with the referee” and control is matched as a synonym for “check”. The other terms connections are clear. This shows the limits of this simple algorithm. A solution to the over-keenness problem would improve the spread of results but the lack of contextual understanding between the text and the thesaurus means matches are being made which could be eliminated if an understanding was there.

The query consisted of eight unique terms out of a total of twelve terms in the query. The repetition of terms makes those terms more important to the score and as such all the repeated terms got a maximum score for the match they made with a term in the description so the repetition
of them increased the average score. The query-to-description matcher is not at present set up to treat repeated terms in the query differently by remembering what score they were awarded before so repeated terms are just as costly in time as other terms are.

### 5.6 Anomalous results

Some investigation was carried out into the origin of the anomalous results that were appearing in the timing of the results. As it appears with both scorers it is not a phenomenon particular to the code of one of them. Nothing in the code of either scorer should cause the fifty times increase in execution time of the algorithm unless it was within a library routine called by both of them. Such an increase in execution time was experienced with just the Connecting Mathematics thesaurus but far less often is it seen because that thesaurus is far quicker to use because of its smaller vocabulary and simple structure. Similarly the lower depths experienced fewer of the anomalies because smaller depth values make the algorithms execute faster. Observing the times in rows of the same web service description shows their no relationship between the description and the anomalous times although the length does have some effect on the time. Similarly the anomalous times occur regardless of the query. So it has been concluded that the anomalous times are a feature of either the Java Virtual Machine or the operating environment. For the record the virtual machine used during the testing process is the Sun Java Virtual Machine 1.5.0.08 on Ubuntu Linux 6.10 i686, kernel 2.6.17-11.

<table>
<thead>
<tr>
<th>Query term</th>
<th>Count</th>
<th>Step 0</th>
<th>Step 1</th>
<th>Step 2</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic</td>
<td>1</td>
<td>basic</td>
<td></td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>tool</td>
<td>1</td>
<td>tool</td>
<td>hand tool</td>
<td>square</td>
<td>0.7225</td>
</tr>
<tr>
<td>manipulation</td>
<td>2</td>
<td></td>
<td>handling</td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>reality</td>
<td>1</td>
<td></td>
<td>actuality</td>
<td>state</td>
<td>0.125</td>
</tr>
<tr>
<td>words</td>
<td>3</td>
<td></td>
<td>row</td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>control</td>
<td>2</td>
<td></td>
<td>see</td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>meaning</td>
<td>1</td>
<td></td>
<td>mean</td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>people</td>
<td>1</td>
<td></td>
<td>free</td>
<td></td>
<td>0.85</td>
</tr>
</tbody>
</table>
5.7 Considerations

5.7.1 Weighting

The timescale has not allowed the weightings to be tuned as we would have liked. Some information for comparison on the closeness of the meaning of queries and the descriptions would have allowed the weightings to have been tuned automatically. Running tests with altered weightings produced no evidence of how to change the weightings. On a term by term basis in the results the weightings seem to be as close to appropriate as general weightings can be, being as they are based on principles that were outlined in Section 4.1.1. Although the ordering of the size of the weightings seems appropriate finer tuning can make a difference to the accuracy of results. The values chosen for the weightings are stated in Section 5.5.3.

The weighting system implemented abstracts how the scores of synsets are arrived at to be a function of the synset being branched from and the relationship that it has to the synset to generate the score for. Future versions could experiment with not using multiplication to generate the weightings but multiplication was chosen as it made the weightings proportionate to one another and allowed them to approach but not reach or pass zero. Negative weightings are undesirable because they imply a match of opposing meaning is found and that is not something which is searched for by the algorithms.

Ideally the weighting given to synsets would be specific to the closeness of meaning of that synset to the synset from the query. But thesauruses with large and deep vocabularies with weighted relationships do not appear to be distributed as yet. The wikisauri generation discussed in Section 4.2.3 appears a good way to generate thesauruses with a reasonable measure of the semantic similarity of concepts. Terms in synsets do not have identical meaning because of terms being defined by their usage. Different terms with theoretically the same meaning are used in different situations and because of the subtleties of their meaning that have been instilled in the user. Similarly a single term that is in a single synset can vary its distance from other terms by how it is used. But as the current generalised system works as an approximation, a per relationship weighting although still an approximation would be a significant improvement over it.

The weighting system as it is presently implemented works within the confines allowed of the depth argument given to the scoring algorithm. The depth argument is designed to limit the number of relationships traversed by the algorithm but in doing so causes inconsistencies with the weights as they are assigned at present. With a depth value of three as the tests appear to recommend the furthest reachable synset along branches of hyponyms, that if a term within it was found in the descriptions text, would receive a score of $0.85^3$ which equates to 0.614125. This is a higher score than is awarded to all the terms in synsets reachable by non-hyponym paths as is $0.85^4$. It seems unreasonable that would be scored as if they have much closer meaning to the source term are discounted because of a general limit. After four hyponym steps away from
the source term the weightings suppose that the terms there have half the meaning of the original
term yet these terms are ignored. The depth variable only limits the hypernym and hyponym
relationships as these are the only relationships where steps beyond the first are deemed to retain
meaning close enough to the source term to be worth searching for. As such it would seem
appropriate in future versions to define a separate limit for the hyponym relationships as the limit
of three is appropriate for the hypernym relationship steps because 0.5^3 is 0.125 and is therefore
smaller than the other weightings given to synsets. Using this idea the ideal distance to travel
along the hyponym relationships is around ten or eleven steps. But walking this far from terms in
WordNet tends to if that length of relationships exists take the synset to be far more specialised
than could imaginably be appropriate as a replacement for the source term. Study of WordNet
supports the case for the depth of three or four being chosen for both hypernym and hyponym
relationships but then the values of the relationships which seem appropriate are not cut off at
consistent levels so perhaps using constant values for the weightings is where the error lies and
the weighting for subsequent steps should be reduced so that the apparent change of meaning
with distance is appropriately represented. This is another aspect where the algorithm can be
tuned for improved results.

The values of the weightings may in fact be too low. On the scale of how similar meanings are
a term within one step of a synset in a thesaurus will have relatively, compared to the number of
steps that could be traversed to reach other terms, have a relatively similar meaning. But working
on such a scale would require deeper searches of the thesaurus and result in narrower spread of
results as the descriptions of the web services from the repositories chosen by the user are likely
to have a common theme.

5.7.2 Fixing over-keenness

If the thesaurus could be made to only identify misspellings and actual terms, not terms within
longer sets of words or terms that the given term is only part of, then the algorithms would operate
to their design. It appears thesauruses is an area where much further work can be done. The
licensing of the JWNL currently being used to interface with WordNet allows such alterations to
be made to it although there is a limit to how far this library and the WordNet data itself can be
taken for optimum semantic scoring. A simple solution might be to check the searched for term
exists within the returned synset but this may suffer from the removal of correctly matched terms
where the form of the term is necessarily different. Commonly this error may occur where the
given term is a different tense or in a different person or plurality.

5.7.3 Removing polysemy

At present the matcher may benefit from not identifying from the structure of the sentence what
part of speech it should be finding the word in. This means a broader set of terms are used for
matching which can be a mixed blessing because sometimes the term as any part of speech is appropriate for getting a match but in other cases it introduces concepts that are entirely unrelated and only serve to weaken the quality of the match if they are found. So there is an argument in favour of detecting the part of speech that a term corresponds to in both the query and the description to shrink the search space. This would make the parsing stages a slower part of the algorithm but this may be offset by the reduced number of thesaurus synsets that have to be branched from. Part of speech detection techniques are not guaranteed to correctly detect the part of speech in sentences. But an algorithm that provided a confidence value with the result could allow the term to be used in a single part of speech only if the confidence in guessing the correct part of speech was above a given value. (Dickinson & Meurers 2003)

Parts of speech are one dimension of vocabularies and the context within which concepts are used is another. If the context of a query could be detected, be it simply through labelling of queries by the user or by looking for the common context of the synsets of the terms in the query, then the synsets that were to be branched from could be limited by their context and the search space would be much reduced. Without the user labelling the query with a context the query would have to be parsed once to get a set of synsets for each term along with the contexts of the synsets from which the common contexts could be determined and the synsets in the common contexts could then be branched from. This would also require a thesaurus with context labelled synsets. The investigation into wikisaui in Section 4.2.3 shows that the category labels given to pages could be used to group synsets and the hierarchical nature of these categories is of use in unifying contexts. Some terms may have high generalised usage and their context should be ignored to avoid confusion over the intended context.

5.7.4 Term occurrence

Originally where a hash table contained a count of the number of times each term occurs in the description the algorithm was to use the number of occurrences of the term to effect the value returned. One such scheme that could work effectively with this information requires knowledge of how common a term is expected to be and would allow the query to be matched emphasising the most unique terms of the terms that it contains. The use of the count of term occurrences is not straight forward to implement as the relationship between the number of occurrences and similarity of meaning is not linear. A short description might have one occurrence of each term in the query (or term in the same synset as a term in the query) so the match would seem to be perfect and without the use of the popularity of each term this would be easily reflected in the score returned. In a description far longer than the query there is only a small chance of the desired popularity of terms being resembled by their occurrence in the description. How common a term is could be knowledge from the thesaurus used or generated by parsing the web service descriptions to establish the relative occurrence of terms. Parsing the web service descriptions requires the matcher to be kept up to date with the latest web service descriptions to be published.
Although similar text could be used to reduce the chance of error such as function descriptions from libraries as these will have a similar style of writing.

The occurrence of terms in the web service description could be scored for being proportionate to the users’ use of the terms in the query allowing the user to be the provider of the information on how commonly a term occurs. Use of such a scheme would require extra complication for the user for marginally little gain and with this scheme and the scheme above there is obvious doubts about how consistent ratios of term occurrence imply similar meaning. There are far more clear cut methods of determining the similarity of meaning than occurrence information.

A third possible usage of occurrence data would be to know what proportion of the web service description’s terms that the matched terms were. With descriptions that the query covered well being awarded higher scores because that should imply a closer match between what the description and query describe. At present the number of terms of the description is not accounted for as it is in some methods (see Bhattacharyya et al. (2002)) because the web service description can contain terms matching the query within it along with any number of other terms that may or may not be relevant. The user is not expected to enter as a query how they expect the web service description to be written.

5.7.5 Tuning and human computation

A significant problem with the testing of this project is the lack of information on how close the meaning of queries and descriptions is so the exact correctness of matches cannot be determined. The algorithm could be tested against existing scoring systems such as Bhattacharyya et al. (2002) to provide some comparison. Although the quality of the scoring with the deliberately simple and quick algorithm is not expected to be high by comparing it to existing scoring systems the more accurate the algorithms used for comparison the more accurate the algorithm can be changed and tuned to be. Areas for tuning, namely the weightings, have been identified and using target data to tune the algorithm to achieve as close to as possible genetic algorithms could be used to find the optimum solution for achieving the desired scores.

Using the algorithms of others for tuning is a degree removed from the actual similarity of terms as users will believe it to be. So if how closely users believed two sets of text to be could be found then the detail required in the algorithm to replicate it may increase dramatically but the accuracy could be tuned to be much better using the data. Human computation techniques such as those outlined by von Ahn et al. (2006) where users in playing games reveal information to observers that when used on popular enough sites can generate the information required for tuning algorithms accurately. A simple survey of individuals would be unlikely to generate enough data. Though the relative lengths of web service descriptions would make it harder to get user opinions on the similarity of their meaning to other text using either the descriptions or similarly long texts a more optimal algorithm could work over shorter parts of web service descriptions such as
just the “Purpose” sections of NAG library descriptions and not the “Purpose” and “Description” sections as currently deployed.

### 5.7.6 Context not meaning

The results show the algorithm is able to find web services with at least a similar meaning to the given query and in a reasonably short period of time with the highest scoring web service descriptions being found to be the most appropriate after manual checking. A large number of less appropriate web services though still manage to score highly. Mostly because the difference in meaning is not that great and because of matches using meaning that were not intended by the author of the query or using only parts of terms in the web service description (or both). A more intelligent method for detecting terms in the description would allow the second cause to be minimised but the first cause requires the ideas discussed in Section 5.7.3.

The tests that we have been able to perform show the matcher works but the spread of results is not as expected as the web service descriptions should be more often noted as having very little similarity from the query. It appears as if the matcher as it is detects not semantic similarity but contextual similarity with the higher the overlap in the apparent meaning of individual terms the closer the context that the two sets of text fall into. Hence with the web service descriptions all having high contextual similarity they all have a high similarity to the description of a mathematical idea in the query string. But the results for unlikely queries illustrated in Section 5.5.3 show that reasonably high scores are achievable with entirely random queries. Although specific enough queries using deliberately non-vague terms minimise mismatches encountered. The algorithm we believe can be fairly accused of being over simple.
Part III

Trust and relationships
Chapter 6

Aims and expectations

Here we outline what our immediate aims are for this matcher and infer requirements from that.

In Section 1.4 the idea for a system to use users opinions to rate web services was outlined. The idea is to allow users to say how much they like each web service. Then, because users should not rate web services they have not tried, where a user had not rated a web service their opinion on the web service is inferred from other users. This is to be done by allowing users to say how similar their opinions were to other users (how much they trust their opinions to be correct) and then using these other users opinions to infer what the original user would have thought of the web service. The inference process can then be repeated to find opinions on web services the users that the original user trusted do not have direct opinions for by inferring these users opinions as well and so on.

This requires a way of users saying how similar their opinions are to each other. This is in a sense a form of social network, where relationships between users exist where they believe they share opinions and these most straightforward way to provide users with an accessible way to take part in social networks would be by creating a website. How users should declare how much they agree or trust the opinion of other users then needs to be decided and how users should express how they feel about web services. Once this network of information is in place ways need to be developed to as accurately as possible infer users opinions on web services.

There is not the time available for this project to find enough people that use web services to test the social network and the accuracy of the matcher’s algorithm so ways to generate social networks with virtual users will have to be found. As there will undoubtedly be some variance in the structure and density of social networks it will have to be investigated how these changes effect the algorithm. The answers to some of these questions are discussed in the following review of literature.
CHAPTER 6. AIMS AND EXPECTATIONS

6.1 Grimoires

Grimoires (Grid Registry with Metadata Oriented Interface: Robustness, Efficiency, Security)\(^1\) is a UDDI implementing Web service registry project that enables metadata to be used for annotation of services. Grimoires (or GRIMOIRE) is funded by the Open Middleware Infrastructure Institute (OMII)\(^2\) an organization that aims to support the future of the e-Science community. The KNOOGLE project is also supported by OMII and as such Grimoires is the registry service used with the KNOOGLE project. A social networking website will have to allow some way for users to have web services on the site available to review. One way is to allow users to submit web services but this can lead to abuse and puts the onus on users to provide services to be reviewed. A better alternative is to extract details of web services from the registries, in this case Grimoires registries. This would mean the services available to be reviewed were the same services users will put up for consideration by against their queries with the matchers. So a program must be put in place to allow this extraction to happen to give the users the web services to review.

\(^{1}\)See http://grimoires.org
\(^{2}\)See http://www.omii.ac.uk
Chapter 7

Literature Survey

In this literature survey we look at how social networks are structured and develop. Then how trust operates, can be inferred, and can be represented for a matcher to reason with. Ending with how opinions can be dealt with.

7.1 Social Networks

The term social network was first used in Class and committees in a Norwegian island parish by Barnes (1954) although now it is in common usage on the world wide web. Golbeck (2005a) defines a web based social network as being accessible over a web browser, users being able to explicitly state their social connections with other users and relationships being visible and browsable. Social networks can form part of the semantic web, describing relationships between people and individuals’ relationships to things.

Successful social networks, grow rapidly and there are many social networks on the web based around a variety of themes, such as romance, entertainment, religion, and combinations of such. (Golbeck 2005a) Golbeck (2005a) found the most popular use for social networks was entertainment (approximately 70,000,000 members across 55 sites), followed by dating (approximately 49 million members across 23 sites), then blogging (approximately 5.7 million members across 5 sites). Although an area of social networking classified by Golbeck (2005a) as business had 16 sites and 3,300,000 members between them. The most popular social networking sites are not based around web services or anything closely related. As an example of the scale of the growth social networking websites are under going MySpace.com1 now has more than 100,000,000 members with 230,000 joining daily up from the 6 million listed by Golbeck (2005a). So leveraging existing social networks would be beneficial and the growth in social

1See http://www.myspace.com
networks shows that there is an increasing number of web users interested in making these social connections. (Sellers 2006)

### 7.1.1 FOAF documents

Friend of a friend documents or FOAF documents (pronounced “foaf”, to rhyme with “loaf”) as they are more often referred to as, are the most common, open way of representing social network data. (Brickley & Miller 2006) FOAF documents are RDF (Resource Description Framework) documents defined using OWL (the Web Ontology Language) allowing users to manage information about themselves and others. (Golbeck et al. 2003) RDF documents are designed to describe resources, such as people using “triples” each specifying a subject, a predicate and an object to create a directed graph that can be easily walked and analysed by programs. (RDF Core Working Group 2004). OWL is a language used to describe ontologies for use in the semantic web and can give meaning to the nodes in the graphs created from the RDF documents. (Web Ontology Working Group 2004) The semantic web being a distributed representation of a large graph connecting resources and literals with predicates implying meaning, in a format that machines can easily interpret.

An alternative to FOAF for representing social connections is the XHTML Friends Network or XFN which identifies individuals by web addresses (mostly weblog addresses) and simply adds a rel attribute to <a> tags in web pages, where the value of the attribute lists the relationships the target of the link has to the creator of the link. This makes contributing to social networks easy to anyone who creates links and requires no separate files. But does not provide the depth and flexibility of FOAF documents for extending it to contain opinions of web services for instance and is not so well adopted - the site at the centre of promoting it provides a list of individuals whose sites use the additional attribute, with no notably large social networks listed. (Global Multimedia Protocols Group 2006) Another reason behind the lack of adoption behind the XFN representation is that many web users, even if they do have a website or weblog do not bother themselves with manually editing the attributes of links they publish.

Using the Google\(^2\) search engine, Ding (2006) was able to estimate the size of the semantic web at between 10,000,000 and 1,000,000,000 documents, although this range shows how precise an estimation can be made the value depends on the definition used for semantic web documents. Some semantic web documents such as the FOAF documents generated by LiveJournal\(^3\), DeadJournal\(^4\), ecademy.com\(^5\) and meinbild.ch\(^6\) are not found by Google but account for around 2 million entries. (Ding et al. 2005)

---

\(^2\)See http://www.google.com/
\(^3\)See http://www.livejournal.com/
\(^4\)See http://www.deadjournal.com
\(^5\)See http://www.ecademy.com/
\(^6\)See http://www.meinbild.ch/
From Ding et al. (2005), making social connections is a popular usage of RDF documents with more than 1000 different ontological definitions for the term “person” being used in RDF documents according results from Swoogle\textsuperscript{7}, a semantic web search engine. In Jun 2004 the FOAF ontology was the 3rd most popular ontology on the web with more than 1,126,002 documents populated with FOAF tags, only outdone by the RDF and the RDFS ontologies which according to Ding et al. (2005) had existed in over 1,129,749 documents. The FOAF documents generated by LiveJournal showed the most popular element of the FOAF ontology was \texttt{foaf:interest} used on average 1.68 times per FOAF document as opposed to \texttt{foaf:knows} that was used on average 0.95 times in each FOAF document. Golbeck (2005\textsuperscript{a}) found there to be around 6 million FOAF files on the web with LiveJournal providing more than 90\% of them. The \texttt{foaf:knows} element is used to create the links in social networks between individuals, so with each person in the LiveJournal examination making a link to fewer than one other individuals on average it appears most FOAF documents are not well connected to others. Of the 26,936 non-blog sites FOAF documents surveyed \texttt{foaf:knows} was only used on average 0.79 times per document possibly due to the less social nature of their use, and the extra effort involved in generating meta-data by hand.

The social networking site requires a way that the matcher can navigate the network and infer opinions on web services. By publishing the information in RDF documents and using FOAF tags this already popular method can be used. If in the future more information generated from the social networking site needs to be used then the use of this format should make it easier to access.

### 7.1.2 Degrees of separation

Web service rating and recommendation is not likely to ever be done by as large a group of people as are represented by these figures but their structure could be expected to be similar. Milgram’s (1967) article *The small world problem* in which he described the small world phenomenon where he hypothesised US citizens were connected by an average chain of six people each who knew people either side of them. He showed this by looking at packages passed by hand by individuals towards a target destination in a different part of the country. It is from his experiments that the term six degrees of separation comes from. There is a website illustrating this phenomenon called the Oracle of Bacon\textsuperscript{8} hosted by the University of Virginia that shows how actors are currently on average only 2.960 degrees removed from Kevin Bacon with nodes being linked by film projects that both actors have worked on. The number of degrees from Kevin Bacon is referred to as an actor’s Bacon number. This is a reference to Hungarian mathematician Paul Erdős who having written a large number of papers in his lifetime was connected to a large number of people (504) with whom he had collaborated with on the papers. The mean

\textsuperscript{7}See http://swoogle.umbc.edu/
\textsuperscript{8}See http://oracleofbacon.org/
Erdős number, as it is known, is 4.65 according to the Erdős Number Project\textsuperscript{9} (Grossman 2002). In a paper by Kleinberg (2002) it is argued that the six degrees of separation figure may be very optimistic number for the real world and it is clearly a property that varies with the connectivity of the network.

Watts & Strogatz (1998) showed that social networks inherently have small world properties with their small world nature increasing as the random interconnection of nodes increases. This indicates that clique social structures on social network should make every node reachable in only a small number of steps. Although both Bacon and Erdős numbers use two way links and trust as it is not symmetric as is discussed in Section 7.2.2 means networks of trust will be directed graphs and therefore harder to traverse.

### 7.1.3 Dunbar’s number

Also relevant to the nature of social networks is Dunbar’s number, which is the maximum group size as determined by the neocortex size of humans. The mean group size estimated by Dunbar was 147.8, although the actual average could fall somewhere between 100 and 230. He also found evidence of this in the sizes of villages and tribes. This places a limit on the number of connections that any individual can be expected to have, although someone with around 150 connections should not be expected to remove people they no longer know from the list of people they know. Users may also trust people they do not know. (Dunbar 1993)

### 7.1.4 Generating social networks for testing

Ding et al. (2005) identified the shapes that the networks form in, starting as singleton nodes with no connections. The singletons then form a number of other links to people they know or admire at which point their shapes are referred to as stars, then as stars connect with other stars, cliques are formed which on a successful social network will continue to expand and merge with other stars and cliques. Established social networks will contain a mixture of these sub-graphs but encouraging connections between nodes and hence cliques will provide more nodes to infer opinions from.

Studies have shown that graphs with enough random cross cutting edges have small world properties. It has also been shown that social networks are graphs of the same nature also with small world properties and so it is possible to generate a random graph with a proportion of cross cutting edges to simulate a naturally evolved social network. This answers how a social network to test inference on can be generated. (Watts & Strogatz 1998, M.E.J.Newman & Watts 2001)

Onnela et al. (2006) studied the social network formed by users of a Finnish cellular phone

\textsuperscript{9}See http://www.oakland.edu/enp/
They took pairs of users who had both called each other to be said to have a relation, and measured the time and number of calls between them as the strength of their relationship. They identified three hypotheses explaining how relationship (or tie) strength between users is determined. Dyadic hypothesis claims that the strength of the relationship is only a function of the two individuals who have the relationship. The global hypothesis claims tie strength is a function of the whole network and therefore cannot be determined without knowledge of the whole network’s structure. The final hypothesis they were able to demonstrate as occurring in their sample network. The strength of weak ties hypothesis proposed by Granovetter (1973) and elaborated by Granovetter (1983) claims the strength of ties increases with the overlap of common friends. The overlap function was computed and compared against the strength of ties between users by Onnela et al. (2006) and it was found that the relationship between the two was almost linear. The strongest relationships did not necessarily have the overlap expected because a disproportionate number of the users in these relationships were found to have no other relationships other than to the one user that they had a very strong relationship with.

This structure means the social groups containing the strongest links are those that are highly localised and the links between social groups are weaker but provide the global connectivity between these groups in the network. It was found that if these weak links were removed first that at a threshold of tie strength the network would go through a phase transition, with its properties quickly changing whilst a phase transition did not occur when the strongest links were removed first. This indicates the importance of cross cutting weak links within social networks as they join together otherwise disparate clusters of nodes. Onnela et al. (2006) argues that it is the medium strength links that act as the information pathways within social networks. Where the transmission of information is dependent on the strength of the link as communities are interconnected with strong links and users with strong ties inherently know the same information from being in the same place. The first transmission of information from communities is through the weaker links as claimed by Granovetter (1973) but because these links are weak the information’s chances of survival is low, hence the perhaps shorter intermediate links are the pathways that the information is found to be using. Where tie strength is no influence on the transmission of information the flow between communities will use the weaker links as these go further. Such models are now used to simulate issues such as the spread of diseases between villages. The information transmitted could be trust or opinions.

The large mobile phone users network was partitioned for analysing by Onnela et al. (2006) using a technique known as snowballing where all the users within 11 steps from a source node were taken. This gave them a sample set of less than 2000 nodes after the surface nodes (the nodes with relationships with users outside of the sample set) were separated.

Fitting with the strength of weak ties hypothesis makes social networks scale free networks. Scale free networks are a kind of network where the degree distribution follows a power law relationship. Where $P(k)$ is the probability of a node having a degree of $k$ the relationship is as
follows:

\[ P(k) \sim k^{-\gamma} \]  \hspace{1cm} (7.1)

The network used by Onnela et al. (2006) had a \( \gamma \) of 8.4 significantly higher than the \( \gamma \) of between 2 and 3 observed in scale free networks such as airline networks. The average degree of the nodes in their network was 3.3 and the most connected individual had a degree of 144 which fits with the work of Dunbar discussed in Section 7.1.3. The Trust Project social network’s nodes had an average degree of 3.19, with a standard deviation of 4.28 degrees and an average connectance of 0.66 (1/3 of possible neighbours are not directly connected). This provides us with information on how to model the network beyond the previous simple random network model.

7.2 Trust

7.2.1 Definition of trust

Users need to state how much they agree with other users’ opinions to be able to use these other users’ opinions to infer users’ opinions on web services. How much one person thinks they agree with another person should be the trust that person has in the other. The definition of trust from the (Merriam-Webster 2003) dictionary is a:

a: assured reliance on the character, ability, strength, or truth of someone or something
b: one in which confidence is placed

The definition by Gambetta (1990) found via Abdul-Rahman & Hailes (1998, 2000) is as follows:

... trust (or, symmetrically, distrust) is a particular level of the subjective probability with which an agent will perform a particular action, both before [we] can monitor such action (or independently of his capacity of ever to be able to monitor it) and in a context in which it affects [our] own action.

In the case we are interested in this is the probability that one user will agree with the other user about their experiences with a web service. Abdul-Rahman & Hailes (2000) mentions that the definitions use of the word probability makes it inherently unsuitable for a trust metric, but Abdul-Rahman & Hailes (2000) avoids this by using the term “subjective probability” to mean the level of trust the truster has for the trustee varies between subjects. None the less, these understandings of trust, with the provisions of Abdul-Rahman & Hailes (2000) can be combined with supporting social networks to infer trust values between all connected individuals and resources.
7.2.2 Properties of trust

Transitivity

Trust does not have perfect transitivity, but there is a notion that trust can be passed between entities. (Golbeck 2005a) It is not adequate simply to say that if A trusts B and B trusts C then A will trust C. Abdul-Rahman & Hailes (1998) calls this property of trust conditional transitivity where the conditions for transitivity are as follows:

1. B must be able to indicate to A that B trusts C.
2. A trusts B to be able to make a recommendation about trust. That is to say that trust can be different values between entities for different fields and in this case A must trust B’s ability to give recommendations.
3. A can value the quality of B’s recommendation.
4. A may trust C less than B trusts C because trust is not absolute.

If M distrusts N who in turn distrusts O it could be argued using the logic of “the enemy of my enemy is my friend” that M should trust O. This transitivity would be multiplicative, as opposed to additive which would be to use the logic of “not trusting an entity who is not trusted by an entity you do not trust”. Multiplicative trust can lead to an entity not trusting themselves, although all entities should already have their own trust value for themselves that is not inferred from others. Using trust values inferred from entities that you distrust is best avoided because of the decisions involved. (Guha et al. 2004)

Composability

Using the alphabetical entities from the previous section, A’s trust of C will not depend purely on A’s trust of B and B’s trust of C if A trusts anyone else with a trust value for C. A trust value for C is composed of the opinion of entities A trusts the opinion of C on and A’s own past experience and knowledge. A question is then raised as to if B is the only user A trusts who has a value for the trust of C but some of the other users A trusts trust users who have trust values for C then should these longer chains of trust be taken into account in composing A’s trust of C? (Golbeck 2005a, Abdul-Rahman & Hailes 2000)

Personalisation

The trust that an entity A has for another entity C is personal to A. (Golbeck 2005a) It might be the case that if C is far enough removed from A that A’s trust in C corresponds to a global trust
level of $C$ but that would be merely coincidental. Trust must always be between exactly two entities. (Abdul-Rahman & Hailes 1998)

Because trust is personal, trust is asymmetric. If $A$ trusts $B$ it does not imply that $B$ trusts $A$ in the same way. (Golbeck 2005a, Abdul-Rahman & Hailes 1998)

**Subjectivity**

You might trust your friend to return a borrowed CD, but you probably would not trust them to perform a heart transplant (unless they were a cardiological surgeon). Trust values vary by subject and as such the social networking site will have to ensure that users know that the area they are assigning trust values of users on is web services. (Golbeck et al. 2003, Abdul-Rahman & Hailes 2000)

### 7.2.3 Global trust

Many networks work on the basis of assigning a global trust value to every entity. This is useful in situations such as web searching, where some or all of the users do not have a node in the network. Examples of global trust metrics include PageRank and CiteseerRank which value web pages and papers respectively by the value of web pages and papers that link to them. Levein’s Advogato measures the distance and ranking of users from seed nodes. Global values are inherently harder to manipulate because of the scale involved. (Ruderman 2004, Levein 2001, Page et al. 1998)

Trust though has the property of being personal to individual entities and seeing as in a social network users are nodes in the network trust should be inferred for each individual - locally.

### 7.2.4 Inferring Local Trust

**Trust Propagation**

To infer trust for an individual requires propagation of trust values for the individual from other members of the network. Guha et al. (2004) lists four atomic forms of trust propagation. Atomic propagation being propagation from one node to another, as opposed to propagation down a linked chain of nodes which can be built from atomic propagations.

**Direct propagation** Where trust propagates directly along an edge. If $A$ trusts $B$ and $B$ trusts $C$ then $A$ trusting $C$ would be a result of direct propagation.
Co-citation Co-citation is a backward-forward step propagation. If B trusts both C and D and A who C does not know trusts D then it might be assumed that A should have some trust for C because of the common trust of D that A has with B.

Transpose trust Earlier it was covered that A trusting B does not imply that B trusts A but A trusting B can lead to B developing some amount of trust for A according to Guha et al. (2004).

Trust coupling Trust coupling combines aspects of both co-citation and trust transposition and implies that if B and C both trust D then if A trusts B they should also trust C because B and C trust some of the same people.

The strength of these propagations from intuition appears to decrease as the list is descended. Similarly the ease at which the propagations can be used decreases as the list is descended with direct propagation working linearly. The other three propagation methods require knowing who trusts an individual as opposed to the more available information of who the individual trusts.

The propagation of trust could include the propagation of distrust and Guha et al. (2004) examined the accuracy of three angles on this with his algorithm. Trust only means the complete ignoring of distrust nodes in the network so the opinions of people that are distrusted by you or anyone you trust is ignored and only positive opinions of the individual you want to infer your trust of are taken into account, ignoring bad reviews. One-step distrust takes account of the distrust that nodes have for the target node but does not propagate trust via distrusted nodes between the source and the target. Propagated distrust takes trust and distrust to just be opposite ends of the same scale, propagating them both in the same way. One-step distrust came out best in the tests Guha et al. (2004) carried out, it does not count the opinion of people that are distrusted which seems intuitive and takes into account the distrust the people you trust have for the target node. The ratio of trust to distrust ratings found in the investigation of Guha et al. (2004) was 17 : 3. Our inferring, although it requires the inference of trust along the way, requires the inference of an opinion of a web service as the final step so will require the last step in the path to the web service to users’ opinions of a web service and it is this step which will count as the one step that can include distrust (or more appropriately just a negative opinion).

Path length

Guha et al. (2004) also tested whether results were more accurate when path length was taken into consideration. By weighting trust values using a $\gamma^k$ where $k$ is the number of steps taken inferring that trust value and $0 < \gamma \leq 1$ when combining trust values from multiple neighbours opinions of members of the network closest to you are valued higher than those at a great distance. Guha et al. (2004) found that the accuracy of trust inference was better when path length was penalised ($\gamma = 0.5$) and that accuracy decreased as the maximum allowable $k$. 
Trust inference by Guha et al. (2004)

The Guha et al. (2004) implementation was tested on Epinions.com\(^\text{10}\), a site that uses a trust network to prevent the abuse of reviews of a wide range of products. The algorithm they describe uses amongst others the principles described above. The trust values between users in the social network are stored in a matrix and the propagations are defined as operations to transform the matrix to infer trust values. The propagations are weighted for combination and results are rounded to discrete values for presentation to the user. The algorithm is shown on the Epinions userbase to achieve prediction errors of 6.4% but is not a method that can be quickly executed because of its use of large matrices. The paper mentions that matrix - matrix operations are not necessary and instead a “Lanczos-style” matrix vector operation can be used at each step instead for the same result to reduce some of the complexity.

TidalTrust

TidalTrust infers trust using the trust values assigned by those on the shortest path(s) between the source (the node trust is being inferred for) and the sink (the node trust is being inferred of). This is done based on the assumption that accuracy of inferences is increased as path length is shortened agreeing with Guha et al. (2004) referred to previously. The weighted average of the trust values from the shortest paths is then the inferred trust value. The algorithm runs in linear time dependant on the size of the network. TidalTrust has been shown to produce significantly better results than both simple averaging and Beth-Borcherding-Klein inference algorithms.(Golbeck 2005\(^a,b\), Beth et al. 1994)

Richardson et al’s trust metric

The algorithm described in Richardson et al. (2003) is a probabilistic method of inferring trust. It treats trust values as probabilities of following that link to a node. A’s trust of B is the sum of the likelihoods that a random network “spider” walking along trust connections in the network would make it from each node to B where \(0 < \lambda_i \leq 1\) is the users self-belief value and is the chance that the “spider” would leap back to A. The more self-belief the more the result is influenced by the source’s trust values and not the overall trust value that would exist for the sink. This technique of inferring trust is inspired by the PageRank algorithm, but provides local values of trust and when tested on the Epinions.com network had a precision of 87% ± 13%.(Richardson et al. 2003, Page et al. 1998)

\(^{10}\)See http://www.epinions.com/
7.2.5 FOAF trust schema

Golbeck et al. (2003) describes an extension of the FOAF schema that was developed to allow individuals to explicitly declare a level of trust for other users they know. Rating from “Distrusts absolutely” to “Trusts absolutely” on a ten point scale. The range of users’ trust can be described as continuous between those two extremes but Golbeck et al. (2003) chooses a discrete ten point scale on the basis that it makes it easier for users to work with. The schema also supports users providing trust ratings in specific fields, so subjective trust ratings, as discussed in Section 7.2.2, can be provided enhancing the information available for inferring ratings and trust. The subjective part of the scheme does not appear to be implemented in the social networks created by Golbeck but given the subjectivity of trust it would appear to be worthwhile using.

7.3 Ratings and social networks

Inferring ratings for products from the opinions of people you trust in your social network can mean more appropriate, personal ratings. In this section the rating of products by users is explored.

7.3.1 Explicit and implicit ratings

Ratings can be gathered either explicitly or implicitly (or both ways at once). The explicit gathering of ratings has the cost of users having to provide the information. This can mean a lack of ratings provided because of the effort required by the users but it does mean that the ratings should be accurate for each user that records them within the limits of whatever discrete scale(s) are used. Because ratings are desired by many systems, users are often compensated for the effort taken in creating them increasing the volume of ratings. Gathering ratings implicitly can provide different information because it asks different questions. It removes the cost to the user but has limited accuracy in areas such as product satisfaction because the user themselves is normally in the best position to evaluate what that is for them. In the field of rating web services explicit ratings are necessary to find the opinions of users because it cannot be implied or observed from users’ actions and the user does not interact with the services through the system so implying anything about the user would be intrusive. (Nichols 1998, Abdul-Rahman & Hailes 1998)

7.3.2 Potential Rating Inference Problems


Credibility and attack-resistance  This is a problem that comes about from a lack of account-
ability of users in a system. False profiles that promote or demote products or individuals are difficult to prevent. Using trust networks can be used to help avoid the abuse, because false profiles should not be trusted by individuals so will not be part of a network of trust in the case of TidalTrust and others. In the Guha et al. (2004) algorithm trust can be propagated from users who trust the same user making it more susceptible to false profiles because the false profiles can trust the same users as a genuine user and the Guha et al. (2004) algorithm will cause the trust inferred by the genuine user to be effected by the false user profile.

**Product-user matrix sparseness** This is a problem that occurs because there are not enough ratings in a network for a particular entity making it hard to infer user specific ratings for them.

**Computational complexity and scalability** Inferring ratings for products and users becomes increasingly harder has the number of users and products increases for many algorithms such as Richardson et al’s trust metric.(Beame et al. 1990)

A fourth problem of user error was discussed in Golbeck (2005a). The problem is associated with asking the users for explicit ratings for products. If users misinterpret the scale, perhaps thinking a rating of 0 is a bad value when in reality it corresponds to an average value. Users may also not rate another user or a product honestly if the information is publicly available, feeling socially obliged to give a particular answer biasing the accuracy of anything inferred from it. The only solution to this is to ensure the scales are clear and easy to understand and that honesty is encouraged in ratings of users and products.

### 7.3.3 FilmTrust ratings

The FilmTrust social network and film rating site described in Golbeck (2006) describes a website where users build a network of trust that can be defined in the FOAF trust schema mentioned in Section 7.2.5 and can post reviews and ratings of films. The TidalTrust algorithm (see Section 7.2.4) is used to infer trust used to weight ratings given by users to films to produce an accurate average for each film for each user based on the trust network. This works on the principle that users should have a similar taste in films to people whose opinions of films they trust and people prefer to receive recommendations from the people that they both know and trust.(Sinha & Swearingen 2001) The system also allows users to allow members of the network they do not know to be their trusted neighbours if the user feels they share a similar taste in films. It was found that users would be more likely to trust users with similar extreme views on films and that sharing common opinions on many films did not mean users would trust other users unless they agreed with each other with their extreme views.(Golbeck 2006, Ziegler & Lausen 2004)

The FilmTrust site with the provision of subjective trust is a good match for what needs to be
developed here. Similarly the TidalTrust algorithm also seems an appropriate algorithm to at least start with.
Chapter 8

Design and implementation

As discussed in the previous two chapters three products need to be developed to allow this matchmaking by user opinions and that of their friends. A social networking site within which users can establish relationships and assign trust values to other users as well as express their opinions on web services. There also needs to be an application to provide the social networking site with the list of web services to be reviewed by the users of the social networking site. Then, of course, there needs to be the matcher which finds or infers the opinions of users on web services as it is asked. This chapter will cover the design and implementation of these products.

8.1 WSTRN

WSTRN (the Web Services Trust and Reputation Network) is a social networking website developed for this project and deployed at http://people.bath.ac.uk/cs3nb/WSTRN presently. The site was coded in PHP\(^1\) and used the Pear DB\(^2\) to interface with an SQL database behind the website. The site was written in PHP 4 because of its wide support on the available hosts. The aim of the site was to provide an area for users to find other users with similar opinions and to show their opinions on web services by scoring and reviewing them. The source code for the website is provided on the attached CD.

Due to budgetary, personnel and time constraints the scale, design and scope of the website has not been aimed to compete with successful social networking sites such as Facebook\(^3\), MySpace\(^4\) and LinkedIn\(^5\). These sites feature more general approaches to social networking. The use

---

\(^1\)See http://www.php.net

\(^2\)See http://pear.php.net

\(^3\)See http://www.facebook.com

\(^4\)See http://www.myspace.com

\(^5\)See http://www.linkedin.com
of existing social networks was researched but no site seemed to provide accessibility to the information required for the inferring of opinions or the ability to assign trust values to users and rate web services. The WSTRN website was therefore designed to provide these features with a focus on web services.

Once a user is logged in to the site they are presented with a page called the Home page that provides them with an overview of the latest reported news, the latest reviews they have posted, a selection of their friends details and lists of the latest reviews and the top rated web services. There is also two search boxes for searching for web services and friends that are present on every page and along the top of the page there are six links to pages focusing on specific features of the site. These links are to pages on news, friends and other users, reviews, the user’s profile, help and for logging out of the website. These pages are covered in more detail in the subsequent sections.

### 8.1.1 User information

Users registering for the website need to provide a unique user name, email address and password with which they can log on to the website. A name and optionally a nickname which is used if available to label the user on the website is also asked for. The site as yet does not attempt to verify the user’s email address is their own, but checks on the format of the given email address are used. It is debatable whether it is better for the user to have to provide their real name because of the popularity of pseudonyms on the web, but by having a name field the option is there for users to provide their name if they want or alternatively think of a fake name to fill the box with.

By clicking on the link labelled Profile at the top of every page on the site, or clicking on their own name where it is used on the site, users can edit the information the site stores and publishes on them. The information users can provide here is inspired by the FOAF vocabulary specification as that is how the information collected here will be published. With the exception of the users emails accounts which although they will be visible to users the user acknowledges as their friend are not published to others because of the mail the user could end up receiving because of it. Instead the SHA1 sum of the users primary email address is published as per the FOAF specification because this allows web users who already know the email address of the user to confirm it is them by computing the SHA1 sum without revealing the user’s email address.

The details users can edit on this page are as follows:

**Basics** This section allows the user to edit their name and their nickname which will appear in the users FOAF file. Here also the user can input their birth date but this is not published in their FOAF file despite the specification allowing it because the property is labelled as unstable in the specification because of security concerns referred to as the Birthday Issue.

---

6See [http://xmlns.com/foaf/0.1/](http://xmlns.com/foaf/0.1/)
7See [http://rdfweb.org/topic/BirthdayIssue](http://rdfweb.org/topic/BirthdayIssue)
CHAPTER 8. DESIGN AND IMPLEMENTATION

Password This obviously private information can be edited on this page.

Email Accounts Users must have one email address associated with their account but can add more and with each email address is associated a use that the email address has, such as “Business” or “Cycling”. None of these email addresses are published to anyone other than the users the user acknowledges as the users friends.

Chat IDs FOAF allows users to publish their account names for the five main online chat networks.

Work FOAF allows users to specify a resource that defines their job and another that defines their workplace. The WSTRN website supplements this by providing the option to fill in the job title and the name of the users workplace making it quicker to understand for users’ friends. This could, in the future, allow users to if they wanted to network easier with colleagues from work or with other users working in the same job type.

Schools FOAF allows the users to specify schools they attended by their URL and the WSTRN website allows users to add to this by providing the school’s name and the year that they were in the class of at school. This could, in the future, allow users to if they wanted to network easier with friends from their school.

Sites Allows users to list websites that they are responsible for, each along with a title to make them better to present to the users friends.

Weblogs Allows users to list the weblogs that they maintain online each also having a title associated with them.

FOAF Files Here the URL of the FOAF file is provided for the users to use as part of their query for the matcher. Users can add addresses of other FOAF files they have and these will be linked from the users FOAF file using the RDF Schema seeAlso property.

Interests Here users can list the titles and URLs of their interests which will be added to their FOAF files. This information could allow users to be grouped by common interests by either their interests’ URLs or the titles of them.

Publications Here the users can submit an URL of a page containing publications that they have.

Plan The plan field is intended to replicate the use of the .plan files on UNIX and UNIX-like operating systems. As a large piece of text users can use it for instance to publish bibliographical information. This information is also stored in users’ FOAF files.

Picture By default users are represented with a simple blue silhouettes along with their names but they can submit an URL of an image that represents them and this URL is also included in their FOAF file using both the FOAF depiction and the img properties. This
presents an area where the site is susceptible to abuse and this may be an area of the site that needs more improvements as at present only the dimensions of the file are checked. Manual checking may need to be added.

In Figure 8.1 is an example FOAF file generated using the fields in the user’s profile. It shows the support for the `holdsAccount` property that indicates within the FOAF file that the user has an account at the parent site.

Part of the reasoning behind supporting many aspects of the FOAF vocabulary other than to create interesting points for the users to share and have in common was that by supporting the vocabulary strongly it increases the number of files published in this open standard. The greater the use of FOAF files on the internet the more information there is for the matcher algorithms and others to use for semantic inference.

### 8.1.2 Connecting users

Being a social networking website a key aspect of the site is to connect users. Here we want users to assert trust values on other users wherever they feel they can say how much they will agree with other users, although more particularly where they do trust the other users highly. The more high trust values a user assigns the more links there are to use in inferring that user’s opinions and in general the more common that user’s opinion should be. Users are likely to want to establish links with other users that they already know, perhaps the users that recommended them to the site for instance so this must be supported. Users may also discover after joining the site, where they can see the reviews of users, some of these users have similar opinions on web services to what they have so these purely through the site encounters must also be facilitated.

The friend search box on the top left of each page on the site provides users with a quick way of finding users they already know by their name or nickname. The Friends link at the top of every page provides a focus for finding users on the site by name, nickname, email, user name or for just browsing the friends they have already added. This page also displays the requests from other users the user has received.

When viewing a user’s profile who is not already a user’s friend the user is given the option to add that user as a friend. Doing this informs the other user of this action and exposes more information to the added user on the adding user. If the friendship is reciprocated the user can see the profile details of the added user but either way the user can add a trust value about the user. The database behind the website is designed in such a way that if in the future other subjects users needed to express trust in had to be added they easily could be. At present the only subject users can assign trust to other users on is “Web Services” and it is this trust type that is to be looked for and used by the matcher when inferring opinions.

Whether or not to require users to assign trust values to users they had not added as friends was
<?xml version="1.0"?>
<rdf:RDF
    xmlns:foaf="http://xmlns.com/foaf/0.1/
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:review="http://www.purl.org/stuff/rev#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:trust="http://trust.mindswap.org/ont/trust.owl#">
    <foaf:Person rdf:ID="nick">
        <foaf:mbox_sha1sum>48b7fc310d444eac57f0eb9373f7427b36baa0dc</foaf:mbox_sha1sum>
        <foaf:name>Nicholas Brunwin</foaf:name>
        <foaf:nick>Nick</foaf:nick>
        <foaf:plan>If only I could think of something to write here.
                At least the option’s there.</foaf:plan>
        <foaf:msnChatID>nicnoc_@hotmail.com</foaf:msnChatID>
        <foaf:jabberChatID>brunwin@gmail.com</foaf:jabberChatID>
        <foaf:publications rdf:resource="http://www.bath.ac.uk/~cs3nb/publications/"></foaf:publications>
        <foaf:holdsAccount>
            <foaf:OnlineAccount>
                <foaf:accountName>nick</foaf:accountName>
                <foaf:accountServiceHomepage rdf:resource="http://www.bath.ac.uk/~cs3nb/WSTRN/"></foaf:accountServiceHomepage>
            </foaf:OnlineAccount>
        </foaf:holdsAccount>
        <foaf:img rdf:resource="http://profile.ak.facebook.com/v52/1228/116/n204506496_7654.jpg"/>
        <foaf:homepage rdf:resource="http://www.bath.ac.uk/~cs3nb/"
        <foaf:weblog rdf:resource="http://www.bath.ac.uk/~cs3nb/blog/"
            <rdfs:seeAlso rdf:resource="http://www.foaf.com/foaf"/>
        <foaf:interest>
            <rdf:Description rdf:about="http://www.fishing.com">
                <dct:title>Fly fishing</dct:title>
            </rdf:Description>
        </foaf:interest>
        <foaf:schoolHomepage rdf:resource="http://www.bath.ac.uk/"
        <foaf:workplaceHomepage rdf:resource="http://www.kentcollege.com/"
            </foaf:Person>
    </rdf:RDF>

Figure 8.1: User’s information within a WSTRN FOAF file
a question faced in the development of the site. Allowing trust values to be assigned to anyone may encourage more trust values to be assigned because the user being assigned the trust value does not have to know about being added as a friend of the user for this to take place. On the other hand by requiring the adding of the other user as a friend before they can be assigned a trust value, means that if a user adds another user with similar tastes in web services as a friend to able to assign them a trust value, this other user will be alerted to the presence of the adding user and be more likely to reciprocate the relationship. Requiring users to add users as “friends” before they can assign trust values to them should by increasing the cost of adding users reduce the number of low trust values assigned. Because although the assignment of trust values is important to the connectivity of the trust network the Tidal Trust algorithm used on the network as explained in Section 8.3.2 will operate better where higher trust values are prevalent. The other reasoning behind the requiring of the adding of users as friends to assign trust values is that the site is forming a social network as well as a trust network and the adding of social, mutual links should be encouraged.

When users add other users as friends and this is reciprocated the relationship is acknowledged in their FOAF file using the FOAF knows property. That the users actually are friends is not required by the FOAF specification only that it is expected that the relationship is to be reciprocated. Topical trust properties are added to a user’s FOAF file when a user states they trust another user (more than nought out of ten on the scale). The trust scale used on the website and within the RDF FOAF document is the ten point scale discussed in Section 7.2.5. It is this property that is to be used by the matcher. Figure 8.2 shows the property of one user’s FOAF person that shows that they know Bob and trust them 8/10.

News

In an effort to make the WSTRN site less about just rating web services and your friends a news page was added to allow an extra channel of communication for users. Users are notified with news of users inviting them to add them as friends and this notification would have been required one way or another anyway. Users are also notified if their friends update their profile, publish reviews or have a birthday and if they publish news stories. Users can post news stories that their friends can read and pass on themselves. This provides a simple announcement and messaging system and as users can see all the posters of a news item they can use this as another way to discover similar users.

8.1.3 Reviewing web services

The opinion inferring matchers working over the data generated from social networking website would be expected to use the numeric score given web services to infer users opinions of them. But only letting users pick numbers to review web services leaves the reason behind their scoring
Figure 8.2: User’s relations in WSTRN FOAF files
unknown. So instead reviews consist of three parts - the given score, a text review of the web service, and a brief one line summary of the review to head it with.

Users are allowed through the site to score web services by using the eleven point scale of nought to ten. This scale was chosen as it matches that of the trust ratings of users (with the addition of nought to the scale), is compatible with the review vocabulary used in the generated RDF files and importantly is a simple scale for users to use.

Users can only have one review for each web service at any one time, but they can retract reviews they have put their name to. Because users share opinions (that being a basis for this project) and because writing reviews is not what every user wants to spend much time doing users can add their names to reviews that they agree with, instead of writing new reviews themselves. It would be hoped this would show the users agree with both the score and the text of the review but user testing would be the only way to know for sure.

Reviews are added to the RDF files containing users’ FOAF and relationship information by including FOAF made properties in the users’ FOAF data which point to the review at the bottom of the document. The review in turn refers back to the reviewer and includes what the review is about, the score, text and time of the review and a link to the web service’s own RDF document that the site provides as a central location of all the web services reviews, labels (as discussed in Section 8.1.4) and links to the RDF documents generated by the site on the operations of each web service. An example of a review in a user’s FOAF document is shown in Figure 8.3 where the user “nick” has reviewed the Publish web service and given it a rating of nine out of a possible ten.

8.1.4 Labelling web services

Both web services and their operations can be labelled by users of the site. Labels provide a mechanism for sorting web services and along with searching by their names provide a way of users searching for web services. Labels are published using the tag RDF ontology within both web services’ RDF documents and within web services’ operation’s RDF documents. The labels improve the searching on the site through providing a mechanism for a folksonomy to develop. It was hoped that in the future simple matchers could be developed to utilise this information and that possibly if the labels were encouraged to be actual terms that the labels could be used by a matcher such as that in Part II. The labels are not restricted, as is often the case with similar tagging systems on websites, to single words. The labels can be added to individual operations or a general label can be added to the whole web service (which effectively adds a label to each operation), the number of each kind of label used on web services can be seen in their label clouds which display the labels using larger for the more common labels. Clicking on labels on

---

8 See http://www.purl.org/stuff/rev
9 See http://www.holygoat.co.uk/projects/tags/
<rdf:RDF>
  <foaf:Person rdf:ID="nick">
    ...
    <foaf:made rdf:resource="#wsreview-19-nick"/>
  </foaf:Person>
  <rdf:Description rdf:about="http://alis.cs.bath.ac.uk:18080/grimoires/services/publish?wsdl">
    <dc:title>Publish</dc:title>
    <review:hasReview>
      <review:Review rdf:ID="wsreview-19-nick">
        <dc:date>2007-03-30</dc:date>
        <review:reviewer rdf:resource="#nick"/>
        <review:rating>9</review:rating>
        <review:text>I do not know what I was doing using this.</review:text>
        <rdfs:seeAlso rdf:resource="http://www.bath.ac.uk/~cs3nb/WSTRN/wsrdf.php?ws=22"/>
      </review:Review>
    </review:hasReview>
  </rdf:Description>
</rdf:RDF>

Figure 8.3: User’s reviews in WSTRN FOAF files
the site takes the user to a page where all the web services with the given label are listed.

### 8.2 Grimoires Synchroniser

The Grimoires Synchroniser program is a simple Java application that uses Apache Axis and the Grimoires client libraries to synchronise information between the WSTRN website and a set of Grimoires repositories. The source code for the application is provided on the accompanying CD. The application removes what repositories and the web services associated with them that it is instructed to. Then each queries Grimoires repository it is given in turn obtaining a list of WSDL URLs of the web services stored within the repository. The WSDL files at these URLs are then parsed and translated into web service information for the site. Then the copies of these web services already in the WSTRN database are labelled as such so they can be reinserted into the database and continue to have the same labels and reviews after the database has been emptied of web services and their operations. Using the information on the web services obtained after the adding of the web service operation information into the database, the addresses of the RDF files associated with the operations are added into the relevant Grimoires repository. This allows information generated via the WSTRN site to be easily accessed via Grimoires. So a future matcher could access the labels generated on WSTRN via Grimoires to generate a score for each web service.

### 8.3 Trust Network Matchmakers

A trust network scorer (matcher) that inferred users opinions from the information generated by the social network was implemented using the TidalTrust algorithm from Golbeck’s (2005a) thesis, referred to previously in Section 7.2.4. After some testing a more optimized version of the algorithm was developed to avoid being delayed too greatly by the need to access the information across the internet. Two other methods for the matcher were also developed but as an exploration of ideas, as real-life trust networks to test the matchers on and users to check the results against are not available in the timescale available. It has therefore been presumed from the results shown that the TidalTrust algorithm is the most accurate for the domain of calculating individual opinions of resources using explicit subjective trust ratings and explicit scoring of resources (in this case web services). The methods are approached here starting with the simplest.(Golbeck 2005a,b, 2006)
8.3.1 Common features

All the trust network scorers created for this project have certain common features. For the same reasons as explained for the semantic text matcher covered in Chapter 4 the matchers were implemented in Java. To query the RDF documents that the matchers traverse the ARQ library from the Jena framework\(^\text{10}\) is used. The source code of matchers is included in Section A.2 and on the accompanying CD.

Each trust network scorer takes as an argument the address of the user’s FOAF information (including the identifier of the FOAF person entity within the document) that they are to infer the opinion for and the URL identifying the resource that they wish to infer the score for (the address of web services WSDL files in the case we are interested in). When instantiating the matcher classes the subject over which to infer the trust of the given resource is specified (which is in the case utilised for this project ‘Web Services’). This allows the matchers to easily be deployed to infer opinions of resources in other subject areas, so long as the users fit the pattern of making their reviews a property of the resource, as shown in Section 8.1.3.

The trust network scorers use the links made by users FOAF documents saying other users are trustedPersons on the given trustSubject and ignore the social links made by the FOAF knows of users FOAF documents. They build their models of each user by taking the FOAF person referred to by the URLs they receive and expanding the document that contains the user’s details by following RDF Schema seeAlso properties recursively until there are no more to follow (taking care not to loop) and the model is complete so as not miss any details.

The matchers all trust the user to value their own reviews where they exist as the only source of information that they care to use on a subject. So inference does not take place where the source user has a review for the web service in their FOAF document.

In the event that the matchers are used on networks that are too vast to traverse in a reasonable length of time the matchers are all implemented so that a maximum distance to be travelled from the source can be set. On the networks that the algorithms are tested with this limit does not come into use. But if a network with tens of thousands of users or more deployed such a matcher to infer opinions on resources then a limit would prevent queries taking inordinate long at only the loss of the opinions of distant users.

8.3.2 TidalTrust

The TidalTrust algorithm uses the idea of limiting the flow of trust to the source to the maximum path flow along the shortest paths between the source node (the user to infer the opinion of) and the sink node (the user reviewing the item concerned). Golbeck (2005a) showed evidence of the accuracy of inferred values decreasing with the further the sink was from the node used and that

\(^{10}\)See http://jena.sourceforge.net
for set path lengths, paths with higher trust ratings provide more accurate results. The algorithm tries to take advantage of this information without compromising the amount of inference that can be carried out by not outright blocking the lowest values of nodes.

The view of trust taken by the TidalTrust algorithm is as a weighting given to opinion and therefore an opinion inferred only from the opinion of a single user, so long as some trust in them exists, will result in the same result regardless of how much trust in the opinion of that result there is, because it is the only opinion on the subject available. Opinions here being opinions of trust in other users. TidalTrust can be described as one dimensional as it only produces a single inferred trust value for users an does not track the uncertainty introduced in the inferred values. As the matchers for the inference of opinions of web services only needs to infer the opinion as a single score, information such as uncertainty information is not of use as part of the final opinion (score) given.(Ries 2007)

In the original TidalTrust algorithm the trust in a given user (the sink) would be inferred for the user to infer the opinion for (the source) with one pass of the algorithm. Here we have implemented three variations on the TidalTrust algorithm, where the relationship between trust and opinion of web services and what the sink is defined to be varies.

### 8.3.3 Single Depth TidalTrust

The single depth implementation of the TidalTrust algorithm takes the sink to be the web service itself and not the reviewer of it with the review scores viewed just as trust values. This means the review scores given by the user are effectively taken as the trust in the user of the web service. This also avoids having to search beyond what might be a sink (a reviewer of the web service), to find out if a node actually is a sink, by specifying exactly what the sink is before the search is begun - it is the web service not a user which has a review of the web service. This also allows multiple reviews to be taken into account in a single run of the algorithm. Otherwise the algorithm has to be adapted to allow it to infer the trust of multiple sink nodes in a single pass.

The result of the Single Depth TidalTrust algorithm is the weighted average (according to inferred trust) of reviews of a web service at the closest distance to the source that reviews of the web service can be found at. To select the paths from the source to the sink to infer the trust the source has of the sink the algorithm searches for the sink amongst its neighbours (the users it trusts), if the sink is not found the search continues looking at the neighbours of the source’s neighbours (that are not neighbours of the source themselves) and so on. With no deliberate order of search taking place at each depth, as it does not change the result or reach it any quicker by taking a particular order here. When the sink is found, the depth at which it was found is fixed as the maximum depth at which the sink is to be searched for and the remaining nodes at that depth are searched for the sink. Once this is done the trust threshold is calculated. The trust threshold is the maximum value of the path flow of the trust paths leading to the sink from the source. Where
the path flow is determined to be the maximum minimum trust value between any two adjacent
dnodes in the paths between the source and the node. This limits on the amount of trust that can 
flow from the source to the nodes neighbouring the sink. All the nodes neighbouring the sink 
that have a trust path to them above the trust threshold are then used to calculate the trust from 
the source to the sink.

The TidalTrust algorithm is explained in detail in Golbeck (2005a) but a description of the Single 
Depth TidalTrust algorithm as it is implemented will be given here. To a list of nodes to be 
searched is added the source node. Whilst there are still nodes to search, for each of the nodes 
in this list the sink (a review of the web service) is looked for. If a review of the web service is 
found the sink is set as the child of the node and the review score given by the node to the sink is 
stored as its cached rating. Additionally the path flow to the sink is set as the maximum of any 
previous path flow assigned to the sink, and the minimum of the path flow of the node and the 
score of the review of it. If a review of the sink is found the maximum depth is set as the current 
depth and no new nodes are added to the lists of nodes to be searched. If no review for the web 
service can be found the neighbours of the node, that have not already been added to the list of 
nodes to search, are added to a list of nodes to search after the current set of nodes to search are 
finished with. But not until after first setting these neighbours path flows to be the maximum of 
the any previous path flow assigned to the neighbour and the minimum of the path flow of the 
ode and the score of the trust value of the neighbour. Then setting the neighbours as the children 
of the node. Once the list of nodes to search is exhausted the next list of nodes is iterated through 
unless the maximum depth has been exceeded.

Now if the sink has not been found 0.0 is returned as no opinion of the web service can be inferred 
and it can be considered not endorsed by any user connected to the source user. If the sink has 
been found its path flow is set as the trust threshold. The final score can then be calculated using 
the following set of loops shown in the pseudo code in Algorithm 1. Essentially for each node’s 
set of children the child’s rating is added to the node’s cached rating with a weighting only if the 
child’s rating is above the trust threshold and if the child has a cached rating of the sink. The 
nodes are traversed towards the source calculating their cached ratings, eventually calculating 
the cached rating of the source which is the opinion of the source node of the sink. Equivocally, 
along each trust path with a path flow equal to or greater than the trust threshold a weighted 
average of the opinions of the web service is passed towards the source. The cached ratings of 
the other nodes are not technically what their opinions would be inferred as because they do not 
take into account any neighbours of those nodes that are nearer or just as near the source because 
nodes are not visited twice in search of the sink. The cached ratings are instead perhaps what 
the opinions of the other nodes would be if the nodes just as near to the source or nearer did not 
exist. The global and strength of weak ties hypotheses discussed in Section 7.1.4 indicate that 
removal of some of a node’s neighbours may influence their trust in the remaining nodes making 
the meaning of the cached ratings less likely to be described as such.
Algorithm 1 Calculating opinions once the paths to the sink have been found

1: \textbf{for} $d = \text{depth}(\text{sink}) - 1$ to 0 \textbf{do}
2: \hspace{1em} \textbf{for all} node in searchedNodes($depth$) \textbf{do}
3: \hspace{2em} numerator $\leftarrow$ 0
4: \hspace{2em} denominator $\leftarrow$ 0
5: \hspace{2em} \textbf{for all} child in children of node \textbf{do}
6: \hspace{3em} rating $\leftarrow$ node’s rating of child
7: \hspace{3em} cachedRating $\leftarrow$ child’s cached rating of sink
8: \hspace{3em} \textbf{if} rating $\geq$ trustThreshold \textbf{and} cachedRating $\geq$ 0 \textbf{then}
9: \hspace{4em} numerator $\leftarrow$ numerator + (rating $\times$ cachedRating)
10: \hspace{4em} denominator $\leftarrow$ denominator + rating
11: \hspace{3em} \textbf{end if}
12: \hspace{2em} \textbf{end for}
13: \hspace{1em} \textbf{if} denominator $>$ 0 \textbf{then}
14: \hspace{2em} node’s cached rating of sink $\leftarrow$ numerator/denominator
15: \hspace{1em} \textbf{end if}
16: \hspace{1em} \textbf{end for}
17: \hspace{1em} \textbf{return} source’s cached rating of sink

Optimization via caching

Our implementations of the TidalTrust algorithm was optimised to speed up the average inference time. Although each run of the algorithm needs only to access each user that it encounters’ FOAF documents once when a KNOOGLE query uses the matcher the matcher will be traversing the nodes in the sources neighbourhood for each web service specified by the user’s query. So to avoid the delays introduced by accessing pages over the internet to get the user’s FOAF information, the time it takes to turn the pages into models to be queried and then querying the models to get the required information from them an implementation was created in which the final results of those steps are cached for a given length of time. Because different stages of the algorithm require different parts of the model the caches only contained the parts that had been required of them so far. When reviews are searched for, if they have not already been stored in the cache, the model for the node is built and all its reviews (which would each have to be searched anyway) are stored within the node’s cache. If the desired review is not found and the node’s neighbours are to be iterated through, if they have not already been stored in the cache, the model of the node’s FOAF file would have been cached from previously searching for reviews so is used to find the node’s neighbours after which the model can be disposed of because there is no further need for it. The node’s cache does not actually cache the cached rating discussed previously because nodes having cached ratings depends on who the opinion is being inferred for. For instance if A trusts both B and C and B has a review of a given web service that C has a
cached rating of but no review of, then A should only want B’s cached rating of the web service because C’s cached rating comes from a distance further away. Because B and C are both one step away from A but B is closer to a review.

It is conceivable that the length of time cached results should be kept for should be around 30 seconds allowing plenty of time for the advantages of the caching to be utilised by queries to the matcher but not so long that changes to the trust network are ignored for very long. The worst case time of the cached version of the TidalTrust matcher should be marginally worse than the original TidalTrust matcher because of the small amount of time required for the creating and retrieving of the caches, but after a second query (within the next 30 seconds) this time difference should be more than negated.

Two other caching versions of the single depth algorithm were implemented purely for the purposes of testing. A version which cached text of the pages was created to remove the delay added by network access times. How big a saving that would be depends on the connection between the matcher and the FOAF files. Another version was created that cached the model to discover the delay added by construction of the model of the document, although this will vary as the size of the document varies. These delays are tested in Section 9.2. It could be said the opinion of any node of any web service could be cached briefly. But this same caching of the result of query and web service could be done for any matcher where no random element was added.

8.3.4 Multiple Depth TidalTrust

The Single Depth TidalTrust algorithm computed just the results of finding reviews of a resource (in our case a web service) at the closest distance from the source at which the resource could be found. The Multiple Depth TidalTrust algorithm we have implemented separates the trust ratings from the reviews of resources and defines sinks, of which there can be more than one, to be the reviewers of the given web service. How the scores given by the reviewers found is combined with the trust inferred of the reviewers is handled as a separate issue and two methods of combination are discussed in Section 8.3.4.

The Multiple Depth TidalTrust algorithm does not stop searching beyond the depth at which a sink is first found at. It finds all the reachable reviewers within the trust network (taking into account the maximum depth it is told it can work to) and then proceeds to infer the trust the source user should have in them. It manages this without looking ahead because the algorithm stores the cached rating of each users’ trusted neighbours that are not as closer or closer to the source as the user in the network in case one of their neighbours is a sink. Sinks, reviewers of the given web service, are only detectable once a model for their FOAF document has been built. But as every reachable user has to be searched to see if they have anymore users connected from them and in case they are sinks there is no loss incurred from not looking ahead to check if users friends are sinks. If a user is found to be a sink it is stored in a list of sinks along with the depth
at which it was found and the score it gave the given web service and the search for more sinks continues.

Once the reachable network is searched the trust of each of the sinks can be inferred using the depth values associated with them and the cached ratings of each of the sinks that the users one step closer to the source user contain. This is done using the algorithm shown in Algorithm 1. There is an argument for altering the algorithm so that all the trust values for the sinks are computed in one pass from the furthest part of the network to the closest. But then checks have to be performed to know when to accept the additional effort of searching for each sink. There is little cost incurred in traversing parts of the network more than once compared to the cost of trying to avoid unnecessarily searching for sinks at depths beyond what they should be found at which being a search operation is expensive.

**Combining trust of reviewers with review scores**

The Multiple Depth TidalTrust algorithm results in a set of pairs of trust values of reviewers and the scores that each of them gave to the given web service. The matcher is specified as needing to return a value between 0 and 1 so ways of mapping the sets of pairs to a single value are required. The two mappings correspond to two of the three methods, the other being the Single Depth TidalTrust method of opinion inference.

The first of the two mappings is a simple weighted average of the scores. This results in the same results as implemented by the FilmTrust website from Golbeck (2006). The second of the two mappings takes the trust value of the reviewer to be the trust that the review they give will be better than the expectations of the user. The expectations are not inferred here as they might have been from the average review score the source user gives a web service because not all users will have reviewed web services and the scores users give the web services they review do not reflect necessarily what they expect a typical web service to be like. Instead the expectation is set as 5, in the middle of the range of the possible scores that resources could be given by reviewers. This is configurable so if it was felt the user should be more or less positive about web services this could be recognised. Where the trust values must fall between 0 and 1 and \( n \) is the number of pairs of scores and trust values provided the Formula 8.1 is used, the result of which is divided by 10 to bring between 0 and 1.

\[
res(n) = \begin{cases} 
\frac{\sum_{i=1}^{n} (score_i \times trust_i) + (expected \times (1 - trust_i))}{n} & \text{if } n > 0 \\
0 & \text{otherwise} 
\end{cases} \tag{8.1}
\]

The second mapping should produce results that are closer to 0.5 and with the more influence from the most trusted users than in the second mapping. The distribution of scores is explored in Section 9.4.
Chapter 9

Testing and discussion

To test the matcher and site a network of trust and reviews has to be generated so in Section 9.1 we describe the network we generated and its properties. Then using this network we tested the effects of caching results on the algorithms and after seeing the benefits of caching how the run-times of the algorithms vary using it. Then we observe the reasons behind the distributions of the algorithms scores from inferring opinions on the network. Then using our experience from our investigation we pinpoint requirements for the success of the matcher and areas in which its performance can be improved.

9.1 Trust network generation

To be able to test the trust based scorers, given the time available resulted in a lack of real world data, a system was required to generate a trust network with which to test the networks on. Not being able to know how many reviews will be posted for each web service means that the algorithms will have to be tested with a number of “densities” of reviews. To generate the networks a program called Pajek\(^1\) was used to generate undirected graphs from which to build the test social network. Pajek is a free for non-commercial use program for the analysis of large networks but although it could generate weighted directed graphs it does not allow the control required to construct graphs with weightings resembling what we expect of trust networks.

A trust network generator application was written that provides a text based interface for loading and manipulating social network graphs, the source code for which is available on the attached CD along with the test data. The application allows the loading of graphs generated by Pajek to which it can add a specified number of reviews per user of a specified number of web services, along with the conversion of the undirected edges of the graph into weighted and directed graphs

\(^1\)See http://vlado.fmf.uni-lj.si/pub/networks/pajek/
weighted randomly or with a relation to the number of common friends two users have. The trust network and reviews the program then held could be saved to a file or a Pajek graph or loaded into the WSTRN database from which the FOAF files the matchers could use to infer scores could be generated. Statistics on the network are produced by the program and the number of reviews can be reduced by given amounts so that the density of reviews can be altered for testing.

A single trust network graph was generated for testing the project. The Pajek generated graph from which it was generated is shown in Figure 9.1. The graph was generated based on the information discussed in Section 7.1.4. The graph in Figure 9.1 is a scale free network containing 400 nodes (users) with an average degree of 3.19. Trust values were added to the graph such that connected users with higher overlap had more trust between them, although a random element in the assignment existed. Two users connected in the undirected graph did not necessarily both have a trust value for each other in the trust network. The network has 80890 unreachable pairs out of possible 159600 making the average connectance of the network 0.5068 which is less than the average connectance of the Trust Project social network which was 0.66. The average distance between reachable pairs is 4.23067 and the most distant nodes in the network are 12 steps apart. The small world properties of a social network are likely to change over time and not knowing the nature of web service users social links the network serves only as a model. As much as it may contain inaccuracies compared to what reality may reveal, what that reality is is not known at present. It could be argued that a trust network will not completely follow the strength of weak ties hypothesis as social networks based on friendship or acquaintance, because trust in your friends opinions on web services is not likely to be directly related to the number of common friends you have.

Nine different densities of user reviews were used for testing, when referring to a density it is meant the mean number of reviews per user. The statistics of these configurations are shown in Table 9.1. The reviews generated were about thirty imaginary web services. The actual number of reviews supplied is not meant to be realistic as a total but in terms of reviews published per web service it is meant to be representative. It is reasonable to assume more than thirty web services will be reviewable on the WSTRN site after it is properly deployed but by using a sample set of thirty web services the testing process remains reasonably focussed and simple. The scores given by the reviews were picked at random. The accuracy of the inference of users opinions on web services cannot be judged in these tests because the virtual users do not have opinions, as such, to verify these against.

### 9.2 Caching

The trust network matchers by their design are required to utilise information in a number of files that could be hosted anywhere on the internet. The size of the files that are accessed are typically small, but given a large trust network many files can be accessed by the matchers. Certainly the
Figure 9.1: The Pajek generated graph from which the trust network was generated
Multiple Depth matcher algorithm accesses as many documents in its worst case as it does in all cases. In the KNOOGLE brokerage system each matcher specified by the user is sent a query and the WSDL address of each web service in the user’s chosen repositories. This means that the same sets of files will be traversed many times in quick succession. There is therefore a case for storing what is found at files for a period of time to avoid the large time cost associated with the looking up of pages across the internet. The length of time the files information is stored for should be long enough to provide some benefit but not so long that the files may change in that time.

Three forms of caching implemented are discussed in Section 8.3.3 and the effects the caching has on the run-time of the matchers is shown in Table 9.2, Table 9.3, Table 9.4 and Table 9.5.

The times in Table 9.2 show without caching finding the score of each web service in the table takes on average 0.169539 seconds. The location of the FOAF files being accessed in the testing process were on the local network but did not have to be generated by the WSTRN code each time they were accessed. Table 9.3 shows the times for generating scores with the files being accessed being cached for 30 seconds does not improve the speed of the algorithm compared to the results for no caching. The average time for the results in Table 9.3 is 0.222466 seconds and
Table 9.3: Page caching Multiple Depth algorithm matcher example (microsecond) times

<table>
<thead>
<tr>
<th>Users</th>
<th>WS 1</th>
<th>WS 2</th>
<th>WS 3</th>
<th>WS 4</th>
<th>WS 5</th>
<th>WS 6</th>
<th>WS 7</th>
<th>WS 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>User 1</td>
<td>1315638</td>
<td>249465</td>
<td>230156</td>
<td>294331</td>
<td>620902</td>
<td>127355</td>
<td>131957</td>
<td>158878</td>
</tr>
<tr>
<td>User 2</td>
<td>150826</td>
<td>257450</td>
<td>172481</td>
<td>136694</td>
<td>178294</td>
<td>122292</td>
<td>148033</td>
<td>167014</td>
</tr>
<tr>
<td>User 3</td>
<td>143553</td>
<td>188521</td>
<td>205641</td>
<td>109037</td>
<td>109961</td>
<td>193641</td>
<td>147040</td>
<td>108039</td>
</tr>
<tr>
<td>User 4</td>
<td>224674</td>
<td>188071</td>
<td>109719</td>
<td>158307</td>
<td>182279</td>
<td>221719</td>
<td>159926</td>
<td>207547</td>
</tr>
</tbody>
</table>

Table 9.4: Model caching Multiple Depth algorithm matcher example (microsecond) times

<table>
<thead>
<tr>
<th>Users</th>
<th>WS 1</th>
<th>WS 2</th>
<th>WS 3</th>
<th>WS 4</th>
<th>WS 5</th>
<th>WS 6</th>
<th>WS 7</th>
<th>WS 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>User 1</td>
<td>121495</td>
<td>12651</td>
<td>13425</td>
<td>14752</td>
<td>13170</td>
<td>13786</td>
<td>14499</td>
<td>13350</td>
</tr>
<tr>
<td>User 2</td>
<td>181485</td>
<td>26785</td>
<td>27625</td>
<td>27497</td>
<td>27019</td>
<td>27585</td>
<td>27156</td>
<td>27033</td>
</tr>
<tr>
<td>User 3</td>
<td>98159</td>
<td>20227</td>
<td>21089</td>
<td>22584</td>
<td>20706</td>
<td>21171</td>
<td>20802</td>
<td>20997</td>
</tr>
<tr>
<td>User 4</td>
<td>69532</td>
<td>13569</td>
<td>15199</td>
<td>13872</td>
<td>13893</td>
<td>14258</td>
<td>14065</td>
<td>14130</td>
</tr>
</tbody>
</table>

the slow down is believed to be caused by the passing of the cached page to the model generator as the model generator will not directly accept the cached string of the file and the time to store the files themselves. Table 9.4 shows the effects of avoiding the regeneration of the models of the RDF documents and the calculations involved in this appear to be a significant cost with the times in Table 9.2 around nine times longer than those in Table 9.4 after the models have been cached.

Table 9.5 contains the times from caching a list of the addresses identifying the users’ trusted by each user and how much they are trusted as well as all the reviews written by each user. This caching of results of the searching for reviews and connected users shows a clear improvement in run-time of the algorithm. There is clearly a high cost in setting up the cache and there appears to be no reason in the caching used or the Multiple Depth algorithm itself why long times are shown for each search for 'WS 1' so this may be an artifact of the testing code used, virtual machine or operating system conditions. If the set up cost and the anomalous results occurring with the

Table 9.5: Results caching Multiple Depth algorithm matcher example (microsecond) times

<table>
<thead>
<tr>
<th>Users</th>
<th>WS 1</th>
<th>WS 2</th>
<th>WS 3</th>
<th>WS 4</th>
<th>WS 5</th>
<th>WS 6</th>
<th>WS 7</th>
<th>WS 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>User 1</td>
<td>400706</td>
<td>190980</td>
<td>15889</td>
<td>590</td>
<td>498</td>
<td>622</td>
<td>501</td>
<td>7314</td>
</tr>
<tr>
<td>User 2</td>
<td>128622</td>
<td>156690</td>
<td>75289</td>
<td>40561</td>
<td>475</td>
<td>452</td>
<td>386</td>
<td>385</td>
</tr>
<tr>
<td>User 3</td>
<td>185969</td>
<td>360</td>
<td>298</td>
<td>295</td>
<td>296</td>
<td>312</td>
<td>296</td>
<td>296</td>
</tr>
<tr>
<td>User 4</td>
<td>165129</td>
<td>113599</td>
<td>547</td>
<td>456</td>
<td>457</td>
<td>471</td>
<td>470</td>
<td>455</td>
</tr>
</tbody>
</table>
lower numbered web services are ignored, the remaining times shown in the table and higher numbered web services used in the testing show that the results caching algorithm is around 200 times faster than no caching at all. As expected the gains of the caching are noticeable where the second web service is searched for and beyond. The results caching also benefits over the other types of caching explored here in terms of memory used by caching only the necessary information.

The Multiple Depth algorithm searches every reachable user (up until its given maximum depth) and this may include many more users that have no effect on the score than the Single Depth algorithm examines. So it stands to reason that the Single Depth algorithm will experience some gains from the caching but the gains will not be as immediate or necessarily as large as with the Multiple Depth algorithm. Because as long as some of the web services searched for have reachable reviews at a distance less than the edge of the reachable nodes then the Single Depth algorithm will on average have to examine fewer users than the Multiple Depth algorithm. From here on the versions of the algorithm used will include results caching because it is a desirable feature for the matchers to contain.

9.3 The run-times of the algorithms

The gain in time of the Single Depth algorithm over the Multiple Depth algorithm is tied to the number of reviews reachable from the given user. The Multiple Depth algorithm’s run-time lengthens the more reviews there are as it always has to examine each reachable user, but the more reviews of a web service there are the more reviewers the Multiple Depth algorithm has to calculate the trust of and (although less costly computationally) the more review scores have to be combined. The closer to the user reviews for a web service can be found at the quicker the Single Depth algorithm will be. The hit-rate of the algorithms (the percentage of the 400 users that can reach each of the 30 web services) with the sets of reviews used in testing is shown in Figure 9.2. The hit-rate of both algorithms will of course be the same, the hit rate is a property of the connectivity of the network and the density of its reviews. For web services with few or no reviews the difference between the two algorithms will be less noticeable.

Table 9.6, Table 9.7 and Table 9.8 show the average run times of the algorithms. Table 9.6 shows the average times taken when the web service does not exist (or at least no user has a review for it). This is calculated using eighty sets of results from the times of all 400 users. The variability in these averaged results may indicate flaws in the testing conditions but an attempt was made to minimise these. Highly noticeable in these results is the high times for the Single Depth algorithm where there are high numbers of reviews. This is unusual in that there is nothing the Single Depth algorithm does that would not also be done by the Multiple Depth algorithm that could cause such a significant delay.

Table 9.7 shows the times taken by the algorithms averaged over each of the 30 web services,
CHAPTER 9. TESTING AND DISCUSSION

Figure 9.2: The hit-rates of possible user-web service pairs

each of the 400 users and over 80 runs of the test. Here can be seen the predicted increase in the
times of the Multiple Depth algorithm and the decrease in times of the Single Depth algorithm.
Once again the two versions of the Multiple Depth algorithm perform similarly although less
variation in the weighted average Multiple Depth times are visible here.

The times shown in Table 9.8 are the times taken if the times from where the network is ac-
cessed and the anomalies that appear are removed, which seems to be in the first five web service
searches for each user. The “right-average” times are therefore those for the 6th to 30th web
services. Here the time differences between the Single and Multiple Depth algorithms are more
pronounced.
Table 9.6: The worst case running times (in usecs) of the algorithms against number of reviews

<table>
<thead>
<tr>
<th>Reviews</th>
<th>Density</th>
<th>Single Depth</th>
<th>Multiple Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>weighted average</td>
</tr>
<tr>
<td></td>
<td></td>
<td>weighted average</td>
<td>weighted against expectations</td>
</tr>
<tr>
<td>17</td>
<td>0.0425</td>
<td>795.2</td>
<td>813.2</td>
</tr>
<tr>
<td>37</td>
<td>0.0925</td>
<td>783.0</td>
<td>706.5</td>
</tr>
<tr>
<td>76</td>
<td>0.19</td>
<td>469.6</td>
<td>843.1</td>
</tr>
<tr>
<td>163</td>
<td>0.4075</td>
<td>798.7</td>
<td>665.1</td>
</tr>
<tr>
<td>338</td>
<td>0.845</td>
<td>861.0</td>
<td>739.2</td>
</tr>
<tr>
<td>689</td>
<td>1.7225</td>
<td>858.7</td>
<td>650.9</td>
</tr>
<tr>
<td>1329</td>
<td>3.3225</td>
<td>6626.8</td>
<td>1016.9</td>
</tr>
<tr>
<td>1752</td>
<td>4.38</td>
<td>7684.0</td>
<td>917.2</td>
</tr>
<tr>
<td>2305</td>
<td>5.7625</td>
<td>12387.7</td>
<td>973.2</td>
</tr>
</tbody>
</table>

Table 9.7: The average running times (in usecs) of the algorithms against number of reviews

<table>
<thead>
<tr>
<th>Reviews</th>
<th>Density</th>
<th>Single Depth</th>
<th>Multiple Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>weighted average</td>
</tr>
<tr>
<td></td>
<td></td>
<td>weighted average</td>
<td>weighted against expectations</td>
</tr>
<tr>
<td>47</td>
<td>0.0425</td>
<td>1524.5</td>
<td>1549.3</td>
</tr>
<tr>
<td>67</td>
<td>0.0925</td>
<td>1545.8</td>
<td>1603.5</td>
</tr>
<tr>
<td>106</td>
<td>0.19</td>
<td>1442.6</td>
<td>1508.3</td>
</tr>
<tr>
<td>193</td>
<td>0.4075</td>
<td>1458.4</td>
<td>1608.2</td>
</tr>
<tr>
<td>368</td>
<td>0.845</td>
<td>1398.2</td>
<td>1626.6</td>
</tr>
<tr>
<td>719</td>
<td>1.7225</td>
<td>1303.9</td>
<td>1571.8</td>
</tr>
<tr>
<td>1359</td>
<td>3.3225</td>
<td>1077.7</td>
<td>1734.4</td>
</tr>
<tr>
<td>1782</td>
<td>4.38</td>
<td>1070.8</td>
<td>1778.7</td>
</tr>
<tr>
<td>2335</td>
<td>5.7625</td>
<td>973.7</td>
<td>1852.8</td>
</tr>
</tbody>
</table>
Table 9.8: The “right-average” running times (in usecs) of the algorithms against number of reviews

<table>
<thead>
<tr>
<th>Reviews</th>
<th>Density</th>
<th>Single Depth</th>
<th>Multiple Depth</th>
<th>weighted average</th>
<th>weighted against expectations</th>
</tr>
</thead>
<tbody>
<tr>
<td>47</td>
<td>0.0425</td>
<td>706.5</td>
<td></td>
<td>746.0</td>
<td>747.6</td>
</tr>
<tr>
<td>67</td>
<td>0.0925</td>
<td>745.0</td>
<td></td>
<td>790.3</td>
<td>771.5</td>
</tr>
<tr>
<td>106</td>
<td>0.19</td>
<td>623.2</td>
<td></td>
<td>740.2</td>
<td>732.5</td>
</tr>
<tr>
<td>193</td>
<td>0.4075</td>
<td>634.6</td>
<td></td>
<td>775.5</td>
<td>788.5</td>
</tr>
<tr>
<td>368</td>
<td>0.845</td>
<td>599.6</td>
<td></td>
<td>767.5</td>
<td>801.4</td>
</tr>
<tr>
<td>719</td>
<td>1.7225</td>
<td>492.6</td>
<td></td>
<td>655.3</td>
<td>663.0</td>
</tr>
<tr>
<td>1359</td>
<td>3.3225</td>
<td>454.9</td>
<td></td>
<td>758.9</td>
<td>738.4</td>
</tr>
<tr>
<td>1782</td>
<td>4.38</td>
<td>330.9</td>
<td></td>
<td>740.0</td>
<td>708.9</td>
</tr>
<tr>
<td>2335</td>
<td>5.7625</td>
<td>398.4</td>
<td></td>
<td>903.6</td>
<td>955.1</td>
</tr>
</tbody>
</table>
Figure 9.3: The distribution of scores of the three methods
9.4 Distribution of scores

The other dimension with which the algorithms can be tested is the scores they produce. It has already been expressed that the accuracy of the inference cannot be known from the results gathered so far. The accuracy of algorithms is dependent on how well they express what is the users’ interpretation of the trust and opinions they assign. Taking all the scores inferred for every user about every web service, graphs can be produced showing how similar the scores produced are. The way the results are found in each case indicates that the scores they produce will not be similar very often. Analysis of these results shows that the scores they produce are quite different indicating that at least two of them may be inaccurate. The two Multiple Depth methods of computing an opinion can differ by as much as 0.59.

The Single Depth algorithm produces stratified results with the scores often falling on multiples of one tenth. Because the Multiple Depth algorithm methods search for all the opinions they can find, not just the nearest, its scores are less stratified. The clustering of the Multiple Depth algorithm where the review scores are weighted against the user’s expectations is the least significant.
because the opinions of reviews are averaged with the score 0.5 with a weighting according to the trust the user has in the review against their own opinions and as such the clustering appears around multiples of one twentieth. The distribution of the three methods can be seen in Figure 9.3 where the scores are in their rank order from highest to lowest. This graph is produced with an average of 5.7625 reviews per user. If there are fewer reviews the shape of the graphs remain the same but more of the scores are zero for each algorithm. This averaging with expectations produces a smoother curve and the distribution of opinions it infers is Cauchy distribution-like which at first glance is more desirable than the linear distribution of the Single Depth method because it is closer to the normal distribution. A histogram of the distribution is shown in Figure 9.4 with the Normal distribution there for comparison. The Multiple Depth algorithm where the review scores and inferred trust values are combined as a weighted average’s scores are more spread out approximately in between the distributions of the other two methods. Figure 9.5 shows a histogram of that method’s results with the Normal distribution there for comparison. The reviews are generated at random and therefore the opinions of the reviewers are effectively distributed linearly but a linear distribution of inferred opinions is not necessarily more accurate because the random scores mean on average most of the web services average score will be close to 0.5.
The weighting against the users’ expectations suffers from a lack of knowing the users’ expectations. Using the idea that opinions inferred over shorter distances are more accurate it could be argued the most accurate of the algorithms would be the Single Depth algorithm because although it takes in fewer opinions on web services what is inferred from those opinions is more likely to be accurate. The weighting against expectations algorithm was an attempt to account for when opinions were inferred from single not very well trusted users which creates an amount of uncertainty and could mean the users opinion would be more as they expected rather than they were told. But perhaps in reality the uncertainty does not make much more likely that the user’s actual opinion will lie nearer the middle. The weighting against expectations is almost akin to reversing the opinion of those not trusted because they are not trusted to be right. But in cases such as this it does not mean they are likely to be wrong just less likely to be right than someone who is highly trusted. This is a problem with one dimensional inference, where confidence in the result is not returned with the score.

From these results defining a cut-off point for any of the matchers results, above which can be said to be a good match is not possible to say at this point. The rating system implies that anything above 0.5 denotes a good web service but Section 7.2.4 and Section 7.3.2 show that users are inclined to mostly state positive opinions of other users and themselves so a typical score for a web service may be above 0.5. A cut-off point could be taken to be the average of the inferred opinions if one was required.

The Multiple Depth algorithm using weighted averages should be deployed for real world use because it is based on a method already tested by (Golbeck 2005a). The other explored methods appear to have some theoretical groundings that could be explored further but are not expected to be accurate as they operate at the moment.

9.5 Considerations

9.5.1 Feedback

Although the Multiple Depth using weighted averaging method has been shown to be accurate the inference can suffer from not often enough returning results for web services. A good web service may be ignored merely because no one connected to the user has reviewed it yet. If it is not recommended by the KNOOGLE matcher it is unlikely to be reviewed and so there is a situation where the same web services may be consistently recommended merely because they were reviewed first. This could be avoided by giving web services that a review cannot be found for a score of 0.5 - a theoretically average score but if the best of the web services for which opinions can be inferred scores 0.4 there is more certainty that the web service with the score of 0.4 is at least worth the score it has received perhaps. The score of 0.5 could be replaced with a score more connected to the scores to be found in the reviews in the network by using
the average of all the scores of the reviews that turned out not to be for the desired web service. At first, few reviews will exist so a good average will not be reached. Users are also liable to not reviewing the same number of bad services as good services. So, if only good web services were reviewed, then the web services without reviews would be assigned a good score when in reality this average score is inappropriate. An average of the reachable reviews in a network is also costly to compute. So a default score of 0.5 should be awarded to web services which have no review and that this (or 5/10 as the user will know it) is the score of an average web service should be made clear to users. Using a score of 0.5 as the default reinforces it as the average score and lessens the chances of the same web services being recommended again and again and no new services being reviewed as could happen with the presumption that web services without reviews are unendorsed by users, a presumption which appears to be an oversight. The algorithms have been updated to reflect this and the scores produced by this change to the algorithms cause a high concentration of results around the 0.5 mark as would be expected.

The inference of opinion is not simple. Users can easily see why a web service's description appears to have been scored as if it has a similar meaning to a query but without examining the trust network from a global perspective, users cannot so easily see why a they should have a given opinion of a web service. Using KNOOGLE to access the matcher will hide the score received by each web service in any case. This is a flaw in that users cannot correct for errors in inference caused by their misassignment of trust to users by seeing from their results where they are going wrong. Whereas they can alter the type of query they give the Text Semantics Matcher to achieve better results. This lack of connection to how the results come about can mean the users become disenfranchised with the matcher service. Simplicity of the inference algorithm is therefore an important consideration when constructing and improving algorithms.

### 9.5.2 Are the possible trust values evenly spread?

The WSTRN website and the TopicalTrust RDF vocabulary allow users to assign trust values from 1 to 10 to other users. A trust value of nought is implied by the lack of a trust value being assigned. The methods looked at here presume that a user trusted 9/10 should be trusted as much more than a user trusted 8/10 than a user trusted 10/10 is trusted over them. This is what the user should be understanding when assigning the trust values but a different interpretation could be found to be at work. An area wherein the inference values can be tuned further is how the spread of possible trust values is interpreted. Interpreting the trust values using their values squared would place a far higher emphasis on the most trusted users and this may be desirable.

### 9.5.3 Filling the gaps

Initially the matcher if implemented as discussed will return 0.5 as the score for every web service and without the trust network formed experiencing high connectivity and a high number
of reviews (see Figure 9.2) there will be a number of web services for which no accurate value can be inferred and an average will be substituted in its place. To minimise the monotony of the results to the end user, users could be provided with recommendations of similar users through the WSTRN site to increase the connectivity of the network using Automated Collaborative Filtering algorithms discussed in Section 9.5.5.

9.5.4 Speed optimisations

Caching implemented by the matcher itself has been implemented here as placing it there removes its ties to any particular source of users publishing trust information and reviews of web services. The information the matcher has to search for could be presented more directly to it via web services running with access to the WSTRN database. This would require addresses of files on the WSTRN site to be detected and these users handled using the streamlined method for accessing the information.

The documents that are not created by the WSTRN site but are connected from the documents generated by the site can still have information on them cached by the site because the connections can be followed. Search engines manage to provide useable results yet they do not update their data on each site all the time. So precomputing users’ trust for one another and their opinions of web services can be done. When new reviews are added the users who can reach that review could have their opinions recalculated of it and when new trust values are added the trust in users that flows across the link can be reevaluated. This would need tuning to the demands of the site but on a busy enough site changes of trust values for instance will have a small effect on the results and therefore do not have to cause the recalculations to take place.

It is understandable that because of the large time cost of constructing and querying each model of the RDF documents constructed a specialised parser could be written that extracts the information from the document in one parse, merging the model building and querying stages into a single step. A user’s weblog address is not information needed to be stored at any point by the matcher but in building a model it does this.

Allowing the user to control how uncertainty is treated

The algorithms here are discussed as being one-dimensional in that the only value they return is the inferred opinion for a given user and web service and not the uncertainty information that could be important. A user may prefer to have a web service that they are highly likely to think is a nine out of ten web service than a web service that could be a ten out of ten web service but with much more uncertainty surrounding what its value is. It may be desirable therefore to allow the users to say how certain they would like a result to be. The certainty of an inferred opinion can then judged by how close to the opinion the reviews that formed it are and how much trust
there is in the reviewers. The resulting algorithm would still, as far as the scores produced are concerned, be one dimensional but the information previously being lost by the algorithms would be made better use of to meet the users’ needs.

9.5.5 Using Automated Collaborative Filtering algorithms

Automated Collaborative Filtering (ACF) algorithms could be used as part of testing the accuracy of the algorithms implemented in the matchers. As done in Golbeck (2005a) the users could be grouped by similarity of opinion according to their reviews as computed by an ACF algorithm such as the Pearson Correlation algorithm. Then users’ similarity can then be used to infer the opinion of each user for each resource that has been reviewed using a weighted average of the reviews of the resource according to the user’s similarity to the reviewer. The accuracy of the trust inference algorithms using the explicitly defined data can then be tested against this metric that uses implicit user similarity.(Herlocker et al. 2004)
Part IV

Consolidation
Chapter 10

Common features

The algorithms described in the previous two parts were implemented to fit within web services but did not contain appropriate details to be deployed just as they were. Here we describe some of the features that were common to both of the matchers to allow them to be deployed as web services.

At http://alis.cs.bath.ac.uk:9095/axis/services the web services have been deployed using Apache Tomcat\(^1\) and Apache Axis\(^2\). The interfaces for the matchers fits the KNOOGLE requirement of the operations for generating the score being called match and taking two arguments. The first argument being an XML string containing queries from which the matcher should extract the query information that it needs and the second argument being the URL of the WSDL file associated with the user.

10.1 Web service arguments

For the trust matcher the WSDL file’s location along with the user’s FOAF URL from the query is all the matcher needs. But the text semantics matcher requires text from the query and the location of the documentation associated with the web service. To link the WSDL file to documentation the WSDL file contains an element within the definitions element of the WSDL file that indicates the location of an XML documentation file associated with the web service. It may be more desirable in the future to avoid annotating the WSDL file directly and instead having the matcher retrieve the documentation’s location from the web service’s Grimoire repository but this would require changes to how KNOOGLE operates and this is discussed in Chapter 12. The text semantics matcher then uses the document type declaration (doctype) of the documentation

\(^{1}\)See http://tomcat.apache.org

\(^{2}\)See http://ws.apache.org/axis
file to select an appropriate handler to use for the file to extract the text to be used to calculate the web services score against the query.

At present only two handlers have been created. A default handler that extracts all the text not including the XML tags themselves is used as a fall back for when documentation of an unrecognised doctype is encountered. Details of the second handler, for the NAG library documentation we covered in Section 5.4. As the present state of affairs means that a number of other handlers will have to be created and added at a later date the text semantics matcher web service was designed to allow the addition of new matchers without recompilation. This is done by on the starting of the web service loading information from a properties file. Properties files are files that contain strings labelled with keys that used to access the strings. The properties file can contain entries with keys beginning “Handler” that contain the full Java path of implementations of the Handler interface. Using the Java class loader in such a way that the constructor of the objects are called without any arguments the design allows options that would normally be set in constructor arguments to be set after the class has been instantiated. To do this the Java’s reflection capabilities are used to search the newly instantiated objects for methods with names beginning `set__` and on their discovery the properties file is searched for an entry under the remainder of the method’s name. Methods with names beginning `set__` should take a single string as an argument and the entry in the properties file associated with the method is passed to the method of the newly instantiated object as its single argument. This process allows control over the classes used by the matcher and their parameters to be altered without editing the Java code, recompiling and deploying.

The text semantics matcher is also implemented to take its parameters from a properties file so that the thesaurus class used and other parameters can be easily changed. All the classes loaded dynamically use the pattern of allowing argument less constructors that load default parameters and sets of methods with names beginning `set__`. This prefix for method names was chosen to avoid clashing with methods that were not intended for the purpose of being loaded in this dynamic way.
Chapter 11

Critique

Here we summarise the results and review the method used.

11.1 The Text Semantics Matcher

It appears the algorithms developed for the text semantics matcher are in need of further work. They were designed to be fast and to do so use a simple algorithm that works only over terms and all the terms are treated equally. There is no sense of particular terms operating on other terms, the limiting of the context terms are used within or the part of speech that the terms have. It is conceivable that such additions would improve the results of the algorithm. They also suffer from the over-keenness problem that fixing may improve the algorithms results somewhat but this problem is not as large a contributor as the matching of unexpected terms appears to be. It appears that the over-keenness problem could be changed with the use of a different thesaurus library for accessing WordNet.

The simple algorithms produced here are too simple. A criticism that might be levelled at the matcher’s design is that it was not designed with enough sophistication. But the design is extendable so that further work can improve the results and the design was limited by the time available to complete its testing and implementation.

The matcher does seem to return high scores for the web service descriptions that are expected to get high scores and the matcher can be said to work in general cases. But with the simplicity of its algorithm and without the specialised knowledge it is liable to make mistakes and the scores it returns are generally higher than what they should be as it can be seen that less overlap in meaning exists than the scores reflect.

How much meaning the scores reflect is an area in which the results suffer because this information does not exist. Discussion of how data for comparison can be generated is found in
Section 5.7.5. The accuracy of the algorithm cannot be directly judged without this information but the results do show that more overlap in meaning is detected than should be expected. The improvement that the matcher provides over simple keyword based searching is not directly shown either. But it is clear the increase in complexity over simple keyword based searching implies that an improvement occurs and the results in Figure 5.2 show the extra information that is gained over simply searching within the synset of terms (where the distance to travel is set to be nought). The searching using just synonyms being halfway between searching using just the given terms and searching using terms with the same and related meaning as is being done here.

More time could easily be spent on the development and tuning of a text semantics algorithm that operates quickly enough to be used within the KNOOGLE system. More time could be spent developing an appropriate thesaurus for the system. A number of avenues for further development have been uncovered but the matcher developed does provide a search based score that could be used in combination with other matchers within the KNOOGLE system.

11.2 The Trust Matcher

The work in designing, developing and testing the trust using matcher has led to the development of a number of pieces of software, a number of algorithms and an understanding of what further requirements. More polish of the social networking site and fine tuning of the algorithm may be necessary but only the promotion of the service appears needed for the matcher to successfully operate.

We believe we have shown that an opinion inferring matcher has a place in the KNOOGLE matching services using, unless a better algorithm is found, the Multiple Depth weighted averaging method with an average value as the default. Although there are a number of caveats that need to be considered. The WSTRN site, or a site similar to it, needs to be well promoted within the userbase so that trust networks of sufficient size form. Initial runs of the matcher will likely produce little information for users so encouraging reviewing is important to increase the value of the matcher. Even with a high number of reviews many web services may be left without reviews and in these cases the matcher is limited through its dependence on users creation of reviews and connections across the network. The accuracy of the algorithms themselves should be investigated further and fine tuning of the current best algorithm for the task to utilise some of the principles explored by the others should produce a benefit. Given real world data parameters such as the effect of distance on the inferred value, the effect of doubt on the score and the actual value to be used from the user specified trust values can be tuned using techniques such as genetic algorithms or alpha-beta pruning.
11.3 Overall

The lack of real world data has effected the testing of the accuracy of both the matchers. Particularly the text semantics matcher, as the trust matcher has a basis in a proven algorithm whereas the text semantics matcher has more parameters and factors which effects the results it produces. Although the variations on the TidalTrust algorithm experimented with have not been able to have their accuracy tested because of a lack of values to check their accuracy against some exploration of the algorithms concepts could still occur.

We feel we have succeeded in developing two new matchers for KNOOGLE and laid foundations for their further development as well as the development of other matchers. Such as matchers using the labels generated by the WSTRN site and through the use of the Grimoires Synchroniser discussed in Section 8.1.4 and Section 8.2. We believe the application of the techniques developed in the trust based matcher to be novel, although the algorithms have been based on the work of Golbeck (2005a) effort has been made to explore areas not covered by that work. The text semantics matcher tackles a well investigated area with an attempt to simplify text analysis methods for fitting within a matcher and admittedly suffers from its over simplicity but succeeds in laying the groundwork for a more successful matcher.
Chapter 12

Notes on KNOOGLE

After developing two matchers for the KNOOGLE broker it was felt certain observations on its design could be made to allow more possibilities for matcher development.

12.1 New dimensions

The KNOOGLE design requires matchers to return a value to double floating point precision as their result. This could be seen as limiting the ways in which the matchers values can be combined. For instance the second value that could be returned from a trust matcher, similar to the one built for this project, would be a confidence value in the accuracy of the result. Users may wish to choose a web service that scores highly and has a high confidence value for each of its results from a variety of matchers. Inferred opinion using trust is inherently not one dimensional and other matchers may be able to provide a confidence value or another type of second dimension to their current results.

The selection policy for calculating the final scores of web services provided by the user when they submit their queries can already use the scores returned by matchers by referring to the matchers that generated them. So it could be claimed that there is not reason a matcher could not act as more than one matcher and the confidence value in the inferred opinion in the trust matcher be treated as a value from a matcher that returns the confidence value in an inferred opinion of trust. The calculating of the confidence value separately would be costly in terms of carrying out near identical computations twice.

The KNOOGLE design does allow factors such as the confidence value to be taken into account within the matchers if they choose to support options on how they should deal with the confidence value passed as an argument in the query sent by the user. This does not allow the use of two matchers within one as outlined above but can be used to a similar effect. Matcher designers
may be less inclined to consider allowing the user access to such information as the KNOOGLE system is currently structured and the allowing of multiple, perhaps labelled (by the “virtual” matcher that they are from) return values would make designers consider allowing their matchers to return more information.

12.2 Repository access

A matcher must work with information from just the user’s query and the WSDL file and its address. If a third item was passed to the matcher, the Grimoires repository that the web service was registered in, then the matcher could query the Grimoires repository and extract additional information that it could use to calculate a score. An example being the address of RDF documents describing each web service that are generated by the WSTRN web site but associated with particular web services and their operations via the Grimoires Synchroniser program. This could also mean the location of documentation files does not have to be stored within WSDL files but could be stored within a Grimoires repository.

12.3 Operation names

This is perhaps the most trivial of the observations but the KNOOGLE system appears to be limited to interfacing only with matchers with an operation called “match”. This means each matcher can only publish one interface that can be used by KNOOGLE and generally limits the flexibility of the connection process.
Bibliography


URL: citeseer.ist.psu.edu/article/chai97use.html


URL: citeseer.ist.psu.edu/lin98automatic.html


URL: http://csdl.computer.org/dl/proceedings/wi/2006/2747/00/274700442.pdf

URL: citeseer.ist.psu.edu/nichols98implicit.html

URL: http://arxiv.org/abs/physics/0610104

URL: citeseer.ist.psu.edu/page98pagerank.html

URL: http://portal.acm.org/citation.cfm?id=646996.711287


Tsujii, J. & Ananiadou, S. (2005), ‘Thesaurus or logical ontology, which one do we need for text mining?’, Language Resources and Evaluation 39(1), 77–90.


   &oldid=115846624


   &oldid=118821475

   &oldid=119283696

   &oldid=119572469

   &oldid=117633171


   URL: citeseer.ist.psu.edu/ziegler04analyzing.html
Part V

Appendix
Appendix A

Code

This appendix contains a sample of the code used to implement the algorithms for the matchers created by this project. For the remainder of the matcher code, deployment instructions and the code and documentation for the other software developed for this project please see the attached CD.
A.1 Semantic Text Matchmaker

A.1.1 File: Matcher.java

```java
package uk.ac.bath.stamm;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.StringReader;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.util.Enumeration;
import java.util.HashMap;
import java.util.InvalidPropertiesFormatException;
import java.util.Properties;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.XMLReaderFactory;
import uk.ac.bath.SingleXMLArgumentReader;
import uk.ac.bath.SingleXMLAttributeReader;

public class Matcher
{
    private static ISemanticScorer scorer;
    private static HashMap handlers;
    private static int distanceToTravel;
    private static IWighting weighting;
    private static IWebServiceDescriptionHandler defaultHandler;
    private static String wsdlDescriptionElementUri;
    private static String wsdlDescriptionElementLocalName;
    private static String wsdlDescriptionAttribute;
    private static String textSemanticsQueryElement;

    public Matcher()
    {
        if (Matcher.scorer == null)
        {
            Properties props = new Properties();
            try
            {
                InputStream is =
                             this.getClass().getClassLoader().getResourceAsStream("TextSemanticsMatcherProperties.xml");
                if (is == null)
                {
                    is = new FileInputStream("/TextSemanticsMatcherProperties.xml");
                    props.loadFromXML(is);
                }
            }
            catch (InvalidPropertiesFormatException e)
            {
                e.printStackTrace();
                System.err.println("Unable to load preferences from TextSemanticsMatcherProperties.xml\n" + "Please ensure this file is in the directory tomcat was launched from.");
            }
            catch (IOException e)
            {
            }
        }
```

/∗ ∗
∗ @author nick
∗ The class that gets turned into the web service interface.
∗ It initialises the matcher specified in the TextSemanticsMatcherProperties.xml file using other information from the file by using the names of all the methods of the matcher that begin set_ . ∗
/∗
```
APPENDIX A. CODE

```java
{ e.printStackTrace();
    System.err.println("Unable to load preferences from TextSemanticsMatcherProperties.xml" +
        "Please ensure this file is in the directory tomcat_was_launched from.");
}
Matcher.scorer = (ISemanticScorer)
    this.getInstanceOfClassFromProperties(props, "TextSemanticsScorer", "uk.ac.bath.stamm.QueryToDescriptionScorer");
if (Matcher.scorer == null)
    Matcher.scorer = new QueryToDescriptionScorer();
Matcher.weighting = (IWeighting)
    this.getInstanceOfClassFromProperties(props, "Weighting", "uk.ac.bath.stamm.Weighting");
if (Matcher.weighting == null)
    Matcher.weighting = new Weighting();
this.initializeAppFromProperties(Matcher.scorer, props);
this.initializeAppFromProperties(Matcher.weighting, props);
String distanceToTravel =
    props.getProperty("DistanceToTravel", "3");
Matcher.distanceToTravel =
    Integer.parseInt(distanceToTravel);
Matcher.wsdlDescriptionElementUri =
    props.getProperty("WSDLDescriptionElementURI", "http://www.grimoires.org/wsdocumentation/");
Matcher.wsdlDescriptionElementLocalName =
    props.getProperty("WSDLDescriptionElementLocalName", "documentation");
Matcher.wsdlDescriptionAttribute =
    props.getProperty("WSDLDescriptionAttribute", "location");
Matcher.textSemanticsQueryElement =
    props.getProperty("TextSemanticsQueryElement", "textsemanticsquery");
Matcher.handlers = new HashMap();
Enumeration keys = props.keys();
while (keys.hasMoreElements())
    { String key = (String)keys.nextElement();
        if (key.startsWith("Handler")) // handlers are anything beginning with Handler
            { String handlerS = props.getProperty(key, "");
                if (handlerS.length() > 0)
                    try
                        { Class hC =
                            this.getClass().getClassLoader().loadClass(handlerS);
                            IWebServiceDescriptionHandler iwsdh =
                                (IWebServiceDescriptionHandler)hC.newInstance();
                            String[] doctypes =
                                iwsdh.getDocTypesHandled();
                            if (doctypes != null)
                                for (int i = 0; i < doctypes.length; i++)
                                    { Matcher.handlers.put(doctypes[i], iwsdh);
                                        this.initializeAppObjectFromProperties(iwsdh, props);
                                    }
                        }
                    catch (ClassNotFoundException e)
                        { e.printStackTrace();
                            System.err.println("Class could not be found");
                        }
                    catch (InstantiationException e)
                        { e.printStackTrace();
                            System.err.println("Class could not be instantiated");
                        }
                    catch (IllegalAccessException e)
                        { e.printStackTrace();
                            System.err.println("Class could not be accessed");
                        }
                }
            }
        Matcher.defaultHandler = new TextWebServiceDescriptionHandler();
    }
```
/**
 * @param query XML string containing users FOAF file URL between <foaf:profile> tags
 * @param wsURL URL of the web service's WSDL file
 * @return a score between 0 and 1
 */
public Double match(String query, String wsURL)
{
    try
    {
        XMLReader rdr = XMLReaderFactory.createXMLReader("org.apache.xerces.parsers.SAXParser");
        SingleXMLArgumentReader sxar = new SingleXMLArgumentReader(Matcher.semanticsQueryElement);
        rdr.setContentHandler(sxr);
        rdr.parse(new InputSource(new StringReader(query)));
        String queryText = sxar.getElementText();
        String descriptionText = this.getDescriptionText(wsURL, rdr);
        if (queryText.length() > 0)
            return new Double(Matcher.scorer.score(queryText, descriptionText, Matcher.distanceToTravel, Matcher.weighting));
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
    return new Double(0);
}

/**
 * @param wsURL the WSDL URL to extract the description from
 * @param rdr the XMLReader object to use
 * @return
 */
private String getDescriptionText(String wsURL, XMLReader rdr)
{
    SingleXMLAttributeReader sxar = new SingleXMLAttributeReader(
            Matcher.wsdlDescriptionElementUri,
            Matcher.wsdlDescriptionElementLocalName,
            Matcher.wsdlDescriptionAttribute);
    rdr.setContentHandler(sxr);
    try
    {
        DocumentBuilderFactory docBuilderFactory = DocumentBuilderFactory.newInstance();
        docBuilderFactory.setExpandEntityReferences(false);
        Document doc = null;
        try
        {
            DocumentBuilder docBuilder = docBuilderFactory.newDocumentBuilder();
            doc = docBuilder.parse(webServiceDescriptionLocation);
        }
        catch(ParserConfigurationException e)
        {
            e.printStackTrace();
        }
        catch(SAXException e)
        {
            e.printStackTrace();
        }
        catch( IOException e )
        {
            e.printStackTrace();
        }
        String docType = doc.getDocumentElement().getTagName();
        IWebServiceDescriptionHandler iwsdh = Matcher.handlers.get(docType);
        if(iwsdh == null)
            iwsdh = Matcher.defaultHandler;
        return iwsdh.getText(doc);
    }
*/
private Object getInstanceOfClassFromProperties(Properties props, String classPropertyName, String defaultClass)
{
    String tssS = props.getProperty(classPropertyName, defaultClass);
    try
    {
        Class tssC =
            this.getClass().getClassLoader().loadClass(tssS);
        return tssC.newInstance();
    }
    catch(ClassNotFoundException e)
    {
        e.printStackTrace();
        System.err.println("Class could not be found, using "+ defaultClass);
    }
    catch(InstantiationException e)
    {
        e.printStackTrace();
        System.err.println("Class could not be instantiated, using "+ defaultClass);
    }
    catch(InvocationTargetException e)
    {
        e.printStackTrace();
        System.err.println("Class could not be accessed, using "+ defaultClass);
    }
    return null;
}

/**
*  Looks up the remainder of the method name in the properties
*  file and sends the entry in the properties file to the objects
*  method that begins set_ as a string.
*  @param o the object to initialise from the properties file
*  @param props the properties file to get the initialisation information from
*  @throws SecurityException
*/
private void initialiseObjectFromProperties(Object o, Properties props) throws SecurityException
{
    Method[] methods = o.getClass().getMethods();
    for(int i = 0; i < methods.length; i++)
    {
        if (methods[i].getName().startsWith("set_"))
        {
            String key = methods[i].getName().substring(5);
            String property = props.getProperty(key);
            if (property != null)
            {
                try
                {
                    methods[i].invoke(o, new Object[]{property});
                }
                catch(InvocationTargetException e)
                {
                    e.printStackTrace();
                }
                catch(IllegalAccessException e)
                {
                    e.printStackTrace();
                }
                catch(IllegalArgumentException e)
                {
                    e.printStackTrace();
                }
                catch(IllegalAccessError e)
                {
                    e.printStackTrace();
                }
                catch(InvocationTargetException e)
                {
                    e.printStackTrace();
                }
            }
        }
    }
}
A.1.2 File: QueryToDescriptionScorer.java

```java
/**
 * A scorer that tries to find words in the query in the text to match it against.
 * For details of the algorithm used see the accompanying dissertation.
 */
public class QueryToDescriptionScorer implements ISemanticScorer {
    private TermToDescriptionScorer stws;
    private IThesaurus thesaurus;

    /**
     * @author nick
     * @param thesaurus the thesaurus to use in the scoring
     */
    public QueryToDescriptionScorer(IThesaurus thesaurus) {
        // thesaurus needs extracting
        thesaurus = new TermToDescriptionScorer(thesaurus);
    }

    /**
     * Creates an uninitialised scorer that should be initialised using the set_. methods.
     */
    public QueryToDescriptionScorer() {
    }

    /**
     * @param thesaurus the full classpath of the thesaurus class to use
     * @throws ClassNotFoundException
     * @throws InstantiationException
     * @throws IllegalAccessException
     */
    public void set_Thesaurus(String thesaurus) throws ClassNotFoundException,
             InstantiationException,
             IllegalAccessException {
        Class class = this.getClass().getClassLoader().loadClass(thesaurus);
        thesaurus = (IThesaurus) class.newInstance();
        stws = new TermToDescriptionScorer(this.thesaurus);
    }

    /**
     * @param stopWordListLocation the address at which the list of stop words is located
     * @throws FileNotFoundException
     */
    public void set_StopWordListLocation(String stopWordListLocation) throws FileNotFoundException {
        StopWordStripper.initialize(stopWordListLocation);
    }

    /** (non-Javadoc)
     * @see uk.ac.bath.stamm.ISemanticScorer#score(java.lang.String, java.lang.String, int, uk.ac.bath.stamm.IWeighting)
     */
    public double score(String query, String text, int distanceToTravel, IWeighting weighting) {
        HashSet wordsInText = this.getWordsToLookFor(text);
        StreamTokenizer queryTokenizer = new StreamTokenizer(new StringReader(query));
        queryTokenizer.eofIsSignificant(false);
        queryTokenizer.lowerCaseMode(true);
        int tt = count = 0; // could use different values of count increment to weight different types of token found
        double total = 0.0;
        try
```
```java
while ((tt = queryTokenizer.nextToken()) != StreamTokenizer.TT_EOF)
{
    HashSet wordsToLookFor = (HashSet) wordsInText.clone();
    switch (tt)
    {
    case "":
    case StreamTokenizer.TT_WORD:
    {
        String word = queryTokenizer.sval;
        if (StopWordStripper.isGoWord(word))
        {
            // might be worth caching word and score incase it is in the query twice
            double score = this.stws.score(word, wordsToLookFor, text, distanceToTravel, weighting);
            total += score;
            count++;
        }
        break;
    }
    case StreamTokenizer.TT_NUMBER:
    {
        String word = this.getNumberAsString(queryTokenizer.nval);
        double score = this.stws.score(word, wordsToLookFor, text, distanceToTravel, weighting);
        total += score;
        count++;
        break;
    }
    case '"':
    case '.':
    case ',':
    case '\\':
    default:
    {
        System.out.println((char) tt); // use of unusual characters can cause problems
    }
    }
    }
}
```
A.1.3 File: TermToDescriptionScorer.java

```java
/**
 * Searches for the term and words in the same synset and synsets reachable from it in the description text.
 * For details on the implementation see the Query-to-description algorithm in the accompanying dissertation.
 */
public class TermToDescriptionScorer {

    public static final double MAX_WORD_SCORE = 1.0;
    protected IThesaurus thesaurus;

    public TermToDescriptionScorer(IThesaurus thesaurus) {
        this.thesaurus = thesaurus;
    }

    /**
     * Searches for the term and words in the same synset and synsets reachable from it in the description text.
     * @param word the term to branch from
     * @param wordsToLookFor Hashset of the terms in the description
     * @param text the description text
     * @param distanceToTravel the maximum distance from the source synset that can be searched with
     * @param weighting the weighting object for the source synset
     * @return a number between 0 and 1 representing the closeness in meaning between the terms
     */
    public double score(String word, HashSet wordsToLookFor, String text, int distanceToTravel, IWeighting weighting) {
        double max_match_score = TermToDescriptionScorer.MAX_WORD_SCORE;
        ArrayList searchSets = this.thesaurus.getSensesOfWord(word, weighting);
        SortedList paths = new SortedList(SearchPath.getComparator());
        Iterator it = searchSets.iterator();
        while (it.hasNext()) {
            ASearchSynset ass = (ASearchSynset) it.next();
            Iterator words = ass.getWordIterator();
            while (words.hasNext()) {
                Object o = words.next(); // may want to check for each version of word
                String[] multipleWords;
                if (this.thesaurus.isMultipleWords(o) != null) {
                    if (this.splitTextByMultipleWords(text, multipleWords)[0].length() < text.length())
                        return max_match_score * ass.getWeighting().getValue();
                    else if (wordsToLookFor.contains((String) o))
                        return max_match_score * ass.getWeighting().getValue();
                }
                if (distanceToTravel >= 1)
```
```
```java
paths = this.addPathsFrom(paths, ass, 1);
}
while (!paths.isEmpty())
{
    SearchPath sp = (SearchPath) paths.pop();
    if (sp.getWeighting().getValue() == 0.0) return 0.0;
    Collection ass = this.followPath(sp);
    it = ass.iterator();
    while (it.hasNext())
    {
        ASearchSynset ass = (ASearchSynset) it.next();
        Iterator words = ass.getWordIterator();
        while (words.hasNext())
        {
            Object o = words.next(); // may want to check for each version of word
            String[] multipleWords =
                this.thesaurus.isMultipleWords(o) != null
                    ?
                    (this.splitTextByMultipleWords(text, multipleWords)[0], length() < text.length())
                    return max_match_score * ass.getWeighting().getValue();
                else if (wordsToLookFor.contains((String) o))
                    return max_match_score * ass.getWeighting().getValue();
            if (distanceToTravel > sp.getDepth())
                paths = this.addPathsFrom(paths, ass, sp.getDepth()+1);
        }
    }
    return 0.0;
}

/**
 * @param text the text to split by the multiple words
 * @param multipleWords the multiword term as individual words
 * @return the text split up by occurrences of the multiword term
 */
private String[] splitTextByMultipleWords(String text, String[] multipleWords)
{
    StringBuffer sb = new StringBuffer();
    int lastWordIndex = multipleWords.length - 1;
    for (int i = 0; i < lastWordIndex; i++)
    {
        sb.append(multipleWords[i]);
        sb.append("\\-\_");
    }
    sb.append(multipleWords[lastWordIndex]);
    String[] gaps = text.split(sb.toString());
    return gaps;
}

/*
 * @param sp the path to follow
 * @return a collection of ASearchSynsets found by following the path
 */
protected Collection followPath(SearchPath sp)
{
    switch (sp.getType())
    {
    case ASearchSynset.DIRECTHYPERNYMS:
        return this.thesaurus.getDirectHyponyms(sp);
    case ASearchSynset.DIRECTHYPONYMS:
        return this.thesaurus.getDirectHypernyms(sp);
    case ASearchSynset.HOLONYMS:
        return this.thesaurus.getHolonyms(sp);
    case ASearchSynset.MERONYMS:
        return this.thesaurus.getMeronyms(sp);
    case ASearchSynset.DERIVATIVES:
        return this.thesaurus.getDerivations(sp);
    case ASearchSynset.SEEALSO:
        return this.thesaurus.getSeeAlses(sp);
    }
    return null;
}
A.1.4 File: SearchPath.java

```java
package uk.ac.bath.stamm;
import java.util.Comparator;

/**
 * @author nick
 * A class representing paths for future exploration in the thesaurus. Akin to a set of ASearchSynsets without
 */
public class SearchPath {
    private IWeighting weighting;
    private int type;
    private int depth;
    private ASearchSynset parent;
    
    protected void addPathsFrom(SortedList paths, ASearchSynset ass, int depth) {
        boolean[] p = ass.getPath();
        if (p[ASearchSynset.DIRECTHYPERNYMS]) // a word is a type of its hypernym
            paths.add(new SearchPath(ass.getWeighting().getNewWeighting(ASearchSynset.DIRECTHYPERNYMS, ass),
                                       ASearchSynset.DIRECTHYPERNYMS, ass, depth));
        if (p[ASearchSynset.DIRECTHYPERONYMS]) // a hyponyms is a type of its word
            paths.add(new SearchPath(ass.getWeighting().getNewWeighting(ASearchSynset.DIRECTHYPERONYMS, ass),
                                       ASearchSynset.DIRECTHYPERONYMS, ass, depth));
        if (p[ASearchSynset.HOLOONYMS]) // a word is a part of its holonyms
            paths.add(new SearchPath(ass.getWeighting().getNewWeighting(ASearchSynset.HOLOONYMS, ass),
                                       ASearchSynset.HOLOONYMS, ass, depth));
    }
}
```
A.1.5 File: ASearchSynset.java

```java
/* @param weighting the object representing the weighting given to this synset */
/* @param type the relationship this synset has to the source synset */
/* @param parent the synset that this is a path from */
/* @param depth the distance from the source term's synset that this synset is found */

public SearchPath(IWeighting weighting, int type,
        ASearchSynset parent, int depth)
{
    this.weighting = weighting;
    this.type = type;
    this.depth = depth;
    this.parent = parent;
}

/* @return the synset that this is a path from */
public ASearchSynset getParent()
{
    return this.parent;
}

/* @return the object representing the weighting given to the synsets at the end of this path and relationships traversable from there */
public IWeighting getWeighting()
{
    return this.weighting;
}

/* @return the distance from the source synset that this path goes */
public int getDepth()
{
    return this.depth;
}

/**
 * @return the type relationship to the source synset that this path represents
 */
public int getType()
{
    return this.type;
}

/**
 * @return a comparator that compares the paths by weighting
 */
public static Comparator getComparator()
{
    return new SearchPathComparator();
}

/**
 * a comparator that compares the paths by weighting
 */
private static class SearchPathComparator implements Comparator
{
    public int compare(Object o1, Object o2)
    {
        return (int)(((SearchPath)o1).getWeighting().getValue() - ((SearchPath)o2).getWeighting().getValue())*10.0;
    }
}
```
package uk.ac.bath.stamm;
import java.util.Comparator;
import java.util.Iterator;

/**
 * A for class representing a synset from thesaurus.
 * Defining the score finding a term from the synset
 * can give and the synsets relationship to the source
 * term
 */

public abstract class ASearchSynset {
    // a word is a type of its hypernym
    public static final int DIRECTHYPERNYMS = 0;
    // a hyponyms is a type of its word
    public static final int DIRECTHYPONYMS = 1;
    // a word is a part of its holonyms
    public static final int HOLONYMS = 2;
    // a word is made of its meronyms
    public static final int MERONYMS = 3;
    public static final int DERIVATIVES = 4;
    public static final int SEEALSO = 5;
    public static final int ORIGINAL = 6;
    // for words which do not exist in the thesaurus
    public static final int UNKNOWN = 7;

    /**
     * A representation of the score that can be awarded
     * for finding this synset and the possible relationships
     * that can be followed from here.
     */
    protected IWeighting weighting;

    /**
     * The distance from the source synset.
     */
    protected int depth;

    /**
     * The relationship this synset has to the source synset
     */
    protected int type;

    /**
     * @param weighting the object representing the weighting
     * given to this synset
     * @param depth the distance from the source term's
     * synset that this synset is found
     */
    public ASearchSynset(IWeighting weighting, int depth, int type) {
        this.weighting = weighting;
        this.depth = depth;
        this.type = type;
    }

    /**
     * @return the distance from the source term's synset
     * that this synset is found
     */
    public int getDepth() {
        return depth;
    }

    /**
     * @return the object representing the weighting given to
     * this synset and relationships traversable from here
     */
    public IWeighting getWeighting() {
        return weighting;
    }

    /**
     * @return the relationship this synset has to the source
     * synset
     */
    public int getType() {
        return type;
    }

    /**
     * @return an array that states for each type defined in
     * this class which is allowed to be visited next
     */
    public boolean[] getPath() {
        return weighting.getPaths(this);
    }
}
public int compare(ASearchSynset arg0, ASearchSynset arg1) //should ensure sorting from lowest to highest
{
    double w0 = arg0.getWeighting().getValue();
    double w1 = arg1.getWeighting().getValue();
    if (w0 == w1)
        return 0;
    else if (w0 < w1)
        return 1;
    else
        return -1;
}

/* (non-Javadoc)
 * @see java.util.Comparator#compare(T, T)
 */
public int compare(Object o1, Object o2)
{
    return this.compare((ASearchSynset)o1, (ASearchSynset)o2);
}

A.1.6 File: Weighting.java

/*
 * @return an iterator over the terms in this synset as strings
 */
public abstract Iterator getWordIterator();

/*
 * @return a comparator that compares the synsets by weighting
 */
public static Comparator getComparator()
{
    return new SearchSynsetComparator();
}

/*
 * a comparator that compares the synsets by weighting
 */
private static class SearchSynsetComparator implements Comparator
{
    /* (non-Javadoc)
     * @see java.util.Comparator#compare(T, T)
     */
    public int compare(Object o1, Object o2)
    {
        return this.compare((ASearchSynset)o1, (ASearchSynset)o2);
    }
}

/*
 * An array used to determine the valid paths that may be taken.
 * indexed according to the relationship index values from the
 * ASearchSynset class
 */
private static final boolean[][] paths = new boolean[][]
{
    {{true, false, false, false, false, false, false},
        {false, true, false, false, false, false, false},
        {false, false, false, false, false, false, false},
        {false, false, false, false, false, false, false},
        {false, false, false, false, false, false, false},
        {false, false, false, false, false, false, false},
        {false, false, false, false, false, false, false}};
};
private double[] weightingArray;
private double weighting;

/**
 * @param array an array of the 6 values multiplied by
 * the parent weighting for traversing each type of
 * relationship as defined in the ASearchSynset class
 * @param weighting the value given to this weighting
 */
public Weighting(double[] array, double weighting) {
    this.weightingArray = array;
    this.weighting = weighting;
}

/**
 * The default constructor. The values of the weighting
 * and the
 * weighting array should be set by the set_.. methods.
 */
public Weighting() {
    this(new double[]{0.5, 0.85, 0.3, 0.3, 0.3, 0.2}, 1.0);
}

/**
 * @param weightingArray a string of 6 double values
 * separated by whitespace
 */
public void set_WeightingArray(String weightingArray) {
    StringTokenizer st = new StringTokenizer(weightingArray);
    ArrayList weights = new ArrayList();
    while (st.hasMoreElements()) {
        weights.add(st.nextToken());
        if (weights.size() == 6) {
            this.weightingArray = new double[6];
            for (int i = 0; i < 6; i++)
                this.weightingArray[i] = Double.parseDouble((String)weights.get(i));
        }
    }
}

else
    throw new RuntimeException("The provided weighting_
array contains the wrong number of weights");

/**
 * @param weighting a string representation of a double
 * of the value given to this weighting object
 */
public void set_InitialWeighting(String weighting) {
    this.weighting = Double.parseDouble(weighting);
}

/*@ (non-Javadoc)
* @see uk.ac.bath.stamm.IWeighting#getNewWeighting(int,
* uk.ac.bath.stamm.ASearchSynset)
*/
public IWeighting getNewWeighting(int type, ASearchSynset ass) {
    return new Weighting(this.weightingArray, this.weighting * this.weightingArray[type]);
}

/*@ (non-Javadoc)
* @see uk.ac.bath.stamm.IWeighting#getValue()
*/
public double getValue() {
    return weighting;
}

/*@ (non-Javadoc)
* @see uk.ac.bath.stamm.IWeighting#getPaths(uk.ac.bath.stamm.ASearchSynset)
*/
public boolean[] getPaths(ASearchSynset ass) {
    return Weighting.paths[ass.getType()];
}
package uk.ac.bath.stamm;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.StreamTokenizer;
import java.io.StringReader;
import java.util.Collection;
import java.util.Comparator;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import uk.ac.bath.sortedList.SortedList;

package uk.ac.bath.stamm;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.StreamTokenizer;
import java.io.StringReader;
import java.util.Collection;
import java.util.Comparator;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import uk.ac.bath.sortedList.SortedList;

import uk.ac.bath.sortedList.SortedList;

public class DescriptionToQueryScorer implements ISemanticScorer {
    private int numberOfWordsInTerm;
    private TermToQueryScorer stws;
    private IThesaurus thesaurus;

    /**
     * @author nick
     * @param thesaurus the thesaurus to use in the scoring
     */
    public DescriptionToQueryScorer(IThesaurus thesaurus)
    {
        this.stws = new TermToQueryScorer(thesaurus);
        this.thesaurus = thesaurus;
        this.numberOfWordsInTerm = 4;
    }

    /**
     * @param numberOfWordsInTerm the maximum length of term to search for
     */
    public DescriptionToQueryScorer(IThesaurus thesaurus, int
            numberOfWordsPerTerm)
    {
        this.stws = new TermToQueryScorer(thesaurus);
        this.thesaurus = thesaurus;
        this.numberOfWordsInTerm = numberOfWordsPerTerm;
    }

    public DescriptionToQueryScorer() {
        this.numberOfWordsInTerm = 4;
    }

    /**
     * @param thesaurus the full classpath of the thesaurus class to use
     * @throws ClassNotFoundException
     * @throws InstantiationException
     * @throws IllegalAccessException
     */
    public void setThesaurus(String thesaurus)
            throws ClassNotFoundException,
            InstantiationException,
            IllegalAccessException {
        Class cls =
            this.getClass().getClassLoader().loadClass(thesaurus);
        this.thesaurus = (IThesaurus) cls.newInstance();
        this.stws = new TermToQueryScorer(this.thesaurus);
    }

    /**
     * @param numberOfWordsInTerm the maximum length of term to search for as a string
     */
    public void setNumberOfWordsInTerm(String
            numberOfWordsInTerm)
    {
        this.numberOfWordsInTerm =
            Integer.parseInt(numberOfWordsInTerm);
    }

    /**
     * @param thesaurus the thesaurus to use in the scoring
     * @param numberOfWordsPerTerm the maximum length of term to search for
     */
}
public void setStopWordListLocation(String stopWordListLocation) throws FileNotFoundException {
    StopWordStripper.initialize(stopWordListLocation);
}

public double score(String query, String text, int distanceToTravel, IWeighting weighting) {
    HashMap queryBaseTermsByScores = new HashMap();
    HashMap queryTermsByOccurence = new HashMap();
    HashMap queryTermsByBaseTermsGroups = new HashMap();
    int count = extractTermsFromQuery(query, queryBaseTermsByScores, queryTermsByOccurence,
          queryTermsByBaseTermsGroups);
    HashSet triedTextTerms = new HashSet();
    try // splits words into terms and passes them blocks to be searched for
    {
        StringBuffer[] sb = new StringBuffer[numberOfWordsInTerm];
        sb[0] = new StringBuffer();
        StringBuffer spaceBuffer = new StringBuffer();
        StringReader reader = new StringReader(text);
        int isNumber = 0; // 0 start state, 1 isn't, 2 is, 3 is double
        boolean justHadSpace = true;
        int words = 0;
        int c = -1;
        while ((c = reader.read()) != -1)
        {
            if (Character.isLetter(c))
            {
                justHadSpace = false;
                sb[words].append((char)c);
                isNumber = 1;
            }
            else if (Character.isDigit(c))
            {
                justHadSpace = true;
                sb[words].append((char)c);
                isNumber = 1;
            }
            else if (Character.isWhitespace(c))
            {
                isNumber = 0;
                if (justHadSpace)
                    continue;
                spaceBuffer.append(' '); 
                words++;
                if (words == numberOfWordsInTerm) 
                {
                    StringBuffer[] newsb = new StringBuffer[numberOfWordsInTerm];
                    StringBuffer newSpaceBuffer = new StringBuffer();
                    StringReader reader = new StringReader(text);
                    int termLength = numberOfWordsInTerm;
                    words = this.searchForTerm(sb, spaceBuffer, newsb, newSpaceBuffer, termLength, queryBaseTermsByScores, queryTermsByOccurence, queryTermsByBaseTermsGroups, weighting, triedTextTerms);
                    sb = newsb;
                    spaceBuffer = newSpaceBuffer;
                }
                sb[words] = new StringBuffer();
                justHadSpace = true;
            }
            switch(c)
            {
            case '-':
                {
                    justHadSpace = false;
                    sb[words].append((char)c);
                    isNumber = 2;
                }
            else if (Character.isWhitespace(c))
            {
                isNumber = 0;
                if (justHadSpace)
                    continue;
                spaceBuffer.append(' '); 
                words++;
                if (words == numberOfWordsInTerm) 
                {
                    StringBuffer[] newsb = new StringBuffer[numberOfWordsInTerm];
                    StringBuffer newSpaceBuffer = new StringBuffer();
                    StringReader reader = new StringReader(text);
                    int termLength = numberOfWordsInTerm;
                    words = this.searchForTerm(sb, spaceBuffer, newsb, newSpaceBuffer, termLength, queryBaseTermsByScores, queryTermsByOccurence, queryTermsByBaseTermsGroups, weighting, triedTextTerms);
                    sb = newsb;
                    spaceBuffer = newSpaceBuffer;
                }
                sb[words] = new StringBuffer();
                justHadSpace = true;
            }
            default:
```java
{ isNumber = 0;
  if (justHadSpace)
    continue;
  spaceBuffer.append((char)c);
  words++;
  if (words == numberOfWordsInTerm)
    { StringBuffer[] newsb = new
      StringBuffer[numberOfWordsInTerm];
      StringBuffer newSpaceBuffer = new
      StringBuffer();
      int termLength = numberOfWordsInTerm;
      words = this.searchForTerm(sb, spaceBuffer, newsb, newSpaceBuffer, words, queryBaseTermsByScores, distanceToTravel, weighting, triedTextTerms);
      sb = newsb;
      spaceBuffer = newSpaceBuffer;
      sb[words] = new StringBuffer();
      justHadSpace = true;
      break;
    }
  }
  //send of any remaining terms
  if (!justHadSpace)
    { words++;
      spaceBuffer.append('·');
    }
  while (words > 0)
    { StringBuffer[] newsb = new
      StringBuffer[numberOfWordsInTerm];
      StringBuffer newSpaceBuffer = new StringBuffer();
      words = this.searchForTerm(sb, spaceBuffer, newsb, newSpaceBuffer, words, queryBaseTermsByScores, distanceToTravel, weighting, triedTextTerms);
      sb = newsb;
      spaceBuffer = newSpaceBuffer;
    }
  }
  catch (IOException e)
    { e.printStackTrace();
      //calculate the score from the scores assigned to the queries
      double total = 0.0;
      Iterator it = queryTermsByBaseTermsGroups.keySet().iterator();
      while (it.hasNext())
        { Object term = it.next();
          Iterator baseterms = ((Collection)queryTermsByBaseTermsGroups.get(term)).iterator();
          double querymax = 0.0;
          while (baseterms.hasNext())
            { Object o = baseterms.next();
              SortedList slist = (SortedList)queryBaseTermsByScores.get(o);
              if (!slist.isEmpty())
                { double score = ((Double)slist.pop()).doubleValue();
                  querymax = Math.max(score, querymax);
                }
            }
          Integer occurrences = (Integer)queryTermsByOccurence.get(term);
          if (occurrences != null)
            { total += (querymax * occurrences.doubleValue());
            }
        }
      return Math.min(total / count, 1.0);
    }
/**
 * @param sb the array of StringBuffers each containing a word out of which to build terms to search for
 * @param spaceBuffer a StringBuffer containing the word dividers so that joining two terms is done correctly
 * @param newsb an array of StringBuffers to be filled with the words that didn't make part of the matched term
 * @param newSpaceBuffer an array of the word dividers that didn't make part of the matched term
 */
```
```java
private int searchForTerm(StringBuffer[] sb, StringBuffer spaceBuffer, int firstWord, int termLength, HashMap queryBaseTermsByScores, int distanceToTravel, IWeighting weighting, HashSet triedTextTerms)
{
    if (termLength == 1)
    {
        String term = sb[0].toString();
        if (StopWordStripper.isGoWord(term) && !triedTextTerms.contains(term))
            this.stws.score(term, queryBaseTermsByScores, distanceToTravel, weighting);
        return 0;
    }
    String term = putTogetherTerm(sb, spaceBuffer, 0, termLength);
    // search for term
    if (triedTextTerms.contains(term))
        return 0;
    if (this.thesaurus.doesExist(term))
    {
        this.stws.score(term, queryBaseTermsByScores, distanceToTravel, weighting);
        for (int lengthOfTerm = termLength - 1; lengthOfTerm > 1; lengthOfTerm --)
        {
            for (int i = 0; i < termLength - lengthOfTerm; i++)
            {
                term = this.putTogetherTerm(sb, spaceBuffer, i, i + lengthOfTerm);
                if (!triedTextTerms.contains(term))
                    this.stws.score(term, queryBaseTermsByScores, distanceToTravel, weighting);
            }
        }
    }
    for (int i = 0; i < termLength; i++)
    {
        term = sb[i].toString();
        if (StopWordStripper.isGoWord(term) && !triedTextTerms.contains(term))
            this.stws.score(term, queryBaseTermsByScores, distanceToTravel, weighting);
    }
    else
    {
        for (int i = numberOfWordsInTerm - 2; i >= 0; i --)
            // shift the words along to fit the new one in
        {
            if (newsb[i] != null)
                newsb[i+1] = newsb[i];
        }
        newsb[0] = sb[termLength - 1];
        newsBuffer.insert(0, spaceBuffer.charAt(termLength - 1));
        return 1 + this.searchForTerm(sb, spaceBuffer, newsBuffer, termLength - 1, queryBaseTermsByScores, distanceToTravel, weighting, triedTextTerms);
    }
    return 0;
}

/**
 * @param sb array of StringBuffer containing the words that need to be assembled into a single term string
 * @param spaceBuffer the dividers between the words in the sb argument used in assembling the term
 * @param firstWord the index of the first word in the term in the sb
 * @param termLength the length of the term
 * @return a single term assembled as directed
 */
private String putTogetherTerm(StringBuffer[] sb, StringBuffer spaceBuffer, int firstWord, int termLength)
{
    StringBuffer term = new StringBuffer();
    for (int i = firstWord; i < termLength; i++)
    {
        if (sb[i] != null)
private void append(StringBuffer sb, String s, int length) {
    term.append(sb[i]);
    term.append(spaceBuffer.charAt(i));
}

term.setLength(term.length() - 1);
return term.toString();

/**
 * @param query the query to have the terms extracted from it, multiword terms should be grouped by ""
 * @param queryBaseTermsByScores a HashMap to be filled with maps from the base forms of the query word to
 * SortedLists of their scores
 * @param queryTermsByOccurence a HashMap to be filled by maps from terms in the query to a count of how many
 * times they occur in the query
 * @param queryTermsByBaseTermsGroups a map from terms in the query to a hashset of the base term groups
 * formed from it
 * @return a count of the number of terms in the query
 **/
private int extractTermsFromQuery(String query, HashMap queryBaseTermsByScores, HashMap queryTermsByOccurence, HashMap queryTermsByBaseTermsGroups) {
    StreamTokenizer queryTokenizer = new StreamTokenizer(new StringReader(query));
    queryTokenizer.eolIsSignificant(false);
    queryTokenizer.lowerCaseMode(true);
    int tt, count = 0;
    try {
        while((tt = queryTokenizer.nextToken()) != StreamTokenizer.TT_EOF)
        {
            switch(tt)
            {
            case StreamTokenizer.TT_NUMBER:
            {
                String word =
                this.getNumberAsString(queryTokenizer.nval);
                count =
                this.addQueryWord(queryBaseTermsByScores, queryTermsByOccurence, queryTermsByBaseTermsGroups, count, word);
                break;
            }
            case StreamTokenizer.TT_WORD:
            {
                String word = queryTokenizer.sval;
                count =
                this.addQueryWord(queryBaseTermsByScores, queryTermsByOccurence, queryTermsByBaseTermsGroups, count, word);
                break;
            }
            default:
            {
            }
            }
        }
    } catch(IOException e) // shouldn't happen with a StringReader
    {
        e.printStackTrace();
    }
    return count;

/**
 * @param nval the number to be converted to a string
 * @return the string of the number as an integer if no accuracy is lost in doing so
 **/
private String getNumberAsString(double nval) {
    if(nval == (int)nval)
        return Integer.toString((int)nval);
    return Double.toString(nval);

/**
 * @param queryBaseTermsByScores a HashMap to be filled with maps from the base forms of the query word to
 * SortedLists of their scores
 * @param queryTermsByOccurence a HashMap to be filled by maps from terms in the query to a count of how many
 * times they occur in the query
 * @param queryTermsByBaseTermsGroups a map from terms in the query to a hashset of the base term groups
 * formed from it
 * @param count the number of terms in the previous parts of the query
 */
private int addQueryWord(HashMap queryBaseTermsByScores, HashMap queryTermsByOccurrence, HashMap queryTermsByBaseTermsGroups, int count, String word)
{
    if (queryTermsByOccurrence.containsKey(word)) // increment count of word
    {
        Integer occ = (Integer) queryTermsByOccurrence.get(word);
        queryTermsByOccurrence.put(word, new Integer(occ.intValue() + 1));
        count++;
    }
    else if (StopWordStripper.isGoWord(word))
    {
        HashSet groupOfBaseTerms = new HashSet();
        queryTermsByOccurrence.put(word, new Integer(1));
        Iterator baseforms = this.thesaurus.getBaseForms(word).iterator();
        while (baseforms.hasNext())
        {
            word = (String) baseforms.next();
            if (groupOfBaseTerms.add(word)) // true if word doesn't already exist
        }
    }
    queryBaseTermsByScores.put(word, new SortedList(new DoubleComparator()));
    queryTermsByBaseTermsGroups.put(word, groupOfBaseTerms);
    count++;
    return count;
}

public class DoubleComparator implements Comparator
{
    public int compare(Object o1, Object o2)
    {
        /* (non-Javadoc)
        * @see java.util.Comparator#compare(T, T)
        */
        return (int)(((Double)o1).doubleValue() - ((Double)o2).doubleValue())*10.0);
    }
}

A.2 Trust Network Matchmaker

A.2.1 File: SingleDepthTidalTrustScorer.java

package uk.ac.bath.mmm.TidalTrust;

import java.io.IOException;
import java.io.InputStream;
import java.net.ConnectException;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.Iterator;
import com.hp.hpl.jena.query.Query;
import com.hp.hpl.jena.query.QueryExecution;
import com.hp.hpl.jena.query.QueryExecutionFactory;
import com.hp.hpl.jena.query.QuerySolution;
import com.hp.hpl.jena.query.ResultSet;
import com.hp.hpl.jena.rdf.model.Model;
/**
 * @author nick
 * Based on the Single Depth algorithm outlined in the dissertation
 * accompanying this code. See the description in the dissertation for
 * explanation of the algorithm used here.
 * Class which implements an interpretation of the TidalTrust algorithm by Jennifer Golbeck.
 * The algorithm is slightly extended to support depth cutoffs to prevent searches taking too long.
 */

public class SingleDepthTidalTrustScorer extends uk.ac.bath.mmm.AbstractNetworkScorer {
    private String trustSubject;

    /**
     * @param trustSubject the subject to use to judge trust between users on
     */
    public SingleDepthTidalTrustScorer(String trustSubject) {
        this.trustSubject = trustSubject;
    }

    /**
     * Default constructor — fields can be set with set_* methods
     */
    public SingleDepthTidalTrustScorer() {
    }

    /**
     * @param trustSubject the subject over which to infer trust over
     */
    public void set_trustSubject(String trustSubject) {
        this.trustSubject = trustSubject;
    }

    /* (non-Javadoc)
     * @see uk.ac.bath.mmm.AbstractNetworkScorer#score(java.lang.String, java.lang.String, int)
     */

    public double score(String userFoaUrl, String webServiceIDUrl, int maxDepth) {
        HashSet found = new HashSet();
        found.add(userFoaUrl);
        ArrayList nodesAtCurrentDepth = new ArrayList();
        NodeF source = new NodeF(userFoaUrl);
        NodeF sink = new NodeF(webServiceIDUrl);
        nodesAtCurrentDepth.add(source);
        ArrayList[] nodesAtGivenDepth = new ArrayList[maxDepth];
        int depth = 1;
        nodesAtGivenDepth[0] = new ArrayList();
        while (!nodesAtCurrentDepth.isEmpty() && depth <= maxDepth) {
            ArrayList nodesAtNextDepth = new ArrayList();
            NodeF nf = (NodeF) nodesAtCurrentDepth.remove(0);
            nodesAtGivenDepth[depth - 1].add(nf);
            try {
                Model model = this.generateModel(nf);
                double score = this.lookForReview(nf.getFoaUrl(),
                    webServiceIDUrl, model);
                if (score != -1) // sink in adj(n)
                    if (nf == source)
                        return score / 10.0;
                    nf.setCachedRating(sink, score);
                    maxDepth = depth;
                    this.setPathFlow(nf, (int)score, sink);
                    nf.addChild(sink, (int)score);
            } catch (Exception e) {
                System.err.println(e.toString());
            } else if (depth < maxDepth) {
                ArrayList friendUrls = new ArrayList();
                ArrayList friendRatings = new ArrayList();
                this.findFriends(nf.getFoaUrl(), model, friendUrls, friendRatings);
                for (int i = 0; i < friendUrls.size(); i++) {
                    Object friendUrl = friendUrls.get(i);
                    if (!found.contains(friendUrl)) {
                        found.add(friendUrl);
                        nodesAtNextDepth.add(new NodeF((String)friendUrl));
                    }
                }
            }
        }
    }
}
NodeF friend = this.findNode(nodesAtNextDepth, friendUrl);
if (friend != null) {
    int rating = ((Integer)friendRatings.get(i)).intValue();
    this.setPathFlow(nf, rating, friend);
    nf.addChild(friend, rating);
}
}
}
}
try {
    this.findNode(nodesAtNextDepth, friendUrl);
    if (friend != null) {
        int rating = ((Integer)friendRatings.get(i)).intValue();
        this.setPathFlow(nf, rating, friend);
    }
}
}
}
}
catch (ConnectException e) {
    System.err.println("Timed out while connecting to: " + nf.getFoafUrl());
}
}
catch (MalformedURLException e) {
    e.printStackTrace();
}
catch (IOException e) {
    e.printStackTrace();
}
if (nodesAtCurrentDepth.isEmpty()) {
    nodesAtCurrentDepth = nodesAtNextDepth;
    depth++;
    if (depth <= maxDepth && nodesAtGivenDepth[depth - 1] == null)
        nodesAtGivenDepth[depth - 1] = new ArrayList();
}
{return calculateOpinions(source, sink, nodesAtGivenDepth, depth);}

protected double calculateOpinions(NodeF source, NodeF sink, ArrayList[] nodesAtGivenDepth, int depth)
{
    int max = sink.getFlow();
    if (max >= 0) {
        for (depth = depth - 2; depth >= 0; depth--)
            {
                while (!nodesAtGivenDepth[depth].isEmpty())
                {
                    NodeF n = (NodeF)nodesAtGivenDepth[depth].remove(0);
                    int noOfChildren = n.getNumChildren();
                    double numerator = 0;
                    double denominator = 0;
                    for (int i = 0; i < noOfChildren; i++)
                        {
                            int rating = n.getRating(i);
                            double cachedRating = n.getRating(i).getRating(sink);
                            if (rating >= max && cachedRating >= 0)
                                {
                                    numerator += rating * cachedRating;
                                    denominator += rating;
                                }
                        }
    } double cR = source.getCachedRating(sink) / 10;
    if (cR >= 0)
        return cR;
    return 0.5;
}
/*
 @param source the node to infer the opinion for
 @param sink the web service to infer of
 @param nodesAtGivenDepth an array list of all the nodes at the depths they were encountered from the source user
 @param depth the depth at which the web service was found
 @return a score between 0 and 1
*/
protected void setPathFlow(NodeF nf, int rating, NodeF sink)
```java
{ int flow = Math.min(nf.getPathFlow() == NodeF.UNDEFINED ? Integer.MAX_VALUE : nf.getPathFlow(), rating); sink.setPathFlow(Math.max(sink.getPathFlow(), flow)); }

/**
 * @param nf the node to generate the model for
 * @return the model generated for the given node
 * @throws IOException
 * @throws MalformedURLException
 */
protected Model generateModel(NodeF nf) throws IOException, MalformedURLException {
    String foafUrl = nf.getFoafUrl();
    InputStream in = new URL(foafUrl).openStream();
    Model model = ModelFactory.createMemModelMaker().createFreshModel();
    model.read(in, this.getBaseURI(foafUrl));
    in.close(); // fill out model
    HashSet urls = new HashSet();
    urls.add(foafUrl);
    model = this.completeUserModel(model, foafUrl, urls);
    Iterator it = urls.iterator();
    while (it.hasNext()) {
        nf.addFoafUrl((String)it.next());
    }
    return model;
}

/**
 * @param nodesAtNextDepth the list within which to search for the friend
 * @param friendUrl the url to identify the friend by
 * @return the NodeF representing the friend with the given URL or null
 */
protected NodeF findNode(ArrayList nodesAtNextDepth, Object friendUrl) {
    Iterator it = nodesAtNextDepth.iterator();
    while (it.hasNext()) {
        NodeF nf = (NodeF)it.next();
        Iterator urls = nf.getFoafUrls();
        while (urls.hasNext()) {
            if (urls.next().equals(friendUrl))
                return nf;
        }
    }
    return null;
}

/**
 * @param userFoafUrl the URI of the user to find the friends of
 * @param model the model containing the information about who the user's friends are
 * @param urls an ArrayList to fill with the urls of the found friends
 * @param ratings an ArrayList to fill with the trust ratings of the found friends, in the same order as the found friends
 */
protected void findFriends(String userFoafUrl, Model model, ArrayList urls, ArrayList ratings) {
    String queryString3 = "SELECT ?friend ?trustValue WHERE {
    " + userFoafUrl + " <http://trust.mindswap.org/ont/trust.owl#trustsRegarding> ?x + " + "<http://trust.mindswap.org/ont/trust.owl#trustSubject> <http://trust.mindswap.org/ont/trust.owl#trustedPerson> ?friend" + // indicates the subject is the same as the previous
    "http://trust.mindswap.org/ont/trust.owl#trustValue> ?trustValue.";  
    Query query3 = QueryFactory.create(queryString3);
    QueryExecution qe3 = QueryExecutionFactory.create(query3, model);
    ResultSet results3 = qe3.execSelect();
    while (results3.hasNext()) // read list of friends into a list
    { 
        QuerySolution qs = results3.nextSolution();
        if (qs.get("friend").isURIResource() &&
            qs.get("trustValue").isLiteral())
        {
            urls.add(qs.getResource("friend").toString());
        }
    }
}
```
APPENDIX A. CODE

```java
ratings.add(new 
    Integer(qs.getLiteral("trustValue").getInt()));

} 
}

/**
 * @param userFoafUrl the URI of the user to find the friends of
 * @param webServiceIDUrl the identifier of the web service (or other item) to find the review about
 * @param model the model containing the information about who the user has rated
 */

protected double lookForReview(String userFoafUrl, String webServiceIDUrl, Model model) {
    String queryString = "SELECT?score WHERE { <" + userFoafUrl + "><http://xmlns.com/foaf/0.1/made>" + 
        "_?thingmade " + 
        "_?thingmade" + 
        "http://www.purl.org/stuff/rev#made" + 
        "?thingmade" + 
        " http://www.purl.org/stuff/rev#made" + 
        " http://www.purl.org/stuff/rev#made" + 
        "<http://www.purl.org/stuff/rev#rating>" + "?score" + 
        " <http://www.purl.org/stuff/rev#rating>" + "?score" + 
        "<http://www.purl.org/stuff/rev#rating>" + "?score" + 
        "<http://www.purl.org/stuff/rev#rating>" + "?score" + 
        "<http://www.purl.org/stuff/rev#rating>" + "?score" + 
        "<http://www.purl.org/stuff/rev#rating>" + "?score" + 
        "<http://www.purl.org/stuff/rev#rating>" + "?score" + 
} ORDER BY DESC(?date) " ;
    Query query = QueryFactory.create(queryString);
    QueryExecution qe = QueryExecutionFactory.create(query, model);
    ResultSet results = qe.execSelect();
    if (results.hasNext()) // the review exists, rejoice
        {
            QuerySolution qs = results.nextSolution();
            if (qs.get("score").isLiteral())
                return qs.getLiteral("score").getDouble();
        }
    return -1;
}

/** (non-Javadoc)
 * @see java.lang.Object#toString()
 */

public String toString() {
    return "SDTTS";
}

A.2.2 File: MultipleDepthTidalTrustScorer.java

```
APPENDIX A. CODE

* explanation of the algorithm used here.
* Class which implements an interpretation of the TidalTrust algorithm
* by Jennifer Golbeck.
* The algorithm is slightly extended to support depth cutoffs to prevent
* searches taking too long.
*/
public class MultipleDepthTidalTrustScorer extends
    uk.ac.bath.mmm.AbstractNetworkScorer {
  private String trustSubject;
  protected ScoreCalculator scoreCalculator;
}

/**
 * @param trustSubject the subject to use to judge trust between users on
 * @param sc the score calculator to compose the scores with
 */
public MultipleDepthTidalTrustScorer(String trustSubject, ScoreCalculator sc) {
  this.trustSubject = trustSubject;
  this.scoreCalculator = sc;
}

/**
 * Default constructor — fields can be set with set__ methods
 */
public MultipleDepthTidalTrustScorer() {
}

/**
 * @param trustSubject the subject over which to infer trust over
 */
public void setTrustedSubject(String trustSubject) {
  this.trustSubject = trustSubject;
}

/**
 * @param sc the score calculator class to compose the scores with
 * @throws InstantiationException
 * @throws IllegalAccessException
 */
public void setScoreCalculator(String scoreCalculator)
    throws InstantiationException, IllegalAccessException
{
  Class class =
      this.getClass().getClassLoader().loadClass(scoreCalculator);
  this.scoreCalculator =
      (ScoreCalculator) class.newInstance();
}

/**
 * @param userFoafUrl
 * @param webServiceIDUrl
 * @param maxDepth
 */
public double score(String userFoafUrl, String webServiceIDUrl, int maxDepth) {
  Hashset found = new HashSet();
  found.add(userFoafUrl);
  ArrayList nodesAtCurrentDepth = new ArrayList();
  Node node = new NodeF(userFoafUrl);
  HashMap sinks = new HashMap();
  nodesAtCurrentDepth.add(node);
  while (!nodesAtCurrentDepth.isEmpty() && depth <= maxDepth)
  {
    ArrayList nodesAtNextDepth = new ArrayList();
    NodeF nf = (NodeF) nodesAtCurrentDepth.remove(0);
    sinks.put(nf, new double[]{score, depth});
    try
    {
      Model model = this.generateModel(nf);
      double score = this.lookForReview(nf, getFoafUrl(),
          webServiceIDUrl, model);
      if (score != -1) //sink is n because adj n is review
      {
        if (nf == source)
          return score / 10.0;
        sinks.put(nf, new double[]{score, depth});
      }
      ArrayList friendUrls = new ArrayList();
    }
  }
  return score;
}

```java
ArrayList friendRatings = new ArrayList();
this.findFriends(nf.getUrl(), model, friendUrls, friendRatings);
for (int i = 0; i < friendUrls.size(); i++)
{
    Object friendUrl = friendUrls.get(i);
    if (!found.contains(friendUrl))
    {
        found.add(friendUrl);
        nodesAtNextDepth.add(new NodeF((String) friendUrl));
    }
    NodeF friend = this.findNode(nodesAtNextDepth, friendUrl);
    if (friend != null)
    {
        int rating =
            (Integer) friendRatings.get(i).intValue();
        this.setPathFlow(nf, rating, friend);
        nf.addChild(friend, rating);
        nf.setCachedRating(friend, rating);
    }
}
catch (ConnectException e)
{
    System.err.println("Timed out while connecting to:");
    nf.getUrl();
}
catch (MalformedURLException e)
{
    e.printStackTrace();
}
catch (IOException e)
{
    e.printStackTrace();
}
if (nodesAtCurrentDepth.isEmpty())
{
    nodesAtCurrentDepth = nodesAtNextDepth;
    depth++;
    if (depth <= maxDepth && nodesAtGivenDepth[depth - 1] != null)
    {
        nodesAtGivenDepth[depth - 1] = new ArrayList();
    }
}
return calculateOpinions(source, sinks, nodesAtGivenDepth);

/**
 * @param source the source node to infer the opinion of
 * @param sinks the list of reviews to infer the opinion from mapped to pairs of scores and depth
 * @param nodesAtGivenDepth an array list of all the nodes at the depths they were encountered from the source user
 * @return a score between 0 and 1
 */
protected double calculateOpinions(NodeF source, HashMap sinks, ArrayList[] nodesAtGivenDepth)
{
    if (sinks.isEmpty())
    {
        return 0.5;
    }
    Iterator it = sinks.keySet().iterator();
    double[] trustValues = new double[sinks.size()];
    double[] scores = new double[sinks.size()];
    for (int s = 0; it.hasNext(); s++)
    {
        NodeF sink = (NodeF) it.next();
        int max = sink.getPathFlow();
        if (max >= 0)
        {
            double[] sinkInfo = (double[]) sinks.get(sink);
            scores[s] = sinkInfo[0];
            int depth = (int) sinkInfo[1];
            for (depth = depth - 3; depth >= 0; depth--) // one back for the reviewers, another two for their grandparents
            {
                for (int ndex = 0; ndex < nodesAtGivenDepth[depth].size(); ndex++)
                {
                    NodeF n = (NodeF) nodesAtGivenDepth[depth].get(ndex);
                    int numberOfChildren = n.getNumberOfChildren();
                    double numerator = 0;
                    double denominator = 0;
                    for (int i = 0; i < numberOfChildren; i++)
                    {
                        int rating = n.getChildrenRating(i);
                        double cachedRating = n.GetChild(i).get CachedRating(sink);
                        if (rating >= max && cachedRating >= 0)
                        {
                            numerator += rating * cachedRating;
                        }
                    }
                    scores[s] += numerator / denominator;
                }
            }
        }
    }
    return scores[sinks.size() - 1] / sinks.size();
}
```
denominator += rating;
}
}  
if (denominator > 0)  
n.setCachedRating(sink, numerator / denominator);  
}  
}  
if (source.getCachedRating(sink) >= 0)  
trustValues[s] = source.getCachedRating(sink) / 10.0;  
else  
trustValues[s] = 0;
}  
else  
{  
trustValues[s] = 0;  
scores[s] = 0;
}
}  
return this.scoreCalculator.calculateScore(trustValues, scores);

/**
 * @param nf the node that trusts/rates the sink the given amount
 * @param rating the amount of trust/rates the sink
 * @param sink the node - user/reviewed item - that the
 * trusts/rates
 * @Part of the TidalTrust algorithm
 */
protected void setPathFlow(NodeF nf, int rating, NodeF sink)
{
    int flow = Math.min(nf.getPathFlow() == NodeF.UNDEFINED
    ? Integer.MAX_VALUE : nf.getPathFlow(), rating);  
sink.setPathFlow(Math.max(sink.getPathFlow(), flow));
}

/**
 * @param nf the node to generate the model for
 * @return the model generated for the given node
 * @throws IOException
 * @throws MalformedURLException
 */
protected Model generateModel(NodeF nf) throws IOException, MalformedURLException
{  
String foafUrl = nf.getFoafUrl();  
InputStream in = new URL(foafUrl).openStream();  
Model model = ModelFactory.createMemModelMaker().createFreshModel();  
model.read(in, this.getBaseURI(foafUrl));  
in.close();  
// fill out model  
HashSet urls = new HashSet();  
urls.add(foafUrl);  
model = this.completeUserModel(model, foafUrl, urls);  
Iterator it = urls.iterator();  
while (it.hasNext())  
    nf.addFoafUrl((String)it.next());  
return model;

/**
 * @param nodesAtNextDepth the list within which to search for the friend
 * @param friendUrl the url to identify the friend by
 * @return the NodeF representing the friend with the given URL or null
 */
protected NodeF findNode(ArrayList nodesAtNextDepth, Object friendUrl)
{
    Iterator it = nodesAtNextDepth.iterator();  
    while (it.hasNext())
    {
        NodeF nf = (NodeF)it.next();  
        Iterator urls = nf.getFoafUrls();  
        while (urls.hasNext())
        {
            if (urls.next().equals(friendUrl))
                return nf;
        }  
    }  
    return null;
}

/**
 * @param userFoafUrl the URI of the user to find the friends of
 * @param model the model containing the information about who the user's friends are
 * @param urls an ArrayList to fill with the urls of the found friends
 */
protected void findFriends(String userFoafUrl, Model model, ArrayList urls, ArrayList ratings)
{
    String queryString3 = "SELECT ?friend ?trustValue WHERE {
        <" + userFoafUrl + ">
        <http://trust.mindswap.org/ont/trust.owl#trustsRegarding> " + userFoafUrl + "<http://trust.mindswap.org/ont/trust.owl#trustSubject> " + this.trustSubject + "<http://trust.mindswap.org/ont/trust.owl#trustedPerson> " + ?friend + "//indicates the subject is the same as the previous " + <http://trust.mindswap.org/ont/trust.owl#trustValue> " + ?trustValue"");
    Query query3 = QueryFactory.create(queryString3);
    QueryExecution qe3 = QueryExecutionFactory.create(query3, model);
    ResultSet results3 = qe3.execSelect();
    while(results3.hasNext()) //read list of friends into a list
    {
        QuerySolution qs = results3.nextSolution();
        if(qs.get("friend").isURIResource() && qs.get("trustValue").isLiteral())
        {
            urls.add(qs.getResource("friend").toString());
            ratings.add(new Integer(qs.getLiteral("trustValue").getInt()));
        }
    }
}

protected double lookForReview(String userFoafUrl, String webServiceIDUrl, Model model)
{
    String queryString = "SELECT ?score WHERE {
        <http://xmlns.com/foaf/0.1/made> + userFoafUrl + ">
        <http://xmlns.com/foaf/0.1/made> + this.trustSubject + "" + <http://trust.mindswap.org/ont/trust.owl#trustedPerson> " + ?friend + "//indicates the subject is the same as the previous " + <http://trust.mindswap.org/ont/trust.owl#trustValue> " + ?trustValue"");
    Query query = QueryFactory.create(queryString);
    QueryExecution qe = QueryExecutionFactory.create(query, model);
    ResultSet results = qe.execSelect();
    if(results.hasNext()) //the review exists, rejoice
    {
        QuerySolution qs = results.nextSolution();
        if(qs.get("score").isLiteral())
        {
            return qs.getLiteral("score").getDouble();
        }
    }
    return -1;
}

public String toString()
{ return "MDTTS" + this.scoreCalculator.toString();
}