RELEVANCE MAPPING OF WIKIPEDIA EDITS
USING SEMANTIC WEB
CONCEPTS
by
Poonam Gohil

An Abstract
of a thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
in the School of Computer Science and Mathematics
University of Central Missouri
April, 2017
ABSTRACT

By

Poonam Gohil

The Semantic Web gives a well-defined meaning to the information that enables computers and people to work efficiently in cooperation [1]. It provides enhanced information access based on the machine-processable meta-data. It is an exciting new type of hierarchy and standardization that will replace the current "web of links" with a "web of meaning" [2].

Wikipedia is a massive free online encyclopedia with the concept of "wiki". Wikipedia lets everyone in. Anyone with an Internet connection can become an editor, which increases the frequency of vandalism. Due to the volume of edits, it is crucial to examine the nature of these progressions and help keep up the trustworthiness of Wikipedia articles.

In this thesis, an effective way to categorize the nature of these changes is presented. Both simple and ontology-based web crawlers were studied. Use of ontological hierarchy, cognitive synonyms or synsets and concept based ranking helped in the calculation of relevance of edits [3]. A shallow, cross-domain ontology, called DBpedia, which has been manually created based on the most commonly used infoboxes within Wikipedia, was chosen for this study.

This empirical study is focused on calculating the relevance of a Wikipedia edit by crawling the web while keeping into consideration the metadata of the edit, ontological hierarchy of the article, and synonyms of the extracted keyword. The goal of this thesis is to increase the accuracy of the information provided by Wikipedia and make it more reliable by following a multithreaded ontological approach to model a domain of interest that guides the crawler to the relevant information on the semantic web.
RELEVANCE MAPPING OF WIKIPEDIA EDITS

USING SEMANTIC WEB

CONCEPTS

by

Poonam Gohil

A Thesis
presented in partial fulfillment
of the requirements for the degree of
Master of Science
in the School of Computer Science and Mathematics
University of Central Missouri

April, 2017
RELEVANCE MAPPING OF WIKIPEDIA EDITS

USING SEMANTIC WEB

CONCEPTS

by

Poonam Gohil

APRIL 2017

APPROVED

Thesis Chair: Dr. Sumi Singh

Thesis Committee Member: Dr. Mahmoud Yousef

Thesis Committee Member: Dr. Anshuman Singh

ACCEPTED:

Chair, School of Computer Science and Mathematics: Dr. Xiaodong Yue

UNIVERSITY OF CENTRAL MISSOURI

WARRENSBURG, MISSOURI
Acknowledgment

I would like to thank Dr. Sumi Singh for her guidance, encouragement, and patience over the last year and a half. Your help in every aspect of my research was imperative to my completion of this degree. Thank you for providing indispensable advice, information and support.

I would also like to thank my thesis committee, Dr. Mahmoud Yousef and Dr. Anshuman Singh, for their time and helpful guidance.
Table of Contents

List of Figures .................................................................................................................. ix

List of Tables .................................................................................................................... x

Chapter 1: Introduction and Background .......................................................................... 1

1.1 Introduction .................................................................................................................. 1

1.2 Motivation ..................................................................................................................... 2

1.3 Background .................................................................................................................. 2

   1.3.1 Natural Language Processing for Wikipedia Quality Measure ......................... 4

   1.3.2 Machine Learning for Wikipedia Quality Measure ........................................... 5

1.4 DBpedia as Semantic Web Representation of Wikipedia Articles ............................. 6

Chapter 2: Design of Wikipedia Edit Analysis System ...................................................... 7

2.1 Introduction .................................................................................................................. 7

2.2 Phase 1 ......................................................................................................................... 7

   2.2.1 Fetching edits via web socket ............................................................................. 7

   2.2.2 Analyze the metadata of the fetched edits ...................................................... 8

   2.2.3 Extract URL and title from metadata ................................................................ 10

2.3 Phase 2 ......................................................................................................................... 10

   2.3.1 Visit URL to extract keywords ....................................................................... 10

   2.3.2 Stoplisting ......................................................................................................... 10

   Table 2:2 Importance of stoplisting ........................................................................... 11

   2.3.3 Stemming ......................................................................................................... 11

2.4 Phase 3 ......................................................................................................................... 12
2.4.1 Generating a search query ................................................................. 12
2.4.2 Initialize DBpedia Ontology ............................................................. 13
2.4.3 Search WordNet for synsets ............................................................... 15
2.4.4 Crawling the Web using Multithreading .......................................... 17
2.5 Phase 4 ..................................................................................................... 22
2.5.1 Fetching .............................................................................................. 22
2.5.2 Preprocessing .................................................................................... 22
2.5.3 Page Weight Calculation ................................................................. 23
2.5.4 Relevance Mapping of Wikipedia edit ............................................ 27
2.6 Conclusion ............................................................................................. 28

Chapter 3: Evaluation of Wikipedia Edit Analysis System............................... 30
3.1 Introduction ............................................................................................ 30
3.2 Results .................................................................................................... 31
  3.2.1 Crawler Performance .................................................................... 31
  3.2.2 Comparative efficiency ................................................................. 33
  3.2.3 Evaluation of the Proposed System with respect to relevance mapping .............................................................................. 35
3.3 Conclusion ............................................................................................. 38

Chapter 4: Conclusions and Future Work .......................................................... 39
Appendix 1: System Verified Relevance Mapping .......................................... 41
Appendix 2: User Verified Relevance Mapping ............................................ 53
Bibliography .................................................................................................. 57
List of Figures

Figure 2:1 Proposed system for relevance mapping ................................................................. 8
Figure 2.2 Fetching edits via web-socket .................................................................................. 8
Figure 2:3 Simple Concept Ontological Hierarchy .................................................................. 14
Figure 2:4 Hyponyms and Hypernyms .................................................................................... 17
Figure 2:5 Execution of multiple threads in a single process ..................................................... 18
Figure 2:6 Multithreaded Architecture ..................................................................................... 19
Figure 2:7 Flow of the designed multithreaded web crawler ..................................................... 21
Figure 2:8 URL Frontier Queue .............................................................................................. 20
Figure 2:9 Process Cycle of a single thread ............................................................................. 23
Figure 2:10 Structure of the list of keywords .......................................................................... 23
Figure 2:11 A process of crawling and fetching of the URLs ................................................... 28
Figure 3:1: The CPU time in milliseconds needed to process a single page by a multithreaded
  web crawler .......................................................................................................................... 31
Figure 3:2 The CPU time in milliseconds needed to process a single page by a single threaded
  web crawler .......................................................................................................................... 32
Figure 3:3 The harvest ratio of a crawler using ontology v/s a crawler using only a keyword ... 34
Figure 3:4 Recall of the proposed system ................................................................................. 36
Figure 3:5 Precision of the proposed system ............................................................................ 37
Figure 3:6 F-Score of the proposed system ............................................................................. 38
List of Tables

Table 2:1 Metadata of an edited article fetched via web socket .......................................................... 9
Table 2:2 Importance of stoplisting ................................................................................................... 11
Table 2:3 Stemming example ............................................................................................................ 12
Table 2:4 JSoup Search query .......................................................................................................... 12
Table 2:5 Sample code for creating, loading, indexing and querying a local TDB store .............. 15
Table 2:6 Semantic Relations in WordNet ...................................................................................... 16
Chapter 1: Introduction and Background

1.1 Introduction

The Semantic Web gives a well-defined meaning to the information that enables computers and people to work efficiently in cooperation [1]. It provides enhanced information access based on the machine-processable meta-data. The Semantic Web presents an exciting new type of hierarchy and standardization that will replace the current "web of links" with a "web of meaning" [2]. Semantic Web combines aspects of artificial intelligence, markup languages, natural language processing, information retrieval, knowledge representation, intelligent agents, and databases.

Ontologies are key enabling technology for the Semantic Web. Ontologies offer a way to cope with heterogeneous representations of web resources. The domain model implicit in an ontology can be taken as a unifying structure for giving information a common representation and semantics [4]. Ontological analysis clarifies the structure of knowledge. Given a domain, its ontology forms the heart of any system of knowledge representation for that domain. Without ontologies, or the conceptualizations that underlie the knowledge, there cannot be a vocabulary for representing that knowledge [5]. Thus, the first step in devising an effective knowledge representation system, and vocabulary, is to perform an effective ontological analysis of the field, or domain. Weak analyses lead to incoherent knowledge bases.

In their paper, [6] tested and verified that the ontology contributes to the efficiency improvement in terms of the relevance of retrieved documents. Ontologies help to improve recall and precision. Recall is defined as the number of relevant documents retrieved by the system divided by the total number of relevant documents in the database, and precision is the ratio of
documents retrieved by the system that are relevant to the query divided by the total number of documents retrieved.

In this work we focus on analysis of edits made on Wikipedia articles. As one is aware of, a Wikipedia edits may not be very accurate. With respect to Wikipedia articles, the lack of accuracy is generalized by the term vandalism. In 2005, Wikipedia started to require those who create new articles to have a registered account to fight vandalism.

1.2 Motivation

Wikipedia is a massive free online encyclopedia with the concept of "wiki" (a website that provides collaborative modification of its content and structure directly from the web browser.). Wikipedia is the world's fifth-most-popular website in terms of overall visitor traffic as per Alexa Internet [7]. It contains around 5,377,348 articles [8] in English and receives 18 billion page views and nearly 500 million unique visitors each month [8]. It’s the crown jewel of crowdsourcing and dominates through sheer ubiquity. The key to that success is hardly a secret. Wikipedia lets everyone in. Anyone with an Internet connection can become an editor, which increases the frequency of vandalism [9]. Most edits are minor, like spelling and grammar checks, while others are major, such as change of information and addition or deletion of facts. Due to the volume of edits, it is crucial to examine the nature of these progressions and help keep up the trustworthiness of Wikipedia articles.

1.3 Background

Vandalism is any addition, removal, or change of content, in a deliberate attempt to damage Wikipedia. Examples of typical vandalism are adding irrelevant obscenities and crude humor to a page, illegitimately blanking pages, and inserting obvious nonsense into a page.
Vandalism can be categorized as follows:

1. Blanking - Removing all or significant parts of a page’s content without any reason.
2. Edit summary vandalism - Making offensive edit summaries to leave a mark that cannot be easily expunged from the record.
3. Silly vandalism - Adding profanity, graffiti or patent nonsense to pages.
4. Template vandalism - Modifying the wiki language or text of a template in a harmful or disruptive manner.
5. Page lengthening - Adding very large amounts of content to a page to make the page’s load time abnormally long.

Wikipedia takes several measures to reduce as well as eliminate vandalism. Some of the most common steps have been discussed below.

- Using Wikipedia's history functionality, which retains all prior versions of an article, to restore the article to the last version before the vandalism occurred; this is called reverting vandalism. Most vandalism on Wikipedia is reverted quickly. There are various ways in which the vandalism gets detected so it can be reverted:
  - **Bots**: In many cases, the vandalism is automatically detected and reverted by a bot. The vandal is always warned with no human intervention.
  - **Recent changes patrol**: Wikipedia has a special page that lists all the most recent changes. Some editors will monitor these changes for possible vandalism.
  - **Watchlists**: Any registered user can watch a page that they have created or edited or that they otherwise have an interest in. This functionality also enables users to monitor a page for vandalism.
Incidental discovery: Any reader who comes across vandalism by chance can revert it. In 2008, it was reported that the rarity of such incidental discovery indicated the efficacy of the other methods of vandalism removal.

- Locking articles so only established users, or in some cases only administrators, can edit them.
- Blocking and banning those who have repeatedly committed acts of vandalism from editing for a period or in some cases, indefinitely.
- The "abuse filter" extension, which uses regular expressions to detect common vandalism terms.

Wikipedia quality control can be in the form of vandalism detection [10], error detection, or by ensuring the information stays updated. This process can be done in real time or in batch. The process of automatic quality check of Wikipedia articles can be done divided onto two main classes of methods. One class uses natural language processing for quality assurance, while the other class uses machine-learning concepts.

1.3.1 Natural Language Processing for Wikipedia Quality Measure

Wikipedia like any other semantic web, uses natural language processing methods for quality control of information presented in the articles. Natural language processing methods use plain text aspects of Wikipedia edits to assess the quality of the edits and may not be very sophisticated in error or vandalism detection [11]. To improve detection accuracy, in addition to natural language features, coordination features have been used. The underlying assumptions behind methods combining natural language with coordination features are that the Wikipedia quality can be maintained and improved based on the coordinated efforts of its extremely
dedicated contributors. There may be an argument made that the increase in number of contributors may help drastically improve the quality of Wikipedia articles. Many studies have been conducted that assessed the coordination-based strategy to improve Wikipedia article quality and relationship between the number of editors, per article. One of the studies also analyzed the effectiveness of a specific coordination strategy and found that implicit strategies work better than the explicit strategies [12]. Another study argued that coordination based Wikipedia quality measurement must be performed at both global and local levels. The global measure analyzes the effect of contribution at the level of entire Wikipedia. The local measure restricts itself to an article level. It was found that overall quality was improved if the focus was made at the global level. At the local level, any article specific inequality has been found to have an indirect effect on the quality and is easy to improve [13].

1.3.2 Machine Learning for Wikipedia Quality Measure

In their paper [14], machine learning is used to create classifiers to assess the actionable quality of articles. The classifier is based on features representing variations in actionable quality of an article. Automatic Machine Learning-based vandalism detection using Naïve Bayes and probabilistic sequence mapping classifiers have been studied widely. These methods combine natural language processing and machine learning and uses bag of words as it’s features to classify edits [15]. Another machine learning based vandalism detection method uses data from Wikitrust as features [16]. This method provides an estimate of Vandalism with every Wikipedia revision. This estimate can be used to alter the snapshot of Wikipedia available to the users at various settings. For example, in low tolerance settings the static snapshot of low vandalism revision can be accessed, while in settings where the updates are important, or where the tolerance to vandalism is higher, revision with higher vandalism estimate can be accessed [16]. A
spatial-temporal feature extracted from metadata of each edit has been shown to be an effective method for vandalism detection. These features are used to classify edits and have been shown to be as effective as vandalism detection that use natural language processing [17].

1.4 DBpedia as Semantic Web Representation of Wikipedia Articles

DBpedia represents millions of English Wikipedia articles as RDF (Resource Description Framework) statements [18]. DBpedia is based on semantic web standards that allows for creation and or extension of ontologies [19]. DBpedia helps in extracting structured information from Wikipedia. It allows the user to enter complex queries as input. The outputs of these queries are interpretable by humans as well as machines [20]. DBpedia Mobile [21] [22] enabled the exploration of geospatial semantic web. It was further expanded to Wikidata [23].

In this thesis, an effective way to categorize the nature of changes in Wikipedia article is presented. Both simple and ontology-based web crawlers have been studied. Use of ontological hierarchy, cognitive synonyms or synsets and concept based ranking helped in the calculation of relevance of edits. A shallow, cross-domain ontology, called DBpedia, which has been manually created based on the most commonly used infoboxes within Wikipedia, was chosen for this study. An infobox represents a summary of information about the subject of a Wikipedia article.

This empirical study is focused on calculating the relevance of a Wikipedia edit by crawling the web while keeping into consideration the metadata of the edit, ontological hierarchy of the article, and synonyms of the extracted keyword. The goal of this thesis is to increase the accuracy of the information provided by Wikipedia and make it more reliable by following a multithreaded ontological approach to model a domain of interest that guides the crawler to the relevant information on the semantic web.
Chapter 2: Design of Wikipedia Edit Analysis System

2.1 Introduction

In this chapter, a semantic way of calculating relevance of Wikipedia edits is presented. This process consists of four phases. In the first phase, real-time edits are fetched via a web socket connection, the metadata of the fetched edits is analyzed and the title and the URL of the edited Wikipedia article are extracted. In the second phase, the extracted URL is visited to obtain the edited information, keywords are extracted from the edited information, stop words are removed and stemming of the keyword is performed. In the third phase, a search query, with respect to the title of the edited Wikipedia article is generated, DBpedia ontology is initialized, WordNet database is searched for synsets (a group of synonyms) of the keywords, and the web is crawled via the multithreaded web crawler. Finally, in the last phase, every crawled html page is parsed to calculate the page weight or page score, anchor links are fetched for further crawling and finally, the relevance of the Wikipedia edit is mapped.

This chapter will discuss in detail the proposed process of calculating relevance of the Wikipedia edits with respect to DBpedia ontology, WordNet lexical database, multithreading, web crawling and relevance computation. The flow of the proposed system is shown in Figure 2.1.

2.2 Phase 1

2.2.1 Fetching edits via web socket

A web socket is a two-way communication between a server and a client. In this study, a web socket client is implemented to listen to wikimon – a Wikipedia monitor – to fetch real time edits on Wikipedia articles. This is shown in Figure 2.2.
The real-time edits that are fetched from wikimon contains important information about the metadata of the edits. These include title of the edited Wikipedia article and a URL assigned to it. Table 2:1 shows metadata of an edited Wikipedia article fetched via a web socket.

**Figure 2.2 Fetching edits via web-socket**

**2.2.2 Analyze the metadata of the fetched edits**

Not all attributes that are fetched via a web socket are crucial. For this study, we take into consideration the following attributes:
• “action” – nature of the action taken on the Wikipedia article. All the edits will have value for this attribute as “edit”.

• “is_minor” – if value for this attribute is “yes”, the edit on the Wikipedia article is a minor edit, else, it is a major edit.

• “page_title” – title of the edited Wikipedia article

• “URL” – URL to the edited Wikipedia article

```
{
  "action": "edit",
  "change_size": 23,
  "flags": null,
  "hashtags": [],
  "is_anon": false,
  "is_bot": false,
  "is_minor": false,
  "is_new": false,
  "is_unpatrolled": false,
  "mentions": [],
  "ns": "Main",
  "page_title": "Edward Bartley",
  "parent_rev_id": "767998676",
  "rev_id": "767998391",
  "summary": null,
  "user": "Wtp29"
}
```

**Table 2:1 Metadata of an edited article fetched via web socket**

We analyze the above-mentioned attributes to confirm that the article is edited and the edit has a major impact. Minor edits are not taken into consideration in this study. This is because minor edits comprise of inserting a white space between words, inserting tabs, returning a new line, deleting space, etc. In this study, we are focused on major edits like change of information and addition or deletion of facts.
2.2.3 Extract URL and title from metadata

After an edit is analyzed as a major edit, we extract the title and URL of the edited Wikipedia article from the metadata for further processing.

2.3 Phase 2

2.3.1 Visit URL to extract keywords

In this phase, the extracted URL is visited using the *JSoup Java HTML Parser* that helps to extract the data stored in the HTML documents. The edited, added or deleted information is obtained using the java library of methods contained in JSoup. The obtained information is then tokenized and saved as a list of keywords.

2.3.2 Stoplisting

Stop words are an integral part of a sentence or a phrase structure. Every page contains many stopwords, such as “it”, “can”, “the”, “and”, “so”, “like”, “an”, “as well as”, etc. Stoplisting is the process of removing those stopwords from the list of keywords. Removing stopwords when working with search engines and web crawlers is necessary, because stopwords do not relate to the content or describe the content, and they carry no value, unless read in a sentence by a human. Eliminating stop words can decrease index size, increase retrieval efficiency, and improves effectiveness that ultimately helps web crawlers to perform better in searching results.

In this study, we compare our list of keywords with a list of stop words. This helps us to eliminate words that do not relate to the content. It also helps to extract all possible words, phrases, terms or concepts that can potentially be keywords. An example of stoplisting is given in Table 2:2.
Hello World is typically the first basic program every beginner writes when learning any new programming language.

<table>
<thead>
<tr>
<th>List of keywords before stoplisting</th>
<th>List of keywords after stoplisting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hello</td>
<td>Hello</td>
</tr>
<tr>
<td>World</td>
<td>World</td>
</tr>
<tr>
<td>is</td>
<td>typically</td>
</tr>
<tr>
<td>typically</td>
<td>program</td>
</tr>
<tr>
<td>the</td>
<td>beginner</td>
</tr>
<tr>
<td>first</td>
<td>writes</td>
</tr>
<tr>
<td>basic</td>
<td>learning</td>
</tr>
<tr>
<td>program</td>
<td>programming</td>
</tr>
<tr>
<td>every</td>
<td>language</td>
</tr>
<tr>
<td>beginner</td>
<td></td>
</tr>
<tr>
<td>writes</td>
<td></td>
</tr>
<tr>
<td>when</td>
<td></td>
</tr>
<tr>
<td>learning</td>
<td></td>
</tr>
<tr>
<td>any</td>
<td></td>
</tr>
<tr>
<td>new</td>
<td></td>
</tr>
<tr>
<td>programming</td>
<td></td>
</tr>
<tr>
<td>language</td>
<td></td>
</tr>
</tbody>
</table>

**Table 2:2 Importance of stoplisting**

### 2.3.3 Stemming

The process of stemming normalizes words by conflating several morphologically similar words to a single root form or stem. For example, “connect”, “connected” and “connection” are all reduced to “connect”.

In this study a standard stemmer, the *Porter stemmer* is used for stemming terms using the *Apache Lucene API*. The Porter stemming algorithm focuses on the process of performing linguistic normalizations on words. In this process, the variant forms of a word are reduced to a common form as shown in Table 2:3.
<table>
<thead>
<tr>
<th>Word 1</th>
<th>Word 2</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matching</td>
<td>matches</td>
<td>Match</td>
</tr>
<tr>
<td>Connection</td>
<td>connected</td>
<td>Connect</td>
</tr>
<tr>
<td>Engineering</td>
<td>engineered</td>
<td>Engineer</td>
</tr>
<tr>
<td>Failing</td>
<td>failed</td>
<td>Fail</td>
</tr>
<tr>
<td>Hissing</td>
<td>hissed</td>
<td>Hiss</td>
</tr>
</tbody>
</table>

Table 2:3 Stemming example

2.4 Phase 3

2.4.1 Generating a search query

In this phase, a search query is generated which also acts as a seed URL for web crawling. In this study, we have used as our default search engine. Using JSoup we send our search query to Google. An example is shown in Table 2:4. The search query comprises of nothing but the title of the edited Wikipedia article extracted in Phase 1. In the given example, our query “q” is “Edward Bartley” that is being searched on the Google search engine.

```java
    .userAgent("Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html")
    .timeout(5000).get();
```

Table 2:4 JSoup Search query
2.4.2 Initialize DBpedia Ontology

In the context of computer and information sciences, an ontology defines a set of representational primitives with which a domain of knowledge is modeled. To understand this concept better, let us first understand what is ontology is and its importance. The term ontology means “specification of a conceptualization” [24] in the context of knowledge sharing. That is, an ontology is a description (like a formal specification of a program) of the concepts and relationships that can exist for an agent or a community of agents.

An ontology allows a programmer to specify the concepts and relationships that collectively characterize some domain of interest, in an open, meaningful way. Examples might be the concepts of red and white wine, grape varieties, vintage years, wineries and so forth that characterize the domain of “wine”, and relationships such as “wineries produce wines”, “wines have a year of production”.

To interpret the concepts and relationships computationally, a model called RDFS (Resource Description Framework Schema) is commonly used. RDFS encodes semantic relationships between items of data. It allows ontologists to build a simple hierarchy of concepts and a hierarchy of properties. It is a W3C (World Wide Web Consortium) recommendation for describing resources for the semantic web [25] [26]. An important benefit of using ontology is the ability to derive additional truths about the concepts being modeled. Ontologies also fill in the missing information about a concept [27].
2.4.2.1 DBpedia Ontology

In this study, the DBpedia ontology is used. DBpedia utilizes an adaptable and extensible framework to extract various types of organized information from Wikipedia. It is a shallow, cross-domain ontology, which has been manually created based on the most commonly used infoboxes within Wikipedia. It allows us to semantically query relationships and properties of Wikipedia resources, including links to other related datasets.

2.4.2.2 Utilizing DBpedia Ontology

DBpedia ontology is queried using Jena API. Jena supports RDFS ontology language. Our aim is to extract an ontological hierarchy for a given concept. That is if the given concept is “Lexus”, the additional information that we will have is “Luxury” and “Car” (as seen in Figure 2:3). This additional information is presented as the ontological hierarchy of given concept, which is stored as a list after extraction.

Figure 2.3 Simple Concept Ontological Hierarchy
In this study, DBpedia ontology is downloaded and stored in Jena TDB (a component of Jena for RDF storage and query) store and queried using TBDquery. The TBD is then indexed locally using Jena API. The local TDB is then queried using the title of the edited Wikipedia article extracted in Phase 1.

A sample code for creating a local TDB store, loading and indexing it locally and querying it using Jena API for Java is shown in Table 2:5 Sample code for creating, loading, indexing and querying a local TDB store.

```
Load and index TDB
/** The Constant tdbDirectory. */
public static final String tdbDirectory = "Users/poonamgohil/Thesis/TDBLoadOntology";

/** The Constant dbdump0. */
public static final String dbdump0 = "Users/poonamgohil/Thesis/dbpedia_3.8/dbpedia_3.8.owl";

/** The Constant dbdump1. */
public static final String dbdump1 = "Users/poonamgohil/Thesis/dbpedia_en.nt";

Model tdbModel = TDBFactory.createModel(tdbDirectory);

FileManager.get().readModel(tdbModel, dbdump0);
FileManager.get().readModel(tdbModel, dbdump1, "N-TRIPLES");
tdbModel.close();

Query the Jena TDB

String queryStr = "title of the edited Wikipedia article";

Dataset dataset1 = TDBFactory.createDataset(tdbDirectory);
Model tdb = dataset1.getDefaultModel();

Query query1 = QueryFactory.createQuery(queryStr);
QueryExecution qexec = QueryExecutionFactory.create(query1, tdb);

/*Execute the Query*/
ResultSet results = qexec.execSelect();

while (results.hasNext()) {
    /*Store the results in a list*/
}
qexec.close();
tdb.close();
```

Table 2:5 Sample code for creating, loading, indexing and querying a local TDB store

2.4.3 Search WordNet for synsets

In this phase, WordNet – a large lexical database of English, is searched to find synsets – set of synonyms, of the extracted keywords. WordNet specifically resembles a thesaurus, where
words are grouped together based on their meanings. It consists of nouns, verbs, adjectives and adverbs that are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept.

Symsnets, hence is a group of synonyms that are interlinked by means of conceptual semantic and lexical relations. The semantic relations in WordNet apply broadly throughout English language.

WordNet database is accessed locally in this study with the help of the Java API for WordNet Searching (JAWS). As its name implies, JAWS is an API that provides Java applications with the ability to retrieve data from the WordNet database. It is a simple and fast API that is compatible with both the 2.1 and 3.0 versions of the WordNet database files and can be used with Java 1.4 and later.

<table>
<thead>
<tr>
<th>Semantic Relations</th>
<th>Syntactic Category</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synonymy (similar)</td>
<td>Noun, Verb, Adjective, Adverb</td>
<td>tube, pipe</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rise, ascend</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unhappy, sad</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rapidly, speedily</td>
</tr>
<tr>
<td>Antonymy (opposite)</td>
<td>Adjective, Adverb, (Noun, Verb)</td>
<td>wet, dry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>powerful, powerless</td>
</tr>
<tr>
<td></td>
<td></td>
<td>friendly, unfriendly</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rapidly, slowly</td>
</tr>
<tr>
<td>Hyponymy (subordinate)</td>
<td>Noun</td>
<td>sugar maple, maple</td>
</tr>
<tr>
<td></td>
<td></td>
<td>maple, tree</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tree, plant</td>
</tr>
<tr>
<td>Meronymy (part)</td>
<td>Noun</td>
<td>brim, hat</td>
</tr>
<tr>
<td></td>
<td></td>
<td>gin, martini</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ship, fleet</td>
</tr>
<tr>
<td>Troponomy (manner)</td>
<td>Verb</td>
<td>march, walk</td>
</tr>
<tr>
<td></td>
<td></td>
<td>whisper, speak</td>
</tr>
</tbody>
</table>

Table 2:6 Semantic Relations in WordNet
2.4.3.1 Algorithm for obtaining synonyms for a given keyword

Below are the steps taken for obtaining synonyms for a given keyword extracted from Wikipedia edits.

1. Obtain an instance of WordNet database
2. Search the database to find synsets for the given keyword.
3. Get hyponyms (words of more specific meaning than a general or superordinate term applicable to it. For example, purple is a hyponym of violet) for each retrieved synset.
4. Get hypernyms (a word with a broad meaning that more specific words fall under or a superordinate. For example, color is a hypernym of red) for each retrieved synset.
5. Get word forms – string of letters, for all hyponyms and hypernyms retrieved. This is the list of synonyms obtained for a given keyword.

![Figure 2-4 Hyponyms and Hypernyms](image)

The synonyms retrieved for the keywords, help in enhanced relevance calculation as discussed in Page Weight Calculation.

2.4.4 Crawling the Web using Multithreading

To understand this task better, let us understand how multithreading is achieved in this study, what a Web Crawler is and its architecture.
2.4.4.1 Multithreading

Multithreading is a technique that allows a program or a process to execute many tasks concurrently (at the same time and parallel). It allows a process to run its tasks in parallel mode on a single processor system. In the multithreading concept, several multiple lightweight processes are run in a single process/task or program by a single processor. For Example, when a word processor is used, many different tasks such as printing, spell check, etc. are performed. Multithreaded software treats each process as a separate program.

Web crawling speed is governed not only by the speed of one’s own internet connection, but also by the speed of the sites that are to be crawled. Especially if one is crawling sites from multiple servers, the total crawling time can be significantly reduced if many downloads are done in parallel.

![Figure 2:5 Execution of multiple threads in a single process](image)

The idea of using multithreading in this study is to achieve better parallelism by dividing the crawling process among separate independent threads. Fetching pages from the web can be carried out independently by separate threads. A thread controller will be responsible for creating the threads and managing the URL frontiers of each thread. In implementing multithreading, a single URL Frontier becomes a bottleneck for the working of \(N\) active threads.
Thus, we implement N URL Frontiers by which each hyperlink will be stored in the mapped URL Frontiers. URL Frontier is shown in Figure 2:7.

2.4.4.2 Web Crawler

A Web crawler, sometimes referred as a spider, is an Internet bot that systematically browses the World Wide Web. Prior to the Web turning into the most unmistakable piece of the Internet, there were already web crawlers set up to help individuals discover data on the internet. Programs with names like "gopher" and "Archie" kept indexes of files that were put away on servers which connected with the Internet, and drastically reduced the amount of time required to find programs and documents. Before a search engine can tell you where a file or document is, it must be found. To discover data on the countless Web pages that exist, a web search tool - specifically a web engine - utilizes unique programming robots, called spiders, to construct lists of the words found on websites. The process of a spider building its lists is called Web Crawling.

To understand this better, view WWW (World Wide Web), or Web as a directed graph, with its vertices representing the web pages and the directed edges representing the links from
one webpage to the other. A Web Crawler is a program that uses this graph structure of the web (the web-graph) to visit web-pages in an automated manner by following the edges from every page in the web-graph. This process is called crawling the web or web crawling.

2.4.4.3 Architecture of a Web Crawler

The crawler is initiated with several seed URLs from the URL queue frontier. The crawler fetches a URL from the frontier, visits the URL, parses the fetched page and then adds the links found on the page to the frontier. This process is referred as crawling loop. The crawler can stop when a certain number of pages are crawled, or when a certain crawl-depth is reached to end the loop.

![URL Frontier Queue](image)

**Figure 2:7 URL Frontier Queue**
Figure 2:8 Flow of the designed multithreaded web crawler
• **URL Frontier Queue**

A URL Frontier queue is a special kind of collection in which the entities in the collection are kept in an order. The principal or the only operation on this collection is the addition of entities to the rear terminal position and removal of entities from the front terminal position. This makes the queue a First-In-First-Out (FIFO) data structure. The frontier can quickly get very large as pages are crawled. With an average of 10 links a page, the frontier will contain about 100 URLs when 10 pages are crawled.

2.5 **Phase 4**

2.5.1 **Fetching**

After the crawler receives a new URL from the frontier, it fetches the page from the Web. To do so, *JSoup – a Java HTML Parser* is used. It provides a very convenient API for extracting and manipulating data, using the best of DOM (Document Object Model), CSS (Cascading Style Sheets), and jquery-like methods.

2.5.2 **Preprocessing**

Pre-processing steps are taken when a page has been fetched to the crawler’s local memory. Pre-processing is done to identify and extract the useful information. To add the URLs from the page to the frontier, these URLs must first be extracted from the page. A parser is used to find the anchor tags and grab the values of the associated “href” attributes and add the URL to the frontier for visiting. HREF is an acronym for Hypertext reference, that serves as a pointer to another page. The crawler then uses text-parsing techniques to
extract content information from the page. This content is then used to index the page and give a weight or score to a page.

![Diagram of Process Cycle of a single thread](image)

**Figure 2:9 Process Cycle of a single thread**

### 2.5.3 Page Weight Calculation

Page score calculation helps in calculating the prominence/importance of each keyword in a page. At this point we have an updated list of keywords which includes the keywords extracted from the edited wikipedia article, synonyms obtained from the WordNet database and the keywords extracted from the ontological hierarchy as shown in Figure 2:10

![Diagram of List of keywords](image)

**Figure 2:10 Structure of the list of keywords**
After a URL has been visited, the source code of the visited page must be analyzed with respect to the list of keywords to get the page weight. This process is called as parsing the web page. To make a simple web page you need to know the following tags:

- `<HTML>` tells the browser your page is written in HTML format
- `<HEAD>` this is a kind of preface of vital information that does not appear on the screen.
- `<TITLE>` Write the title of the web page here - this is the information that viewers see on the upper bar of their screen.
- `<BODY>` This is where you put the content of your page, the words and pictures that people read on the screen.
- `<META>` This element used to provide structured metadata about a Web page. Multiple Meta elements with different attributes are often used on the same page. Meta elements can be used to specify page description, keywords and any other metadata not provided through the other head elements and attributes.

The following rules will affect the total weight of a page. Keywords are weighted depending on where in the page they are found. (importance of each rule is ordered in descending order):

Whether the -

⇒ The body tag includes the keyword
⇒ The title tag includes the keyword
⇒ The Meta tag includes the keyword
⇒ The heading tag (h1 through h6) includes the keyword
⇒ The URL includes the keyword
The above rule can be expressed as prominence of a keyword, that is, how close to the beginning of the sentence the keyword is found. The analysis of the source code is as follows:

1. Let the total weight of the keyword be “t” units
2. The body tag has got the weight “B” units
3. The title tag has got the weight “T” units
4. The Meta tag has got the weight “M” units
5. The heading tag (h1 through h6) has weight “H” units
6. The URL has weight “U” units
7. The no. of occurrence of the search string in body be “nB”
8. The no. of occurrence of the search string in title be “nT”
9. The no. of occurrence of the search string in META be “nM”
10. The no. of occurrence of the search string in heading be “nH”
11. The no. of occurrence of the search string in URL be “nU”

The total weight of the keyword or prominence can be given as follows:

\[ t = (nB \times B) + (nT \times T) + (nM \times M) + (nH \times H) + (nU \times U) \]

Assume following units for calculating total weight or prominence of the keyword “t”:

- M = 10 units, U = 8 units, T = 6 units, H = 4 units, B = 1 units

Suppose the search keyword is: “university” (not case-sensitive). The number of occurrences of the keyword “university” in the following tags are as follows:

- nB = 3, nT = 1, nM = 2, nH = 0, nU = 1

The total weight of the keyword or prominence comes out to be:

\[ t = (3 \times 8) + (1 \times 6) + (2 \times 10) + (0 \times 4) + (1 \times 8) \]
t = 24 + 6 + 20 + 0 + 8

\[ t = 58 \]

Besides prominence, the frequency of the keyword appearing in the page is also very important. For example, consider a page where the keyword appears 10 times (page A) may get higher score than a page where the keyword appears only 4 times (page B). However, it is not always true if the prominence of page B is much better than page A. Frequency of a keyword appearing in a page is not fair when the size (number of words) of pages is not equal. For example, a page with 10000 words might probably have higher frequency than a page with only 1000 words. Therefore, we studied a better way of counting the frequency called “page weight” or “relevance score”.

The page weight (w) can be determined as follows:

\[
\text{page weight (w)} = \frac{\text{total weight of the keyword (prominence)}}{\text{total number of words in a page}}
\]

Assume that there is a total of 100 words in a page and the total weight or prominence of the keyword “university” in the page is 58 (as computed above). Therefore, the weight of a page or relevance score is calculated as follows:

\[
\text{page weight (w)} = \frac{\text{total weight of the keyword (prominence)}}{\text{total number of words in a page}}
\]

\[ w = \frac{58}{100} \]

\[ w = 0.58 \]
2.5.4 Relevance Mapping of Wikipedia edit

To analyze if an edit on the Wikipedia article is relevant or irrelevant, the relevance of all the pages crawled for a given seed URL is taken into consideration. The seed URL for the crawler is the search query with the title of the edited Wikipedia article generated in Phase 3.

In this study, we limit our crawl to maximum of 100 web pages for a given seed URL, which means, for a given seed URL (which is the google search query as mentioned above), top 10 links are fetched from the google search, that are crawled and for every crawled URL, maximum of 10 anchor links from that web page are fetched for further crawling. This is depicted in the Figure 2:11.

The average page weight or the average page score of all these web pages determines if an edit on the Wikipedia article is relevant or irrelevant. The threshold for an edit to be categorized as relevant is 0.5 and above. If the average page weight (w) of all the web pages crawled is greater than or equal to 0.5, then the edit is relevant. That is, an edit on a Wikipedia article is considered relevant if and only if the average (page weight (w) of all web pages crawled) $\geq 0.5$. 
Figure 2:11 A process of crawling and fetching of the URLs

Therefore, prominence i.e. total weight of a keyword and page weight are two major factors for calculating relevance of an edit.

2.6 Conclusion

A four-phase semantic system is proposed in this chapter to map the relevance of an edit on the Wikipedia article. The first phase is involved with fetching Wikipedia edits to be analyzed. The second phase is a preprocessing phrase, generating clean edits for third phase. The third phase is the most important phase that involves DBpedia and WordNet for semantic evaluation of edits. The fourth and final phase relies on the semantic web to gather information on relevance
of edits using the concept of prominence i.e. total weight of a keyword and page weight are two major factors for calculating relevance of an edit.
Chapter 3: Evaluation of Wikipedia Edit Analysis System

3.1 Introduction

Two studies were conducted to empirically determine the performance of a multithreaded web crawler and a simple web crawler. The first study focuses on the execution time performance of each web crawler. The second study focuses on comparative efficiency of a web crawler with ontology and a web crawler without ontology.

For the execution time study, the first goal was to determine if the crawler can scale to continual input, process and output of data. The second goal was to have a highly-optimized execution path that minimizes the ratio of overhead to useful work. Finally, the third goal was to compare the performance of a simple crawler without multithreading with a multithreaded web crawler.

For the comparative efficiency study, the goal was to compare the relevance of a page against a given ontology commitment vs. a simple keyword. When semantic data was found using an ontological commitment, the page weight calculation was computed for all the concept labels in the ontological commitment. When only simple keyword was used, only this keyword was used to compute the weight of the page.

The first study was performed by recording the execution time taken by each crawler to crawl up to 900 pages and the second study was performed by recording the relevance of a page against given ontology and simple keyword. For this study, DBpedia ontology, WordNet and multithreading concepts were used as discussed in DBpedia Ontology, Search WordNet for synsets and Multithreading.
3.2 Results

3.2.1 Crawler Performance

In this section, we provide some statistics on the performance of the crawler. The crawler is run with several crawler threads as we have discussed in the multithreading section to record execution time of the crawler with multithreaded architecture and without crawler threads as per the single threaded execution flowchart shown in Figure 3:1 to record execution time of a simple crawler. In the following test, we have used ten threads. An average of 925 web pages were crawled by both single threaded web crawler and multithreaded web crawler to compare the performance by recording the execution time. The execution time in this study is given in milliseconds.

![Crawler Performance](image)

Figure 3:1: The CPU time in milliseconds needed to process a single page by a multithreaded web crawler
Figure 3:2 The CPU time in milliseconds needed to process a single page by a single threaded web crawler

In both cases, we monitor the average CPU time it takes to process one page. Since we use several threads in a multithreaded architecture, we cannot measure the actual time it takes for a page to be processed. While a crawler thread is processing a page, the CPU will also give the other threads time to process their pages. This would result in faulty values for CPU times for every thread. Hence, we have used the management interface for the thread system of the Java virtual machine to measure the CPU computational time for every thread separately. In this test, the crawler has been run on a Mac system. For every twenty-five pages, the average CPU time is plotted on the graph in Figure 3:1: The CPU time in milliseconds needed to process a single page by a multithreaded web crawler and Figure 3:2 The CPU time in milliseconds needed to process a single page by a single threaded web crawler.

As depicted in the graph above, the average CPU time stays between 400 to 550 milliseconds for a simple web crawler, whereas, the average CPU time stays below 100 milliseconds for a multithreaded web crawler. We noticed that the time needed to process a page
by a multithreaded web crawler did not significantly increase during the crawl. We also noticed
two high peaks in the performance graph. This can be explained by the fact that there might be a
few links that point to a host that is “dead” or “offline” on a certain page. The crawler threads
will try to fetch the pages on these hosts and will receive a connection timeout. During such
timeouts, the crawler thread cannot process other pages. One reason why we use several crawler
threads is to get around this problem. While one thread is waiting for the connection timeout,
other threads can still process other pages. This increases the speed of the crawler significantly.
Still, when a certain page contains a lot (more than the number of threads) of links to a “dead” or
“offline” host, all the threads will be waiting on a connection timeout. This scenario, where all
the threads are waiting for a connection timeout can be perceived twice on the graph.

3.2.2 Comparative efficiency

For this study, we have briefly evaluated our relevance computation system by
performing an empirical study of our multithreaded web crawler with ontology in an
uncontrolled and practical environment. Perhaps the most crucial evaluation metric in focused
crawling is the rate at which relevant pages are acquired, and how effectively irrelevant pages are
filtered from the crawl. This evaluation metric is called the “harvest rate”. Harvest rate can be
denoted by \( P(C) \) can be defined as the rate at which crawled pages satisfy a given predicate; if a
classifier is used to give numeric relevance values then a page is said to satisfy a predicate if the
relevance value exceeds a certain “threshold” [28]. The major impediment is defining that
“threshold”. In this evaluation, pages that have a relevance score of 0.5 or higher will be
considered as relevant. We found this score to be the best threshold value after manually
evaluating few example pages.
To compare the efficiency of the two crawlers, with ontology and without ontology, we compare the harvest rate on the two crawls. The first crawl uses only keywords as the crawl criteria whereas the subsequent crawls use the DBpedia ontology.

![Graph showing harvest ratio comparison](image)

**Figure 3:3 The harvest ratio of a crawler using ontology v/s a crawler using only a keyword.**

As seen in the graph above, we notice that both the crawlers have a similar harvest rate in the beginning of the crawl. This is because, in the beginning both the crawlers will come across same web pages. After crawling few pages, the ontology based crawler will come across more relevant pages based on the ontological hierarchy accessed that the keyword based crawler will not be able to find. Due to this, we notice a dissimilarity in the harvest rate graph. As the crawl advances, the ontology based crawler’s harvest rate gets close to 1 which denotes that the crawler is crawling the pages that are most relevant to the topic. The keyword based crawler; however, has a fluctuating harvest rate which denotes that it is crawling the pages that are relevant as well.
as irrelevant. Hence we conclude that the ontology based crawler is much more efficient than a simple crawler.

3.2.3 Evaluation of the Proposed System with respect to relevance mapping

In this section, we define the metrics used for the evaluation of the proposed system. The system was tested through some experiments to evaluate its performance. As far as the proposed system is concerned, the process of relevance mapping includes the task of query expansion, the task of crawling and categorizing the edits as relevant or irrelevant based on ontology.

Based on the performance of the proposed system, we can speak of 4 disjoint sets of documents.

- True positive (TP): This set contains all relevant URLs that have been identified by the system as relevant.
- False positive (FP): This set contains all irrelevant URLs that have been identified by the system as relevant.
- True negative (TN): This set contains all irrelevant URLs that have been identified by the system as irrelevant.
- False negative (FN): This set contains all relevant URLs that have been identified by the system as irrelevant.

Clearly, two only of these sets are desired: the set of true positive and the set of true negative, and for this reason two evaluation metrics are defined. The most common evaluation metrics for IR system performance are recall, R, and precision, P. Precision is a measure of result relevancy, while recall is a measure of how many truly relevant results are returned.

Precision, P, and Recall, R are defined as follows:
Recall (R) is defined as the number of true positives (TP) over the number of true positives (TP) plus the number of false negatives (FN).

\[ R = \frac{|TP|}{|TP| + |FN|} \]  

(1)

Precision (P) is defined as the number of true positives (TP) over the number of true positives (TP) plus the number of false positives (FP).

\[ P = \frac{|TP|}{|TP| + |FP|} \]  

(2)

It is apparent from the graph for recall analysis, Figure 3:4 Recall of the proposed system that the recall value varies from 52% to 88%, which reflects the completeness or sensitivity of our
The recall value here means less number of crawl jobs that are false negative in nature, or in simple words, how many truly relevant URLs are returned as relevant.

The results indicate clearly that the proposed system provides higher precision. This indicates that the measure of the relevancy of the result is high.

In statistical analysis, the F-score (also F1 score or F-measure) is a measure of a test's accuracy. It considers both the precision, P, and the recall, R, of the test to compute the score. The F-score can be interpreted as a weighted average of the precision and recall, where an F-score reaches its best value at 1 and worst at 0.

Figure 3:5 Precision of the proposed system
This measure is approximately the average of the two when they are close, and is more generally the harmonic mean, which, for the case of two numbers, coincides with the square of the geometric mean divided by the arithmetic mean.

\[
F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}
\]

![Figure 3:6 F-Score of the proposed system](image)

Figure 3:6 F-Score of the proposed system

It is apparent from the graph in Figure 3:6 F-Score of the proposed system, that the system has an average F-score of 0.77. This indicates that the system has good accuracy.

### 3.3 Conclusion

In this chapter, we evaluated the efficiency of multithreaded crawler over single threaded crawler for crawling the webpages. The multithreaded crawler shows improved performance with respect to time. Another important finding was the use of ontology improved modeling of domain of interest and guiding the crawler to find relevant pages in web and helping in making decision on edits are relevant or not. This is evaluated using the precision and recall evaluation metrics. We can also conclude that the system is 77% accurate.
Chapter 4: Conclusions and Future Work

In this work, a four-phase semantic system is proposed to assign relevance to an edit on a Wikipedia article. The first phase is involved with fetching Wikipedia edits to be analyzed. The second phase is a preprocessing phrase, generating clean edits for third phase. The third phase is the most important phase that involves DBpedia and WordNet for semantic evaluation of edits. The fourth and final phase relies on the semantic web to gather information on relevance of edits using the concept of prominence, i.e. total weight of a keyword and page weight are two major factors for calculating relevance of an edit.

We evaluated the efficiency of multithreaded crawler over single threaded crawler for crawling the webpages. The multithreaded crawler resulted in improvement in performance with respect to time as expected. Another important finding was the use of ontology improved modeling of domain of interest and guiding the crawler to find relevant pages in web and helping in making decision on edits are relevant or not.

It can be concluded that with the use of ontology and web crawling, vandalism on the Wikipedia articles can be identified. Multithreaded architecture for a web crawler helps in speeding the process of crawling the web. A multithreaded web crawler can crawl up to four times of the total number of pages crawled by simple web crawler in same amount of time. Ontology helps in increasing the relevancy of web page.

In this proposed work, semantic properties of Semantic Web have been used to provide a better search options to identify an edit on Wikipedia as relevant or irrelevant. This system uses the ontology for semantic related concept search. But, this proposed system is designed for a specific ontology which is DBpedia. DBpedia ontology extracts structured information from Wikipedia. In future, this work can be extended for multiple Ontologies. Semantic related
concepts from ontology are used for query expansion and concepts relations are used in the calculation of relevance of pages. In further extension of this work some other improved algorithms can be utilized for query expansion and ranking of the pages like DMatch Ontology Matcher and the DOGMA Framework.

In the future, we also plan to validate the infobox within a Wikipedia article. Inconsistencies could be pointed out along with proposals on how to solve these inconsistencies. This way, we could contribute back to the Wikipedia community and help to improve the overall quality of Wikipedia. The support for different languages should also be extended.
Appendix 1: System Verified Relevance Mapping

Relevance: 1

Relevance: 1

Relevance: 1

Relevance: 0

Relevance: 1

Relevance: 1

Relevance: 1

Relevance: 1

Relevance: 1

Relevance: 1

Relevance: 1

Relevance: 1

Relevance: 1

Relevance: 0
Appendix 2: User Verified Relevance Mapping

Bibliography


