



***Semi-automatic web resource discovery  
using  
ontology-focused crawling***

by

***Erik Kristoffersen  
Marius A. Sætren***

**Master's Thesis in  
Information and Communication Technology**

**Agder University College  
Faculty of Engineering and Science**

**Grimstad**

**May 2005**

## Abstract

The enormous amount of information available on the Internet makes it difficult to find resources with relevant information using regular breadth-first crawlers. Focused crawlers seek to exclusively find web pages that are relevant for the user, and avoid downloading irrelevant web pages. Ontologies have recently been proposed as a tool for defining the target domain for focused crawlers.

In this project we have developed a prototype of an ontology-focused crawler. We have accomplished this by developing extra modules to the Java open source crawler Heritrix. In one of the modules we have developed, we measure the relevancy of web pages in relation to an ontology describing the area of interest. We have also developed a link analysis module to determine the importance of web pages. This module uses the link analysis component from the open source search engine Nutch. The importance measure is used to ensure that the most important web pages are downloaded first.

This thesis also contains an evaluation of several open source crawlers. We found that Heritrix was the easiest to extend, and best suited for our purpose. Our prototype is therefore built upon Heritrix.

To measure the performance of the prototype several test crawls with different settings has been carried out. Focused crawlers are often evaluated by harvest rate, which is the ratio between number of relevant and all of the web pages downloaded. The prototype performed well in the tests, and in one of them the prototype had a harvest rate of about 0.55. In a similar unfocused crawl, the harvest rate was only about 0.15. Both the prototype and the algorithm are designed to be easily configured. More testing and adjustments of the settings could improve the performance of the prototype even further, but we have shown that ontologies are a suitable technology for creating focused crawlers.

## Preface

This thesis is written for the company InterMedium, as a part of the Master of Science degree in Information and Communication Technology at Agder University College. The work has been carried out in the period from January to May 2005.

The project group has consisted of Erik Kristoffersen and Marius A. Sætren. Both have a BSc degree in Computer Science from Agder University College. Presently, Marius is also working part time for InterMedium.

We would like to thank our supervisors, Asle Pedersen at InterMedium and Vladimir Oleshchuk at Agder University College for valuable help and guidance during the project period.

Grimstad, May 2005

*Erik Kristoffersen and Marius A. Sætren*

# Table of contents

<b>ABSTRACT</b> .....	<b>2</b>
<b>PREFACE</b> .....	<b>3</b>
<b>TABLE OF CONTENTS</b> .....	<b>4</b>
<b>LIST OF FIGURES</b> .....	<b>6</b>
<b>LIST OF TABLES</b> .....	<b>7</b>
<b>1 INTRODUCTION</b> .....	<b>8</b>
1.1 BACKGROUND .....	8
1.2 THESIS DEFINITION .....	8
1.3 OUR WORK .....	9
1.4 REPORT OUTLINE .....	9
<b>2 FOCUSED WEB CRAWLING</b> .....	<b>10</b>
2.1 RELATED WORK .....	10
<b>3 ONTOLOGIES</b> .....	<b>12</b>
3.1 INTRODUCTION TO ONTOLOGIES .....	12
3.2 ONTOLOGY LANGUAGES .....	12
3.2.1 <i>RDF</i> .....	12
3.2.2 <i>OWL</i> .....	13
3.2.3 <i>Topic Maps/XTM</i> .....	13
3.3 SELECTED ONTOLOGY LANGUAGE .....	13
3.3.1 <i>TM4J – Topic Maps for Java</i> .....	14
3.3.2 <i>Ontopia Omnigator</i> .....	15
3.4 ONTOLOGY-FOCUSED CRAWLING .....	15
<b>4 EVALUATION OF EXISTING CRAWLERS</b> .....	<b>16</b>
4.1 NUTCH .....	16
4.2 HERITRIX .....	17
4.3 WEBLECH .....	17
4.4 WEBSPHINX .....	18
4.5 J-SPIDER .....	19
4.6 HYPERSPIDER .....	19
4.7 ARALE .....	20
4.8 EXTENDING AN EXISTING CRAWLER OR DEVELOPING A NEW CRAWLER .....	20
4.8.1 <i>Extending an existing web crawler</i> .....	20
4.8.2 <i>Developing a web crawler from scratch</i> .....	21
4.9 EVALUATION .....	21
<b>5 DESCRIPTION OF HERITRIX</b> .....	<b>22</b>
5.1 FRONTIER .....	22
5.2 TOETHREADS .....	23
5.3 CRAWLURI AND CANDIDATEURI .....	23

5.4	CRAWLSCOPE .....	23
5.5	PROCESSOR CHAINS .....	23
<b>6</b>	<b>FOCUSING ALGORITHM AND SEARCH STRATEGY .....</b>	<b>24</b>
6.1	ONTOLOGY BASED COMPARISON OF DOCUMENTS .....	24
6.2	OUR RELEVANCE ALGORITHM .....	24
6.3	LINK ANALYSIS .....	27
<b>7</b>	<b>THE PROTOTYPE .....</b>	<b>28</b>
7.1	NEKOHTMLPARSER .....	29
7.2	RELEVANCECALCULATOR .....	29
7.2.1	<i>initialTasks()</i> .....	30
7.2.2	<i>innerProcess()</i> .....	30
7.3	WEBDBPOSTSELECTOR .....	32
7.3.1	<i>Description of WebDBPostselector</i> .....	32
7.3.2	<i>ModifiedBdbFrontier</i> .....	34
7.3.3	<i>Modified Page class in Nutch</i> .....	34
<b>8</b>	<b>PROTOTYPE TESTS .....</b>	<b>36</b>
8.1	PERFORMANCE MEASURES .....	36
8.2	TEST SETTINGS .....	37
8.2.1	<i>Scope filter</i> .....	37
8.2.2	<i>Seed URLs</i> .....	38
8.2.3	<i>Input Ontology</i> .....	38
8.3	TEST RESULTS .....	39
<b>9</b>	<b>DISCUSSION .....</b>	<b>45</b>
9.1	THE PROTOTYPE .....	45
9.1.1	<i>Problems running Nutch on Windows</i> .....	46
9.1.2	<i>Problems with DNS lookup in Heritrix</i> .....	47
9.2	TEST RESULTS .....	48
9.3	FURTHER WORK .....	50
<b>10</b>	<b>CONCLUSION .....</b>	<b>52</b>
	<b>BIBLIOGRAPHY .....</b>	<b>53</b>
	<b>APPENDIX A - JAVA SOURCE CODE .....</b>	<b>CD ROM</b>

## List of figures

Figure 2.1 a) Standard crawling	b) Focused crawling.....	10
Figure 3.1 TopQuadrant's comparison of some ontology languages in 2003. [19]	.....	14
Figure 3.2 Screenshot from the TMNav application	.....	15
Figure 5.1 Overview of Heritrix. [33]	.....	22
Figure 5.2 Processor chains. [33]	.....	23
Figure 6.1 A very simple TM about Toyota.	.....	25
Figure 7.1 Overview of Heritrix. The emphasized modules are the modules we have added.	.....	28
Figure 7.2 Flowchart for initialTasks() in RelevanceCalculator.	.....	30
Figure 7.3 Flowchart for innerProcess() in RelevanceCalculator.	.....	31
Figure 7.4 Flowchart for innerProcess() in WebDBPostselector	.....	33
Figure 8.1 Scope filter which removes irrelevant file types.....	.....	38
Figure 8.2 The input ontology used in the tests. Pink connections annotate Superclass-Subclass associations, while all other types of relations are purple.	.....	39
Figure 8.3 Harvest rate of focused crawl with relevancy limit 0.01	.....	40
Figure 8.4 Harvest rate of unfocused crawl with relevancy limit 0.01	.....	41
Figure 8.5 Harvest rate of unfocused crawl with relevancy limit 0.01 and irrelevant seeds	.....	42
Figure 8.6 Harvest rate of focused crawl with relevancy limit 0.02	.....	42
Figure 8.7 Harvest rate of unfocused crawl with relevancy limit 0.02	.....	43
Figure 8.8 Harvest rate of unfocused crawl with relevancy limit 0.02 and irrelevant seeds	.....	43
Figure 8.9 Harvest rate of equal crawls with and without link analysis.....	.....	44
Figure 9.1 Code from net.nutch.LocalFileSystem.java	.....	47

## List of tables

Table 6.1 Weights of the topics from the topic map in Figure 6.1 .....	25
Table 7.1 The processors in Heritrix grouped by processor chain. The emphasized rows describe the modules we have added.....	29
Table 7.2 Attributes in the modified Page class in Nutch .....	35
Table 8.1 Top 10 web pages found by focused crawl with relevancy limit 0.01.....	40
Table 9.1 IDF values for the same term in a typical focused and unfocused crawl.....	48
Table 9.2 Relevancy of some web pages in a focused and an unfocused crawl with equal settings.....	49

# 1 Introduction

## 1.1 Background

In January 2000 the Internet had about 72 million hosts advertised in the DNS system. By January 2005 this number had risen to more than 317 million [1]. This means that the Internet is so huge that even the largest search engines only cover a small fraction of the billions of pages estimated to constitute the Internet. Even though the size of the search engines is increasing very fast, no search engines manage to follow the growth rate of the Internet. The Internet contains more information than ever, and it is difficult to separate the relevant information from the less relevant.

The process of finding relevant resources of information is often called resource discovery. Searching for relevant resources manually is a very time consuming and expensive task. Turning this task into an automated or semi-automated process would be very valuable. Different methods can be used to discover resources automatically.

Focused crawlers, as opposed to regular breadth-first crawlers, try to exclusively follow links that are relevant to a specific topic. This kind of crawler plays an important part in many automatic resource discovery approaches, but different methods for calculating the relevancy are used. Some focused crawlers use example documents and machine learning to determine the relevancy of web pages. Others use one or more keywords. Using keywords would resemble a normal search engine search, except that the relevancy of the pages is calculated for each page as they are downloaded, and not in an index that is the result of a regular web crawl. Recently, it has been suggested to use ontologies to define the target domain for focused crawlers. In this project we have developed a crawler that uses ontologies to focus the search on a specific topic.

## 1.2 Thesis definition

This is the definition of our thesis as of 4<sup>th</sup> of February 2005:

*“Resource discovery refers to the task of identifying relevant resources e.g. on the Internet. One example could be to identify all Internet resources worldwide, which publish news about nano-technology. Manually identifying all these resources is very resource demanding and the task could and should be automated. One approach to automatic resource discovery is to use focused crawlers. As opposed to standard crawlers in use by most search engines, which follow each link, typically applying a breadth-first strategy, focused crawlers instead try to identify the most promising links to follow by using some sort of probability measure. The criteria used in the probability measure are usually based on an analysis of hyperlinks, content and structure. Resource discovery is never done entirely from scratch – there are always some a-priori known resources, starting-points and knowledge about the domain, which could be represented using an ontology. Focused crawlers may use such ontologies to guide the identification of promising links.*”



*The project will include an evaluation of some existing web crawlers to find out if it is possible to use one of them as a basis for an ontology-focused crawler. One or more algorithms for using ontologies in focused crawling will then be found or developed. The students shall develop a demonstrator in which the search strategies and algorithms could be evaluated. The students should also define some test criteria, and metrics for measuring the precision of the crawler.”*

### 1.3 Our work

During the project period an ontology-focused web crawler prototype has been developed. It is built as an extension of an already existing Java open source web crawler called Heritrix. The decision to extend this specific crawler was made after a brief evaluation of some existing open source crawlers written in Java. To be able to do link analysis in our prototype we have included a link analysis module from Nutch called WebDB. Nutch is one of the other crawlers we have evaluated.

To decide how relevant a web page is, we have developed a relevancy algorithm. Our relevancy algorithm uses the TFIDF [2] weighting algorithm and is inspired by the relevance computation strategy described in [3]. The prototype has several parameters that can be set to calibrate it or adjust its behavior.

We have also found methods to measure the performance of the crawler by reading articles about other focused crawlers. The prototype has been tested with different parameters, and the results have been logged and evaluated.

### 1.4 Report outline

Chapter 1 describes the background of the thesis project, and gives a short introduction to resource discovery and web crawling. We also present the thesis definition and a short description of our work. In chapter 2 we give an introduction to focused web crawling, and to other work related to this topic. Chapter 3 explains what we mean by an ontology and gives an introduction to some ontology languages and ontology tools. Our choice of ontology language is also explained here. At the end of the chapter we explain the concept of ontology-focused crawling. Chapter 4 contains an evaluation of some existing Java open source web crawlers, and some possible methods for creating the prototype. In this chapter we also explain why we decided to extend an existing crawler, and why we selected Heritrix as a basis for our prototype. Chapter 5 describes the structure of Heritrix, and how the different parts work. Chapter 6 explains the algorithm used in the prototype as well as the algorithms it is based on. Chapter 7 describes how we have built the prototype and how it works. Chapter 8 describes different measures that can be used to measure the performance of a focused crawler. It also includes a description of the test settings and the input ontology we used for the tests. At the end of the chapter we present the results of our tests. Chapter 9 contains a discussion of our prototype and the test results. Chapter 10 contains a conclusion of our project in general.

## 2 Focused web crawling

The Web might be seen as a social network. Authors of web pages often insert links to other pages relevant to their topic of interest. In this way the Web contains a social network of pages linking to other on-topic pages. The link structure and the content of the pages can be used intelligently to decide which links to follow and which pages to discard. This process is called “focused crawling”.

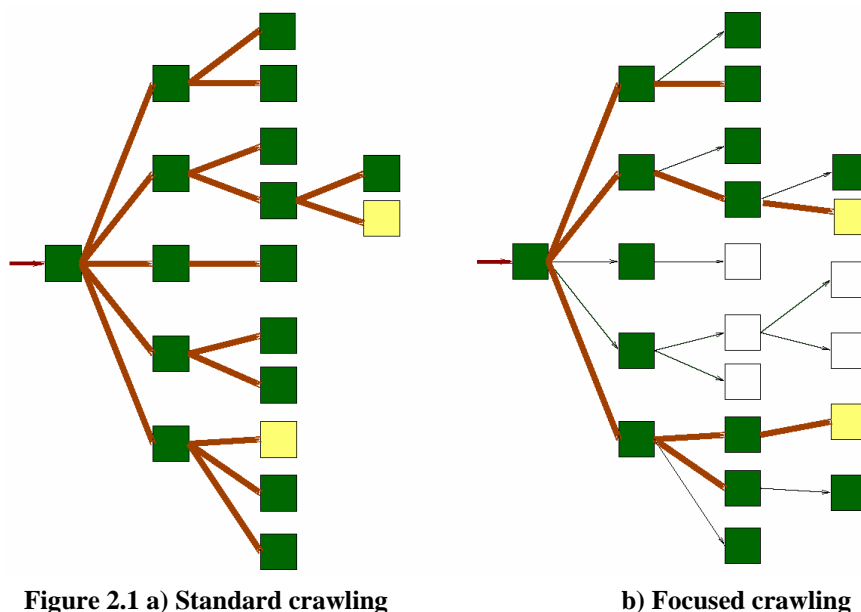


Figure 2.1 a) Standard crawling

b) Focused crawling

Figure 2.1 shows the difference between standard crawling and focused crawling.

a) A standard crawler follows each link, typically applying a breadth first strategy. If the crawler starts from a document which is  $i$  steps from a target document, all the documents that are up to  $i - 1$  steps from the starting document must be downloaded before the crawler hits the target.

b) A focused crawler tries to identify the most promising links, and ignores off-topic documents. If the crawler starts from a document which is  $i$  steps from a target document, it downloads a small subset of all the documents that are up to  $i - 1$  steps from the starting document. If the search strategy is optimal, the crawler takes only  $i$  steps to discover the target. [4]

### 2.1 Related work

Chakrabarti et al. seem to introduce focused crawling for the first time. In the crawler described in their article [5], the user picks a subject from a pool of hierarchically structured example documents. The program learns the subjects by studying the examples, and generates subject models. These models are used to classify web pages. The link structure is also considered by the crawler to discover hubs. Hubs are described by Kleinberg [6] as high-quality lists that guide users to recommended authorities, and authorities are prominent sources of primary content on a topic. Links from hubs can be relevant even though the text on the hub page itself does not appear to be relevant.

The focused crawlers described in the literature generally have a similar structure. One thing that separates them is the algorithm they use to decide whether a web page is relevant. The crawler described by Chakrabarti et al. [5] uses example documents and machine learning principles.

Diligenti et al. [7] describe a focused crawler that uses the same methods to determine the relevancy as Chakrabarti et al. [5]. One difference with Diligenti's crawler is that it generates a context graph that describes the link structure around all the seed documents. This is done to make it easier for the crawler to find relevant documents, hidden behind one or more levels of irrelevant web pages. The web page of a university may for example have links to the home pages of professors, which may have good links to pages about nanotechnology, even though the university web page has no information about nanotechnology.

Diligenti's crawler only focuses on web pages. A user that is interested in finding relevant information resources may be more interested in finding relevant web sites than web pages. Ester et al. [8] explain how a focused crawler can find relevant web sites instead of web pages. Their proposed prototype contains an external and an internal crawler. The internal crawler only views the web pages of a single given web site and performs focused crawling within that web site. The external crawler has a more abstract view of the web as a graph of linked web sites. Its task is to select the web sites to be examined next and to invoke internal crawlers on the selected sites. The crawlers use both link structures and text classifiers to determine site relevancy. The average number of pages used for classification was between 3.2 and 7.4 indicating that web site classification does not require large numbers of web pages per site for making more accurate predictions.

Ester's crawler and other crawlers that use a static initial set of example documents for classification, are very dependent on the quality of the initial training data. Sizov et al. [9] have built a focused crawler that aims to overcome the limitation of the initial training data. It identifies "archetypes" from the documents and uses them for periodically re-training the classifier. This way the crawler is dynamically adapted based on the most significant documents found so far. Two kinds of archetypes are considered: good authorities as determined by employing the link analysis algorithm proposed by Kleinberg [6], and documents that have been automatically classified with high confidence using a linear Support Vector Machine (SVM) classifier.

All of the focused web crawlers mentioned earlier use example documents to determine the relevance. Another approach is to use keywords. Chakrabarti et al. [10] describe a focused crawler that finds hub- and authority-pages within a subject which is given by a few keywords. A hub is a web page that points to many web pages about the subject, while an authority-page contains much information about the subject. The crawler evaluates the text around the link-tag to decide whether the link seems interesting.

## 3 Ontologies

In this chapter we will give a short introduction to ontologies and some of the mostly used ontology languages. Our choice of ontology language is also explained here. At the end of this chapter we explain the concept of ontology-focused crawling.

### 3.1 Introduction to ontologies

According to Wikipedia [11] the term *ontology* is an old term from the field of philosophy, where it means the study of being or existence. This term is also used in the field of computer science, where it has a slightly different meaning. In the field of computer science, an ontology is the result of an attempt to create a rigorous conceptual schema about a domain. Typically an ontology is a hierarchical data structure containing relevant entities, relationships and rules within a specific domain.

Tom R. Gruber [12] defines an ontology as a *specification of a conceptualization*. An ontology is a formal description of concepts and the relationships between them. Definitions associate the names of entities in the ontology with human-readable text that describes what the names mean. The ontology can also contain rules that constrain the interpretation and use of these terms.

An ontology can be used to define common vocabularies for users who want to share knowledge about a domain. It includes definitions of concepts and relations between them, and is written in a language that can also be interpreted by a computer. Ontologies can be used to share common understanding of the structure of information, enable reuse of domain knowledge, separate domain knowledge from operational knowledge and analyze domain knowledge. [13]

### 3.2 Ontology languages

In order to ensure that a computer understands an ontology, the ontology must be represented in a computer-readable language. There exist several different ontology languages which can be used for this purpose. In this chapter we will present some of the most distinguished ontology languages.

#### 3.2.1 RDF

RDF (Resource Description Framework) [14] is a standard developed by W3C, intended to be a universal format for data on the Web. RDF is based on XML and can be used to represent information about resources in the World Wide Web. Particularly it can be used to represent metadata about Web resources, like modification date, author, title or copyright information. RDF aims to make it easier for agents and applications to exchange information by providing interoperability of data.

### **3.2.2 OWL**






OWL (Web Ontology Language) [15] is a W3C recommendation designed for use by applications that need to process the content of information instead of just presenting the information to humans. OWL makes it easier for computers to interpret Web contents than that supported by XML, RDF and RDF Schema. This is done by providing additional vocabulary along with formal semantics. OWL has three increasingly expressive sublanguages: OWL Lite, OWL DL and OWL Full.

### **3.2.3 Topic Maps/XTM**

XTM (XML Topic Maps) [16] is created by the TopicMaps.Org Authoring Group (AG), formed in 2000 by an independent consortium named TopicMaps.Org. Topic Maps was first fully described in the ISO13250 standard which is SGML and HyTime-based, but has now been developed into the XML-based XTM 1.0. Topic Maps is designed for describing knowledge structures and associate them with information resources. It is well suited for knowledge management and provides powerful new ways of navigating large and interconnected information resources. [17] Topic Maps tries to solve the findability problem of information, i.e. how to find the information you are looking for in a large body of information. It can also be used for content management, web portal development, enterprise application integration (EAI), and is also described as an enabling technology for the semantic web. [18]

## **3.3 Selected ontology language**

There are several different ontology languages that could have been used in this project, but we decided to use Topic Maps. One of the reasons why we selected Topic Maps is that InterMedium has much experience with Topic Maps. Another reason is that, as shown in Figure 3.1, Topic Maps is well supported by both the commercial and open source community. Topic Maps is an established standard that has been used for several years and many tools have been developed to support Topic Maps.

	<b>KIF/OKBC/ CG/CyclL</b>	<b>UML</b>	<b>Topic Maps / XTM</b>	<b>RDF(S)</b>	<b>DAML + OIL</b>	<b>OWL</b>
<b>Description</b>	<i>Legacy KR Languages</i>	<i>Universal Modeling Language</i>	<i>Topic Maps/XML Topic Maps</i>	<i>Resource Description Framework</i>	<i>DARPA ML + Ontology Inference</i>	<i>Web Ontology Language</i>
<b>Governance</b>	 / Other					
<b>Years since proposed</b>	>5	>5	>5	>3	3 or less	3 or less
<b>Commercial Support (as a KRL)</b>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/> <input type="checkbox"/>	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	<input type="checkbox"/>
<b>Open Source Support</b>	<i>Yes</i>	<i>Yes</i>	<i>Yes</i>	<i>Yes</i>	<i>Yes</i>	<i>Coming</i>

2 or less vendors       10 or less vendors  
  5 or less vendors        > 10 vendors

© Copyright 2001-2003 TopQuadrant Inc. All rights reserved. Expedition Workshop #22 slide 21

Figure 3.1 TopQuadrant's comparison of some ontology languages in 2003. [19]

In our project we have used several Topic Maps tools in order to develop and use ontologies. Some of the tools we have used will be presented in the following chapters.

### 3.3.1 TM4J – Topic Maps for Java

TM4J [20] is an open source tool for creating and presenting topic maps. The project is divided into four sub projects. Two of these sub projects have been used in our project: the TM4J Engine and TMNav.

The TM4J Engine is a topic map processing engine. It is the core of the TM4J project and provides an API which makes it possible to create and edit topic maps in Java applications. It also gives support for importing and exporting topic maps to and from XTM files. Topic maps can be saved in memory or persistently stored in an Ozone [21] object-oriented database or in a relational database by using Hibernate [22].

TMNav is a Java application for browsing topic maps. Figure 3.2 shows a screenshot of TMNav and how it can display a topic map in tree view and graph view. One problem with TMNav is that the graphical topic map representation only shows the selected topic and the ones directly connected to it. It is therefore not possible to view the entire topic map at once.

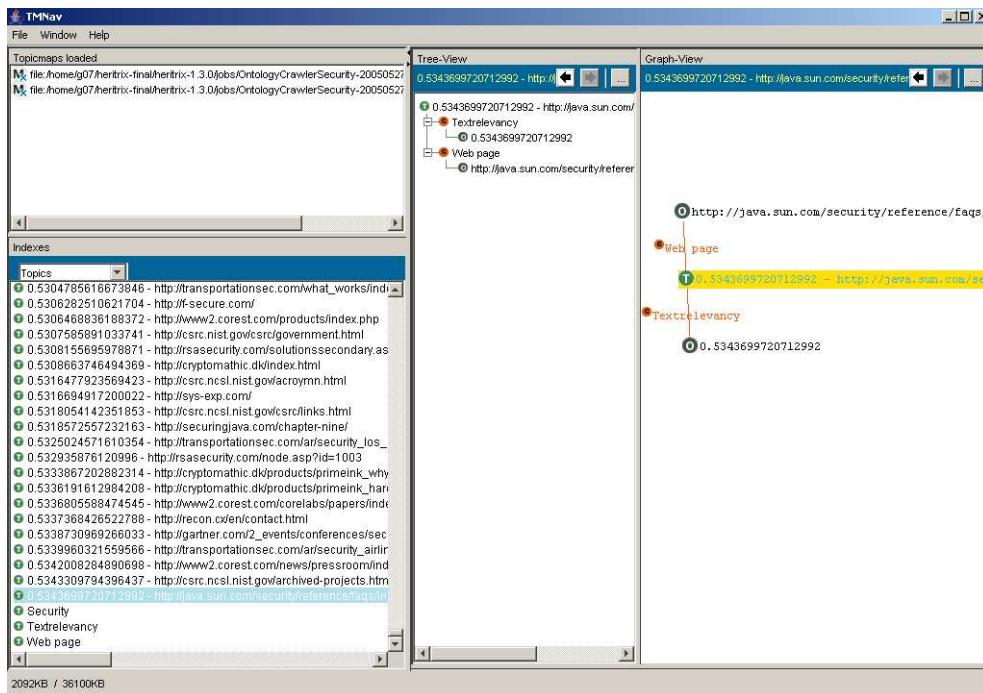


Figure 3.2 Screenshot from the TMNav application

### 3.3.2 Ontopia Omnigator

Ontopia Omnigator is a web-based topic map navigator created by Ontopia [23]. It can be used to browse topic maps in both tree view and graph view. Omnigator also has the ability to merge topic maps on the fly, search in topic maps and export topic maps. In our project we have used Omnigator to view ontologies graphically. Figure 8.2 shows an ontology viewed in Omnigator. Omnigator can show all topics in an ontology at the same time, in contradiction to TMNav which only shows one topic and the nodes connected directly to it.

## 3.4 Ontology-focused crawling

Ontologies can be used in focused crawlers. An ontology-focused crawler uses an ontology to describe the area of interest, in the same way as a search in a search engine uses a list of keywords to describe the area of interest. A problem with standard keyword based search queries is that it is difficult to express advanced search queries. By using ontologies it is possible to express richer and more accurate queries. Ehrig et al. [3] discuss how a focused crawler can find relevant web pages by letting the user make an instantiated ontology. The system has an ontology that describes the area in which the search will be performed, and the user enters different parameters to say what should be weighted in the search. Then the program scans the web for pages containing text that describes the area given by the ontology.

## 4 Evaluation of existing crawlers

This chapter starts with an evaluation of a few chosen open source Java web crawlers. They have been picked as candidates for crawlers that we can use as a basis for our prototype. We give a short description of each crawler, and our own opinion about the ability to use that crawler for our purpose. The evaluation is based on the knowledge we had about the crawlers at the time we tested them. After the individual description and evaluation, the difference between developing our own crawler entirely from scratch, and extending an existing crawler, is discussed.

### 4.1 Nutch

The purpose of Nutch [24] is to promote public access to search technology without commercial bias. Nutch is a transparent alternative to commercial web search engines and makes it possible for everyone to see how their search algorithms work.

Nutch is primarily a breadth-first crawler. It downloads all the web pages it finds, indexes them in a database and provides a web-based user interface for searching the results. The relevance-score of the results is calculated using keyword search and link analysis.

Nutch can do both Intranet crawling and whole-web crawling. Each step of the crawling procedure must be started manually by the user. First the user must inject some seed URLs. Then the user can generate a segment of the URLs and start fetching the pages. When the fetching procedure has been completed the user can update the database with the newly downloaded pages from the segment. Then the user can run an analysis of the database in order to analyze the link structure between the pages. After this the user can generate a new segment with the top-scoring pages from the database and do another round of fetching and analyzing.

When the user is finished downloading pages, all the segments must be indexed in order to make the results searchable. The web application in Nutch can then be used to do keyword-based search in the indexes.

Positive:

- Relatively well documented (API, good web page with usage information and a developer's section)
- Has support for plug-ins.

Negative:

- When we started testing Nutch we had difficulties making the crawler work.
- Nutch is quite complex and it is difficult to understand how all the parts work.
- No stable releases exist yet.



## 4.2 Heritrix

The Heritrix crawler [25] has a modular design, and is easy to extend. It has a powerful web-based user interface where the user can choose which modules to use in his crawl, and the settings for the modules and the crawl job in general. By developing and adding our own modules, we can make the crawler ontology-focused. Heritrix has had some problems with high memory usage, but the developers are working on it, and the latest versions allegedly show improvement. It is a living project, and the crawler is still under development.

In one of the first tests we ran on Heritrix, the page rate was very low. The first 13 minutes had an average of 1.94 pages downloaded per second (i.e. 116.4 pages per minute, 6984 pages per hour). We had disabled a module to make the crawler work, and we suspected that this was the reason for the low download rate. It took 30 seconds to process each URI, and this is a long period of time. After the evaluation period was over, we discovered that the relatively slow crawling was caused by the disabled module. The module did not work due to a bug in the dnsjava [26] library that the module used. This problem, and the solution, is described in chapter 9.1.2.

Positive:

- Heritrix is very modular and extendable. It is designed to make it easy to include new modules, and replace existing modules with different ones.
- It is well documented. In addition to the API, there is a Users Manual and a Developers Manual, as well as many forums.
- The Internet Archive organization [27], which has developed Heritrix, has much experience with web crawling.
- It has been tested extensively. It is used regularly by Internet Archive [27].
- The crawler is highly configurable and is polite towards servers and it obeys the robots.txt protocol.
- The web user-interface is user friendly and easy to understand.

Negative:

- When we tested Heritrix it was very slow because of the problems with dnsjava.
- The 1.2.0 version of Heritrix, which was used in the first tests, used quite much memory and got `OutOfMemoryExceptions` when it had run for a little while, but the Heritrix developers are working on this and the more recent CVS versions show improvement on the memory issues.

## 4.3 Weblech

Weblech seems to be a program mainly for downloading or mirroring a web site, but according to the project web site [28] it is possible to configure it to ‘spider the Whole Web’. Still it will probably take a lot of work to turn this crawler into something we can use in our project. The Weblech project is also in a pre-alpha state, with a latest release version of 0.0.3, so it is not at all finished. The project web site states that the main features work, but that the GUI has not been developed. The latest news posted on the page informs

us that version 0.0.4, which will include the GUI, is being developed, but this post is from June 12<sup>th</sup> 2004, so it is almost a year old.

Positive:

- The crawler is multithreaded, written in Java and open source, but so are most of the other crawlers we have evaluated.

Negative:

- This crawler seems to be specialized on a different task than what we need in our project.
- The project is in a pre-alpha state (version 0.0.3).

#### 4.4 WebSPHINX

WebSPHINX consists of two parts: A crawler workbench and a class library that provides support for writing web crawlers in Java. The workbench can show a graphical representation of web sites and links found, save pages to disk for offline browsing, show/print different pages in a single document, extract text matching a pattern from a collection of pages, and develop a custom crawler that processes pages the way you want. [29]

The WebSPHINX class library offers several features [29]:

- Multithreaded Web page retrieval in a simple application framework
- An object model that explicitly represents pages and links
- Support for reusable page content classifiers
- Tolerant HTML parsing
- Support for the robot exclusion standard
- Pattern matching, including regular expressions, UNIX shell wildcards, and HTML tag expressions. Regular expressions are provided by the Apache jakarta-regexp regular expression library
- Common HTML transformations, such as concatenating pages, saving pages to disk, and renaming links

If we are going to use this crawler as a basis, we would probably have to use the class library part of WebSPHINX. The workbench alone does not seem to be powerful enough, but the features of the class library look promising. We would have to write a Java web crawler that utilizes the class library to get the standard crawler functions done, like fetching web pages, parsing HTML, honoring the robot exclusion standard, and so on. The part of the crawler that parse the ontology and focus the crawl according to it would then have to be created in addition.

Positive:

- WebSPHINX includes a Java library that gives support for developing a web crawler in Java. This could be useful if we decide to develop a crawler from scratch, and parts could also maybe be used if we decide to extend a different crawler.

Negative:

- The WebSPHINX crawler does not seem to be powerful enough for our project.

## 4.5 J-Spider

J-Spider [30] is a configurable and customizable web spider engine. J-Spider is primarily designed for crawling a single web site and has a text based UI.

It can be used:

- to check a site for errors (internal server errors, ...)
- for outgoing and/or internal link checking
- to analyze a site structure (creating a site map, ...)
- to download complete web sites

Positive:

- It is modular and extensible using plug-ins
- Well documented. 121 pages in user manual.

Negative:

- Designed for crawling single web sites
- Under development. No stable releases
- Only text based UI

## 4.6 HyperSpider

HyperSpider [31] is designed to evaluate the link structure of a web site. It can import/export to/from databases and CSV-files. It has a GUI which can be used to analyze the link structure of a web site. HyperSpider can also export the link structure to several formats, e.g. HTML, XML Topic Maps (XTM) and RDF.

Positive:

- Graphical user interface
- Graphical presentation of link structures
- Good support for analyzing link structures
- Good support for exporting link structures to different formats, e.g. XTM and RDF

Negative:

- Only designed for evaluating the link structure of single web sites
- Low modularity and extensibility
- Little or no documentation
- No development activity since 29.08.2003

## 4.7 Arale

Arale [32] is a Java based web crawler primarily designed for downloading files from a single web site. It can also be used to render dynamic pages into static pages. The crawler has a text based UI.

Arale is a very simple crawler, and consumes a lot of memory while crawling pages. Apparently the software has some issues concerning memory usage.

Positive:

- Simple and easy to use
- Can be filtered to download certain file types

Negative:

- Designed for crawling single web sites
- Very simple code, low extensibility
- No development activity since 2001

## 4.8 Extending an existing crawler or developing a new crawler

In this chapter we will discuss the positive and negative aspects of extending an existing web crawler, and developing a new crawler from scratch.

### 4.8.1 Extending an existing web crawler

By extending an existing web crawler, we mean to build extra modules, or alter existing parts of an already existing rather complete web crawler.

One of the main advantages of extending an existing crawler is that we have to do less programming. There are a lot of more or less basic features that are necessary to make the crawler work, that are not actually a part of the relevance algorithms. We could save a lot of time by reusing this functionality from an existing crawler instead of implementing it ourselves. This time could be used to improve the parts of the crawler that are more relevant to our thesis. The basic features reused from an existing crawler will most likely also be better than equivalent features implemented by us. This means that extending an existing crawler will lead to a more robust and polite prototype.

On the other hand, to be able to build extensions for a crawler, it is necessary to understand how the crawler is designed, how it works and how we can extend it. Some of the time saved by reusing code, will be used to gain this understanding. Still, if the extendable crawler is reasonably well documented, getting the necessary understanding will be less time consuming than developing a crawler from scratch. The crawler to extend will most likely have been designed with a different or more generic area of application in mind. This means that an extended crawler will have lower performance than a web crawler developed from scratch solely for the purpose of doing ontology-focused crawling.

### 4.8.2 Developing a web crawler from scratch

When we talk about developing a web crawler from scratch, we mean that instead of building an extension to an existing complete web crawler, we could design our own web crawler using class libraries, modules from other crawlers, and modules implemented by ourselves. Developing a web crawler from scratch does not mean that we must write all the code ourselves.

If we choose this approach, we will have a lot of freedom to design the crawler to make it best fit our needs. We would not be limited by a crawler that someone else has made for a different purpose.

But, as mentioned before, developing our own crawler will take more time, and be more difficult. The crawler would probably have to be made as simple as possible to let us complete our project in time. This could cause the crawler to be more unstable and less polite than an extended crawler.

## 4.9 Evaluation

We decided to extend an existing crawler. One of the reasons was that the crawler is only used to test algorithms and principles, and it is therefore desirable to use as little time as possible on developing the crawler itself. Time is also always a scarce resource. Spending less time on developing a crawler will free time that can be used to choose and develop algorithms on how the crawler can use ontologies to become focused. Another important factor was that it seemed feasible actually extending the most promising crawler, named Heritrix.

After we had decided to extend an existing crawler, we chose to extend Heritrix. The evaluation had led us to believe that Heritrix and Nutch were the most promising alternatives. Because of this we spent more time testing these crawlers than the rest. After these tests, Heritrix seemed to be easiest to extend, easiest to use, and it also seemed to be best documented. These are the main reasons why we selected Heritrix.

## 5 Description of Heritrix

It is stated in the Heritrix developer documentation [33] that the Heritrix Web Crawler is designed to be modular. Adding a new module with extra functionality is easy, and it is possible to choose which modules to use at runtime in the web user interface. Figure 5.1 shows an overview of the most important parts of the Heritrix crawler. The parts shown in the figure will be explained in this chapter. Most of the information in the following subchapters is fetched from the Heritrix developer documentation [33] and *An Introduction to Heritrix* [34], and these documents are recommended if a more detailed presentation of Heritrix is required.

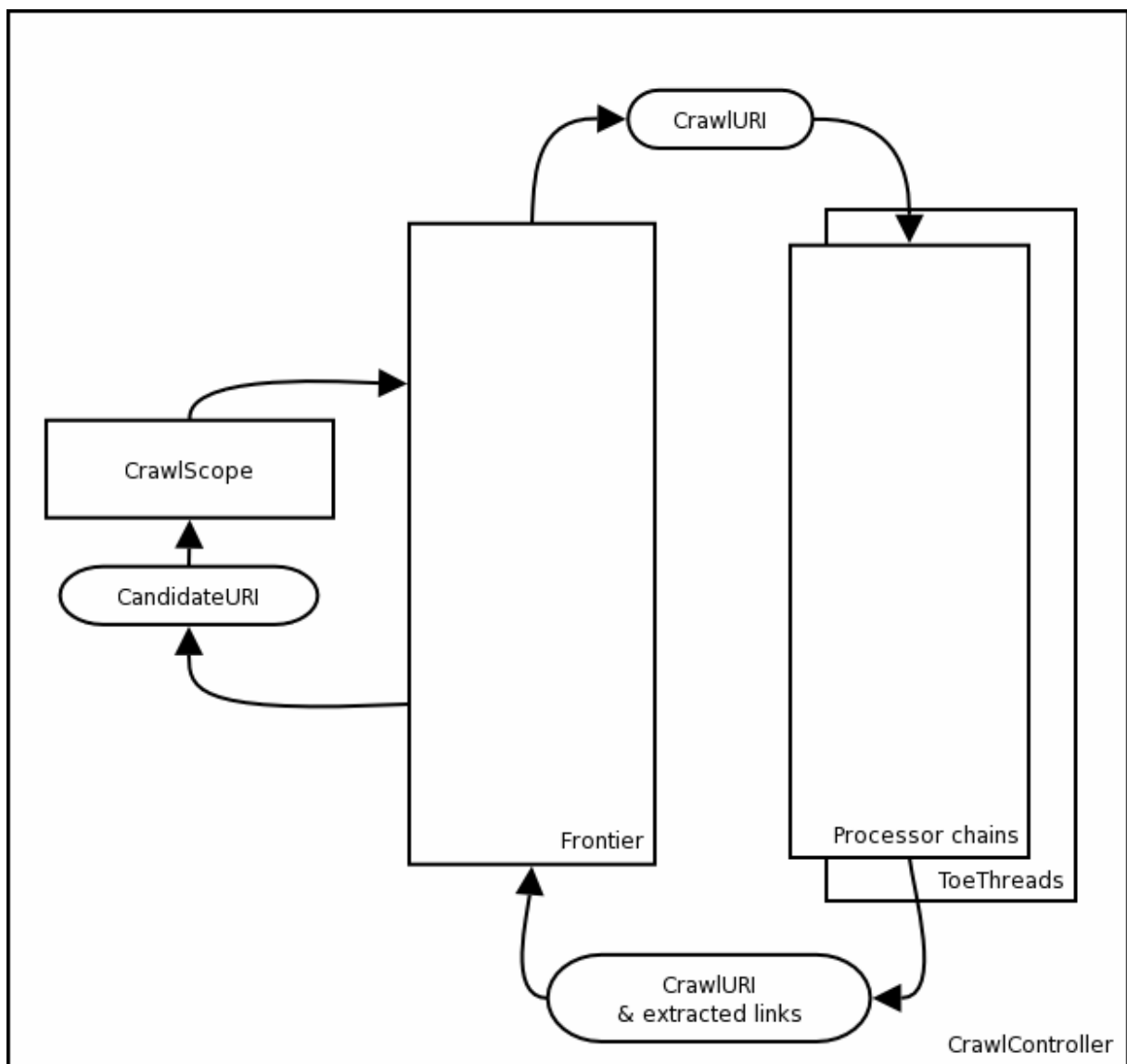


Figure 5.1 Overview of Heritrix. [33]

### 5.1 Frontier

The Frontier maintains the state of the crawl. Among other things it contains the queue of URIs that have not yet been downloaded and a list of visited URIs to prevent the crawler

from downloading pages unnecessarily. When a ToeThread finishes processing an URI, it delivers discovered links to the Frontier, and then asks the Frontier for another URI. The politeness is also controlled by the Frontier. The Frontier has a queue for each domain to distribute the load on different servers.

## 5.2 ToeThreads

The Heritrix Web Crawler is multithreaded in order to be more effective. The threads that do the real work are called ToeThreads. It is possible to configure how many of these a crawl should have. The ToeThreads ask the Frontier for a new URI, and sequentially run it through the processors the crawler is configured to use.

## 5.3 CrawlURI and CandidateURI

All the URIs are represented by a CrawlURI instance. The ToeThreads get a CrawlURI object from the Frontier, and this object contains the URI. The different processors use this object to move information to the succeeding processors. The FetchHTTP processor downloads the web page, and stores the downloaded data in the CrawlURI instance. When the same CrawlURI-object enters the ExtractorHTML processor the downloaded data is fetched from the CrawlURI-object. A CandidateURI is created when a new URI is discovered. If it is accepted in the Frontier the CandidateURI is turned into a CrawlURI.

## 5.4 CrawlScope

A CrawlScope defines which URIs are allowed to be scheduled into the Frontier. Basically it is a filter which looks at the information in a CrawlURI or a CandidateURI and decides whether it should be crawled or not.

## 5.5 Processor chains

The processors are organized into different processor chains according to their functionality. For instance all link extraction processors are part of the Extractor processing chain. A processor chain is the same as a processing chain. The Heritrix documentation is a bit inconsistent and alternates between these two terms.

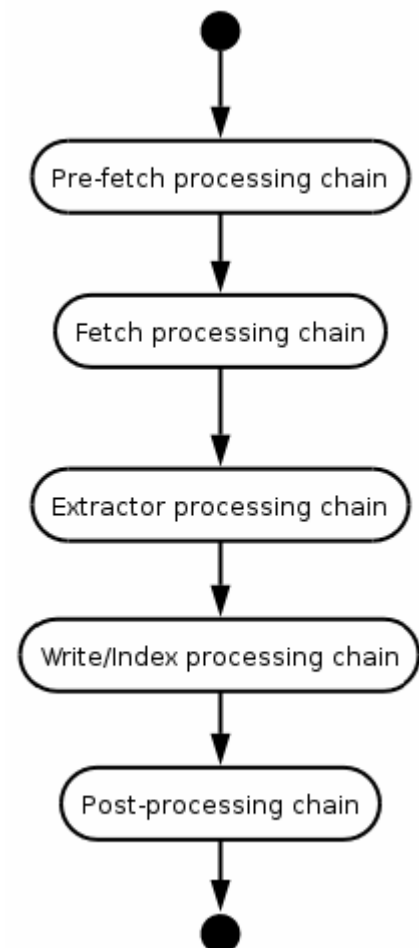


Figure 5.2 Processor chains. [33]

## 6 Focusing algorithm and search strategy

This chapter explains the algorithms used in our prototype and some other relevant algorithms.

### 6.1 Ontology based comparison of documents

The algorithm in [35] has been developed by Vladimir Oleshchuk and Asle Pedersen. Its purpose is to measure similarity between documents. The idea is that the degree of similarity depends on the context and the existing knowledge of the agent performing the comparison. The context/knowledge is represented ontologically. Instead of comparing the document contents, the linkage between ontology concepts and document contents is compared.

The algorithm consists of two parts. The first part aims to generate an ontological “footprint” of a document. The footprint is a subontology of the main ontology, describing the linkage between the document contents and the ontology.

The second part is used to compare two footprints of two different documents. The algorithm can determine on which abstraction level the two documents are similar.

In the beginning of our project, some ideas from this algorithm were used as a starting point when we developed our relevance algorithm.

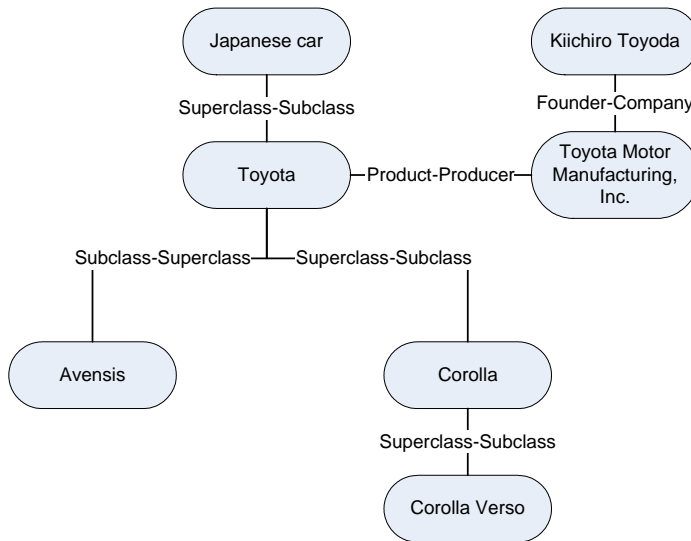
### 6.2 Our relevance algorithm

One of the ideas behind an ontology-focused web crawler is that the user often has some initial knowledge about the area of interest. The prototype developed in this project can take this knowledge in the form of a topic map. The goal of the relevance algorithm used in the prototype is to determine how relevant a web page is in relation to the topic map. The algorithm is inspired by the relevance computation algorithm described in [3].

The names of all the topics in the ontology are extracted, and these words are the basis for the relevance calculation. The user has the possibility to select one or more focal topics in the topic map. These topics will get a higher weight, and if they are found on a web page they will affect the relevancy of the page more than words with lower weights. There are four different weight classes, and the weight of these can be set by the user in the web administrative console. The first class includes the focal topics themselves, and the default weight is 1.0. The second group includes all the topics directly related to one or more focal topics through a Superclass-Subclass association. This group has a default weight of 0.8. The next class embraces all the topics directly related to one or more of the focal topics through any other association than a Superclass-Subclass association. The default weight is 0.5. Topics that are not directly connected to the focal topics constitute the last weight class. These topics get a weight of 0.1 as default. The weight of all the topics in the ontology is calculated once for each crawl job, so changing the weights for the different weight classes during a crawl will not have any effect. If a topic should belong to more



than one of these weight classes, it will get its weight assigned from the class with the highest weight.



**Figure 6.1** A very simple TM about Toyota.

Figure 6.1 shows a very simple example of a topic map about Toyota. It shows that Toyota is a subclass of Japanese car, and the superclass of Avensis and Corolla. The topic map states that TMM Inc. produces Toyotas, and that Kiichiro Toyoda was the founder of TMM Inc. Although simple, this TM contains enough different types of associations to show how the weights of the topics are calculated in the prototype. The table below shows the weights of the topics, given that the default weights are used. The values in the center column presuppose that Toyota is chosen as focal topic, while the rightmost column shows the values as they would be if TMM Inc. and Corolla were set as focal topics.

Topic	Weight (Focus: Toyota)	Weight (Focus: TMM Inc. and Corolla)
Japanese car	0.8	0.1
Kiichiro Toyoda	0.1	0.5
TMM Inc.	0.5	1.0
Toyota	1.0	0.8
Avensis	0.8	0.1
Corolla	0.8	1.0
Corolla Verso	0.1	0.8

**Table 6.1** Weights of the topics from the topic map in Figure 6.1

The weights in Table 6.1 are only used to dictate which part of the topic map is most important. They say nothing about the relevancy. The RelevanceCalculator processor uses a version of the TFIDF (Term Frequency Inverse Document Frequency) [2] algorithm to calculate the relevance of a web page.

The classical TFIDF algorithm [2] can be described with the following equation:

$$\omega_{fd} = tf_{fd} \log\left(\frac{D}{df_f}\right)$$

where  $\omega_{fd}$  is the weight of the feature (term)  $f$  in document  $d$ ,  $tf_{fd}$  the raw frequency of feature  $f$  in document  $d$ ,  $D$  the total number of documents in the training set, and  $df_f$  is the number of documents containing the feature  $f$ . The original TFIDF is explained in detail in [36].

The TFIDF values are found by multiplying Term Frequency with Inverse Document Frequency. TF is the number of occurrences of a specific term in a particular document. The IDF part is found by dividing the total number of documents by the number of documents containing the term, and it increases the TFIDF value for more rare terms. A term that only occurs in 2 % of the documents gets a higher IDF value than a term that occurs in for instance 95 % of the documents.

In the prototype described in this thesis a maximum normalized version of TFIDF has been used, that can be described by the following equation:

$$\omega_{fdm} = \frac{tf_{fd}}{TF_{fd}} * \log\left(\frac{D}{df_f}\right)$$

Here,  $\omega_{fdm}$  is the weight of the feature (term)  $f$  in document (web page)  $d$ ,  $tf_{fd}$  the number of times the feature  $f$  occurs in document  $d$ ,  $TF_{fd}$  the number of occurrences for the feature that occurs most frequently in this document,  $D$  the total number of processed web pages, and  $df_f$  is the number of web pages containing the feature  $f$ .

A TFIDF value is calculated for each term in the topic map that occurs in the document.

Because the crawler is focused, the relevancy of the web page is needed to decide whether to follow the links on the page or not. This means that it is not possible to do the relevance calculation after all web pages have been downloaded. The calculation has to be done during the crawl, so the IDF values will be based on different document sets.

To get an overall relevancy of a web page, the maximum normalized TFIDF values of the terms found on the page needs to be combined. In the algorithm described in [3] the scores of the terms from the ontology are simply summarized to get a final relevance for the document. In our prototype this is achieved in a similar way, according to the following equation:

$$relevancy_d = \frac{\sum_f (\omega_{fdm} w_f)}{\sum_f w_f}$$

where  $relevancy_d$  is the overall relevancy of the web page  $d$ ,  $\omega_{fd}$  is the TFIDF value of the term  $f$  in web page  $d$ , and  $w_f$  is the weight of the term  $f$ . Note that  $w_f$  is not the TFIDF weight, but the weight used for defining the focus of the topic map. The maximum normalized TFIDF values are multiplied by the ontology weights. This is done for all the different terms in the document that occurs in the topic map, and the results are summarized. We normalize by dividing by the sum of the ontology weights of all the topics in the topic map. This sum is the theoretical maximum value of the dividend, and so

the highest possible relevance value for a document is 1. It is this normalization part that differs from the algorithm in [3].

### 6.3 Link analysis

Pages on the Web are heavily interconnected and contain a lot of cross-references. Analysis of the link patterns on the Web can be done with link analysis algorithms. Several link analysis algorithms have been successfully used to discover authoritative information resources on the Web. One well-known algorithm is the Kleinberg HITS algorithm [6]. This algorithm gives a page a high “authority” weight if it is linked to by many pages with high “hub” weight, and gives a page a high hub weight if it links to many authoritative pages. Another popular algorithm is the PageRank [37] used in the Google search engine. PageRank is one of the methods used by Google to determine a page’s relevance or importance.

In order to ensure that our prototype only follows links that are considered important, we have included a link analysis module from Nutch, called WebDB. The WebDB module is a web database that can save and analyze the graph structure of web pages. According to Khare et al. [38] the link analysis algorithm used in Nutch is similar to the PageRank algorithm. The WebDB database contains one table called “Page” and one table called “Link”. The Page table contains information about all the web pages the crawler has discovered, while the Link table contains information about the link connections between the pages. When the link analysis is started, Nutch uses the link structure to calculate a score for each page in the database. The page score gives an indication of the importance or authoritativeness of the web page.

## 7 The prototype

Our prototype is based on the Heritrix web crawler. Link analysis functionality is achieved through the WebDB module from the Nutch crawler. The prototype takes an ontology and a list of URLs as input. The ontology delimits the area of interest, and the URLs are used as seeds (starting URLs). Each downloaded URI is passed in turn to the processors defined to be used in the crawl. Each downloaded web page is parsed with the NekoHTMLParser processor, and the RelevanceCalculator analyzes the contents extracted by the parser and determines a relevance value of the page in relation to the input ontology.

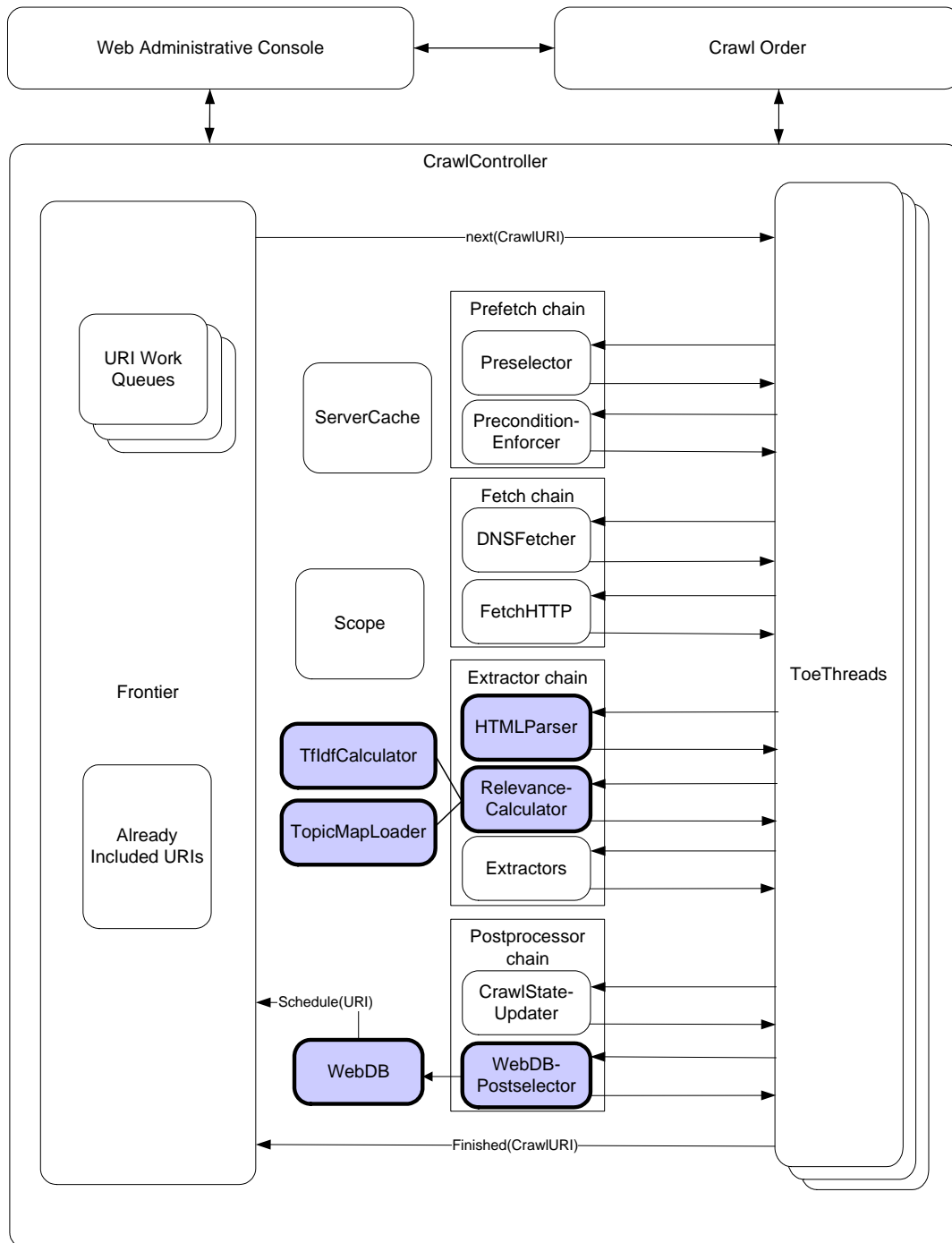


Figure 7.1 Overview of Heritrix. The emphasized modules are the modules we have added.

Figure 7.1 shows an overview of the modules in Heritrix and how they are connected. The emphasized parts are the modules we had to add to Heritrix in order to turn it into an ontology-focused crawler. In the following chapters we will explain how these modules work and how we have created the prototype.

	Name	Function
Prefetch	Preselector	Offers an opportunity to reject previously scheduled URIs not of interest.
	PreconditionEnforcer	Ensures that any URIs which are preconditions for the current URI are scheduled beforehand.
Fetch	FetchDNS	Performs DNS lookups, for URIs of the “dns:” scheme.
	FetchHTTP	Performs HTTP retrievals, for URIs of the “http:” and “https:” schemes.
Extractors	<b>NekoHTMLParser</b>	<b>Parses the content from the current URI.</b>
	<b>RelevanceCalculator</b>	<b>Calculates the relevance of the web page in relation to the ontology.</b>
	ExtractorHTTP	Discovers URIs in the HTTP header.
	ExtractorHTML	Discovers URIs inside HTML resources.
	ExtractorCSS	Discovers URIs inside Cascading Style Sheet resources.
	ExtractorJS	Discovers likely URIs inside Javascript resources.
	ExtractorSWF	Discovers URIs inside Shockwave/Flash resources.
PostProcessors	CrawlStateUpdater	Updates crawler-internal caches with new information retrieved by earlier processors.
	<b>WebDBPostSelector</b>	<b>Adds the links extracted from the current URI to the WebDB, and if the queue in the Frontier is too small, it runs link analysis on the WebDB contents, extracts the URIs with highest page score and schedules them into the Frontier.</b>

Table 7.1 The processors in Heritrix grouped by processor chain. The emphasized rows describe the modules we have added.

## 7.1 NekoHTMLParser

The processor called NekoHTMLParser uses the NekoHTML [39] parser to extract the web page content from the downloaded HTML code. The NekoHTML parser is also used by the Nutch crawler. It is easy to configure the parser, by telling it what to do with the different tags. The parser can either keep the tag, remove the tag, or remove the start and end tag as well as everything between them. The last feature is especially effective to remove all scripts and styles from HTML code. The default is to just remove the tags.

## 7.2 RelevanceCalculator

The RelevanceCalculator’s job is to determine the relevance of the downloaded web pages in relation to the ontology defining the area of interest. It is dependant on the

NekoHTMLParserProcessor. If the parser has not extracted the web page contents from the HTML code, the RelevanceCalculator can not evaluate the relevancy of the page.

### 7.2.1 initialTasks()

The initialTasks method of a processor is called when a crawl is started. In the RelevanceCalculator this initiation includes the following tasks.

First the topic map is loaded from a XTM file with TM4J. In our crawler TM4J uses a hibernate backend with a MySQL database in the bottom to keep the topic map. The name of the XTM file is given by the *topicmap-filename* attribute. After this task has been completed, all the base names of the topics are extracted from the loaded topic map. The next thing that happens is that the list of focal topics (*focal-topic* attribute) and the defined weights of the weight classes (*focus-weight*, *taxrel-weight*, *otherrel-weight* and *rest-weight* attributes) are read from the crawl order. These attributes are then used to assign a weight to each of the topics. For more detailed information on the relevancy algorithm see chapter 6.2. Finally, all the extracted base names are stemmed using the Snowball module from the open source full text indexing engine Lucene [40]. We have configured the Snowball module to use the Porter stemming algorithm.

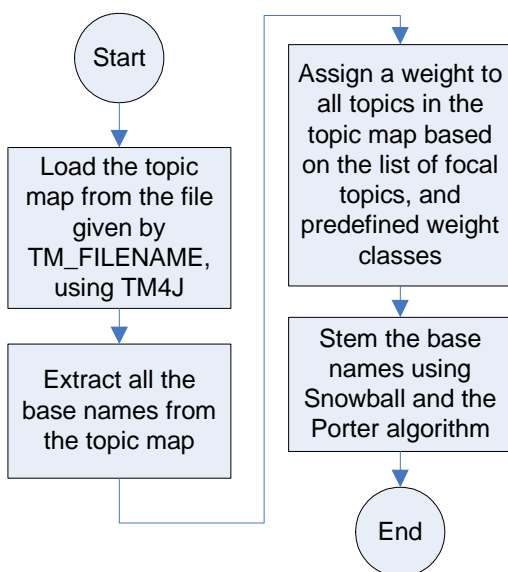


Figure 7.2 Flowchart for initialTasks() in RelevanceCalculator.

### 7.2.2 innerProcess()

The innerProcess method is called for each downloaded web page. Error pages, robots.txt's and URLs that are not http or https are filtered away in the first part of the method. Then the text content of the web page is fetched from the variable where the NekoHTMLParserProcessor stored it. The words in the content are stemmed using the same algorithm as the one used on the base names in the initialization. After this the TFIDF value is calculated for all the topic map base names that occur in the content of the web page. All the TFIDF values are then combined to get an overall relevancy of the web page. For more information about the TFIDF algorithm used see chapter 6.2.

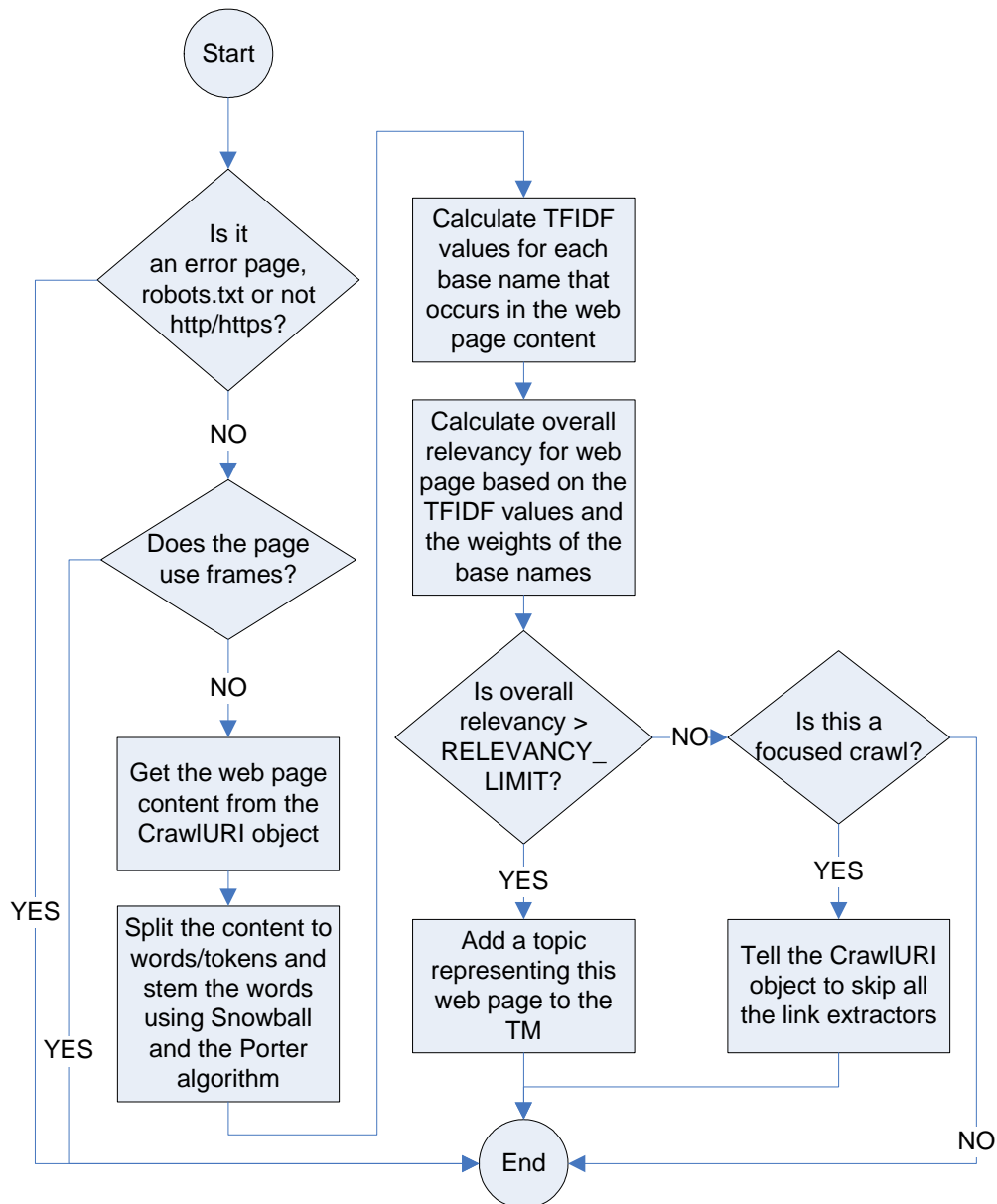


Figure 7.3 Flowchart for `innerProcess()` in `RelevanceCalculator`.

The following attributes can be altered to configure how the `RelevanceCalculator` works.

- topicmap-filename**  
 This attribute holds the filename of the XTM-file that contains the topic map defining the area of interest. The crawler will look for this file in the topicmaps folder in the Heritrix root folder.
- focal-topics**  
 This is a comma separated list with the id(s) of the focal topics in the topic map. These topics will get a higher weight when deciding the relevancy of a web page. The weights are defined in the next four attributes. All the weights are declared as doubles, and their value should not be less than 0 or more than 1.
- focus-weight**  
 In this attribute the weights of the focal topics are defined.

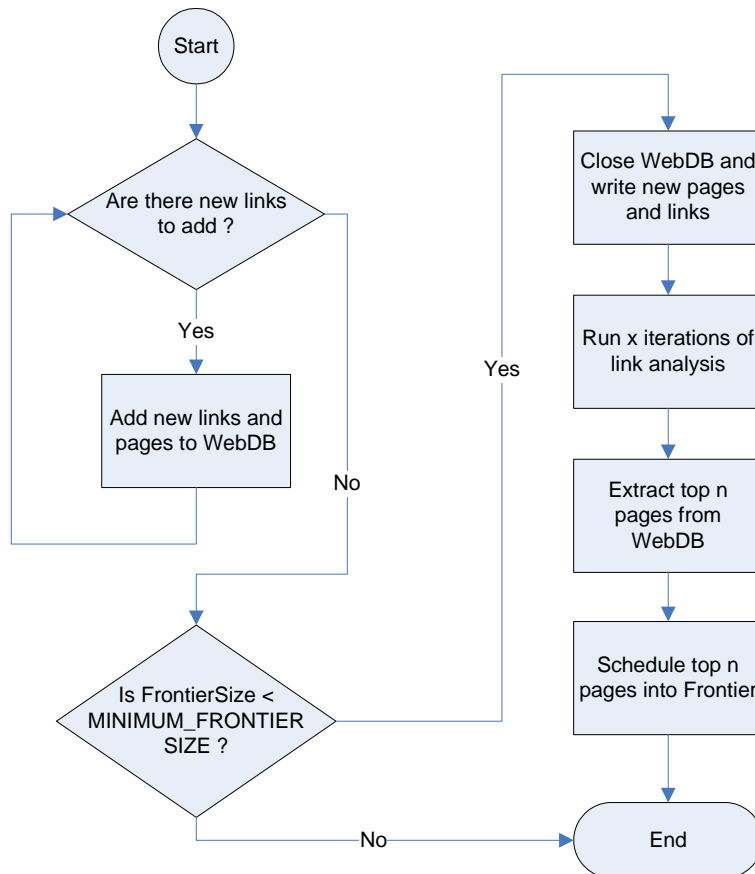
- **taxrel-weight**  
This attribute holds the weight of the topics directly related to the focal topics through a taxonomical association. This means that it is possible to get to this topic from one of the focal topics by traversing only one association of the type Superclass-Subclass.
- **otherrel-weight**  
This attribute holds the weight that should be assigned to all the topics directly connected to the focal topics through any other association type than Superclass-Subclass.
- **rest-weight**  
This attribute defines the weight of all the topics that are neither a focal topic nor directly connected to one.
- **relevancy-limit**  
A web page is not necessarily relevant just because the relevancy value is higher than zero. This just means that one of the words from the topic map occurs in the web page. The relevancy-limit attribute is used to decide the border between relevant and irrelevant pages. Its value should be between 0 and 1. If set to 0, all web pages will be considered relevant, if set to 1, all web pages will be considered irrelevant.
- **focused-crawl**  
This Boolean attribute denotes whether this should be a focused crawl or not. If this value is true, the crawler will only extract links from web pages that are considered relevant by the RelevanceCalculator (i.e. the relevancy of the page is higher than the *relevancy-limit*). If the attribute is set to false, the crawler will function more like a breadth-first crawler, and extract links from all the web pages.
- **harvestrate-logfile**  
This attribute contains the filename of the file that will be used to write harvest rate samples.
- **harvest-samplerate**  
This attribute defines the minimal time between each harvest rate sample, in seconds.

## 7.3 WebDBPostselector

### 7.3.1 Description of WebDBPostselector

The purpose of the standard Postselector in Heritrix is to determine which extracted links and other related information that get fed back to the Frontier. Instead of using the standard Postselector, we have developed our own Postselector called WebDBPostselector. The purpose of WebDBPostselector is to focus the crawler by using link analysis. This module uses link analysis to determine the importance of discovered links, and then follows the most important links first. WebDBPostselector is configurable through module attributes. The module can be configured to add e.g. the 500 most important links. In order to do link analysis WebDBPostselector uses a module from Nutch called WebDB. The database in the WebDB module contains a web graph with all known pages and the links that connect them.





**Figure 7.4** Flowchart for `innerProcess()` in `WebDBPostselector`

The `WebDBPostselector` is part of the post-processing chain in Heritrix. When a page has been fetched by a `ToeThread` the `CrawlURI` object is sent through all the processing chains and at the end to the `WebDBPostselector`. The `WebDBPostselector` receives the `CrawlURI` through the `innerProcess()` method. Figure 7.4 shows the flowchart for `innerProcess()` in `WebDBPostselector`.

When the `CrawlURI` is received the `WebDBPostselector` checks the `CrawlURI` to see if it contains new links. If it contains new links, they are all added to the `WebDB` database. After that, the `WebDBPostselector` checks the size of the `Frontier` queue to see if it is less than the defined minimum size. If the `Frontier` size is not less than minimum, the `WebDBPostselector` is finished and the control returns to the `ToeThread`. If the `Frontier` size is less than minimum, then the `WebDBPostselector` closes the `WebDB` so that the new pages and links are written to `WebDB`. After this, a defined number of link analysis iterations are run in order to calculate a page score for all the pages in `WebDB`. All the pages are then sorted and the pages with the highest score are extracted from `WebDB`. These pages are then scheduled into the `Frontier` and the `WebDBPostselector` is then finished.

As mentioned earlier, the `WebDBPostselector` is based on the standard `Postselector` module in Heritrix. In addition to the standard attributes from the `Postselector`, we have added the following module attributes in order to make the `WebDBPostselector` more configurable:

- **linkanalysis-iterations**  
Number of iterations to run link analysis
- **linkanalysis-topsize**  
This is the number of top pages that will be extracted from the WebDB module and scheduled into the Frontier. The top pages will be the pages with highest page score.
- **minimum-frontiersize**  
This is the minimum number of URLs that should be in the Frontier queue. When the size of the Frontier queue is lower than this minimum value, the WebDBPostselector will start running link analysis and schedule new pages into the Frontier.
- **maximum-hostqueuesize**  
Maximum number of URLs in each host queue. New pages are not added to the Frontier if the host queue already contains the specified maximum number of URLs. This is done to ensure that all the ToeThreads are active and have URLs in their host queues.

### 7.3.2 ModifiedBdbFrontier

To be able to test whether the size of a host queue is less than the defined *maximum-hostqueuesize*, we had to modify the Frontier. The ModifiedBdbFrontier extends the standard BdbFrontier with a method called `getHostQueueSize()`. This method returns the number of URLs in one specific host queue. We created a test in order to ensure that all the ToeThreads are active and have URLs in their host queues at all times. Without this test some of the host queues sometimes became very large and contained many URLs. This resulted in very few active ToeThreads and therefore an ineffective crawler.

### 7.3.3 Modified Page class in Nutch

In our project we have used the WebDB module in Nutch to store information about discovered links and pages. Each new relevant page discovered by Heritrix is inserted as a new row in the page table in WebDB. To be able to extract pages from WebDB and insert them into the Frontier in Heritrix, we had to add some attributes to the Page class in Nutch. The page extracted from WebDB must be transformed into a CandidateURI which can be scheduled into the Frontier. To be able to create the CandidateURI object these extra attributes from the Page object are necessary. Table 7.2 below shows the original attributes in the Page class, plus the attributes we have added.

Type	Name	Description
byte	VERSION	A byte indicating the version of this entry.
UTF8	URL	The URL of a page. This is the primary key.
128bit	ID	The MD5 hash of the contents of the page.
64bit	DATE	The date this page should be refetched.
byte	RETRIES	The number of times we've failed to fetch this page.
byte	INTERVAL	Frequency, in days, this page should be refreshed.
float	SCORE	Multiplied into the score for hits on this page.
float	NEXTSCORE	Multiplied into the score for hits on this page.
<i>Attributes added by us:</i>		
UTF8	FROMURL	The URL of the page where the link to this page was found.
UTF8	PATH	Letters describing the path from the seed to this page.
UTF8	VIACONTEXT	Context of URI's discovery, as per the 'context' in Link.
byte	DIRECTIVE	The type of link this page was discovered via.
boolean	SEED	Whether this page is a seed or not.

**Table 7.2 Attributes in the modified Page class in Nutch**

The attribute called FROMURL might be considered redundant, because the same information is also stored in the Link table in WebDB. The Link table contains information about which pages are linking to which. The reason why we also put this information in the Page object was to ease the task of extracting this information when creating the CandidateURI object.

## 8 Prototype tests

This chapter describes different measures that can be used when evaluating the performance of a focused crawler. It also describes the tests and the input ontology used in the tests. Towards the end of the chapter we present the results of our tests.

All the tests were run on a computer with an AMD Athlon 800 MHz processor and 512 MB RAM, running MandrakeLinux 10.1. The average download speed in the tests was around 500 kb/s. This means that the Internet connection was not a bottleneck.

### 8.1 Performance measures

This chapter presents some performance measures for focused crawlers. The measures have been collected from different articles about other focused crawlers, and are meant to be a list over possible performance measures to use when testing our prototype. Some of the measures also say a lot about the structure of the scanned domain of the Internet.

Chakrabarti et al. [5] use several measures when evaluating their crawler:

- **Harvest rate**

Harvest rate is a common measure on how well a focused crawler performs. It is the number of relevant pages per downloaded page, and shows how well the crawler avoids the irrelevant web pages. This is a good measure because the definition of a focused crawler is to try to avoid irrelevant pages.

$$hr = \frac{r}{p}$$

$hr$  is the harvest rate,  $r$  is the number of relevant pages found and  $p$  is the number of pages downloaded. Preferably, it would be best to let humans evaluate the relevancy of the discovered pages. This is an unrealistic approach, due to the large amount of web pages crawled. Instead we count the number of pages considered relevant by the prototype itself, and use this number to compute the harvest rate. In [5] the same approach has been used. Chapter 4.2 in [5] describes the harvest rate measure in more detail.

- **Crawl robustness**

Two different crawls that focus on the same topic are started with two different seed sets. The overlapping of the web pages found by the two crawls is monitored as the number of downloads increase. The crawl robustness is the number of pages overlapping divided by the total number of pages downloaded. This measure will also depend on the topic the crawl is focused on. The tests done in [5] indicate that crawls done on competitive domains like investing and mutual funds overlap less than crawls focusing on more co-operative topics like cycling.

- **Fraction of acquired pages vs. relevance**

This measure shows how the pages are distributed on different relevance values,

and is therefore dependent on how the relevancy is calculated.

- **Minimum distance from crawl seed**

Some graphs in [5] show how the 100 most relevant pages found in a crawl are distributed on distance in number of links from the closest seed document. This measure can say something about the characteristics of the domain of Internet that concerns the topic the crawler is focusing on.

The measures used in [4] consist of some sort of harvest rate, and in addition the following:

- **Average relevance**

This graph shows how the average relevance value of the downloaded web pages changes as number of downloads increase.

In [8] it is described how it is possible to crawl for web sites instead of web pages. This leads to some different measures. Harvest rate is used in this crawler also.

- **Pages per relevant site rate**

This measure is only relevant in crawlers that aim to discover web sites. It is the number of downloaded pages divided by the number of relevant web sites found.

Harvest rate seems to be the most used measure when it comes to rating the performance of a focused crawler.

## 8.2 Test settings

### 8.2.1 Scope filter

In order to ensure that our crawler only downloads files with textual content, and not irrelevant files like images and video, we have added a filter when performing the tests. The filter is shown in Figure 8.1 and contains a list of file endings for all the file types we do not want the crawler to download. The filter uses regular expressions to decide which links the crawler should not follow and is based on a filter found at [41].

```

^.*(?!i)\.(a|ai|aif|aifc|aiff|asc|au|avi|bcpio|bin|bmp|bz2|c|cab|cdf|cgi|c
gm|class|cpio|cpp?|cpt|csh|css|cxx|dcr|dif|dir|djv|djvu|dll|dmg|dms|doc|d
td|dv|dvi|dxx|eps|etx|exe|ez|gif|gram|grxml|gtar|h|hdf|hqx|ice|ico|ics|ie
f|ifb|iges|igs|iso|jar|jnlp|jp2|jpe|jpeg|jpg|js|kar|latex|lha|lzh|m3u|mac
|man|mathml|me|mesh|mid|midi|mif|mov|movie|mp2|mp3|mp4|mpe|mpeg|mpg|mpga|
ms|msh|mxu|nc|o|oda|ogg|pbm|pct|pdb|pdf|pgm|pgn|pic|pict|pl|png|pnm|pnt|p
ntg|ppm|ppt|ps|py|qt|qti|qtif|ra|ram|ras|rdf|rgb|rm|roff|rpm|rtf|rtx|s|sg
m|sgml|sh|shar|silo|sit|skd|skm|skp|skt|smi|smil|snd|so|spl|src|srpm|sv4c
pio|sv4crc|svg|swf|t|tar|tcl|tex|texi|texinfo|tgz|tif|tiff|tr|tsv|ustar|v
cd|vrml|vxml|wav|wbmp|wbxml|wml|wmlc|wmls|wmlsc|wrl|xbm|xht|xhtml|xls|xml
|xpm|xsl|xslt|xwd|xyz|z|zip)$

```

**Figure 8.1** Scope filter which removes irrelevant file types

This URL filter is used to ignore irrelevant file types before they are actually downloaded. Files that are not removed by this filter are downloaded by the crawler. In order to ensure that the files that actually get downloaded only contain textual content, we have also added a filter that ensures that the content type is html or xhtml.

### 8.2.2 Seed URLs

In the tests we have focused our crawler to find pages within the field of “it-security”. The seed pages we have used to start the crawler with, is listed below. These URLs have been handpicked from the category Computers/Security [42] in the Open Directory Project.

<http://www.w3.org/Security/>  
<http://netsecurity.about.com/>  
<http://www.rsasecurity.com/>  
<http://www.unisys.com/>  
<http://www.us-cert.gov/>  
<http://www.sans.org/>  
<http://www.gocsi.com/>  
<http://www.cisecurity.org/>  
<http://www.icsa.net/>  
<http://www.securezone.com/>  
<http://www.securitysolutions.com/>  
<http://www2.csl.sri.com/jcs/>  
<http://www.securitymagazine.com/>  
<http://www.openssh.org/>

### 8.2.3 Input Ontology

The ontology that was used as input in the test crawls is based on a small selection of the security taxonomy found on [43]. This taxonomy consists of a collection of terms and associations from several web sites and RFCs. For a complete list of sources, see [43]. The input ontology was built up around the term Security. Figure 8.2 shows how the test ontology is presented in Omnigator. Omnigator was the only topic map viewer of the ones

we found, that could show the entire topic map at the same time. The pink connections represent Superclass-Subclass associations, and the purple ones represent Related associations. Unfortunately Omnigator only shows the roles of the associations when the mouse is moved over the association. Because of this the figure is less informative in this report than it is in Omnigator.

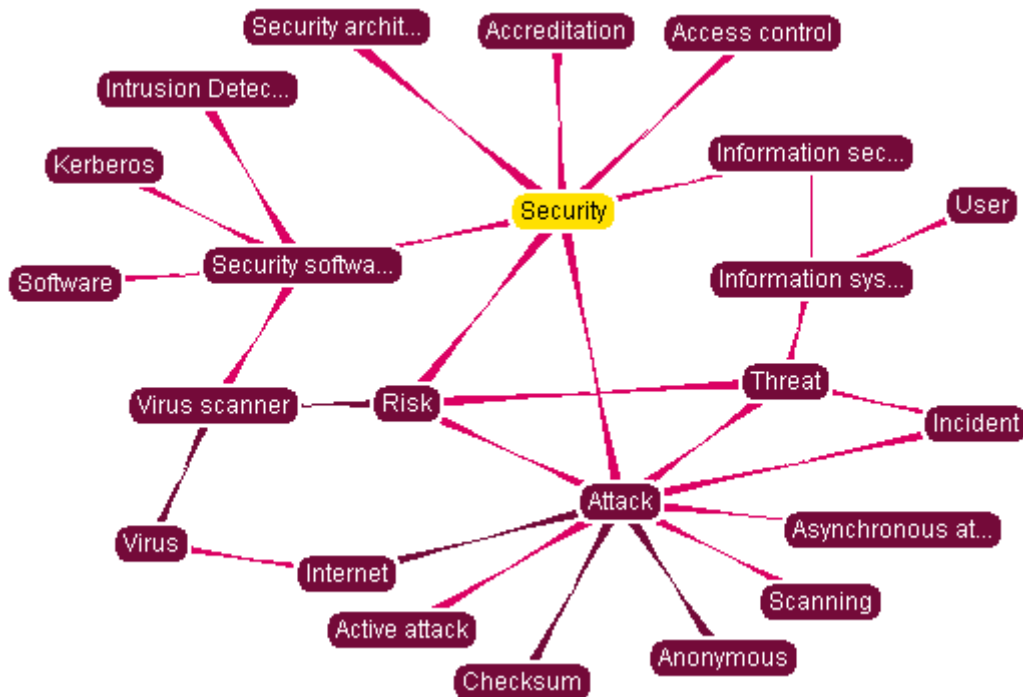


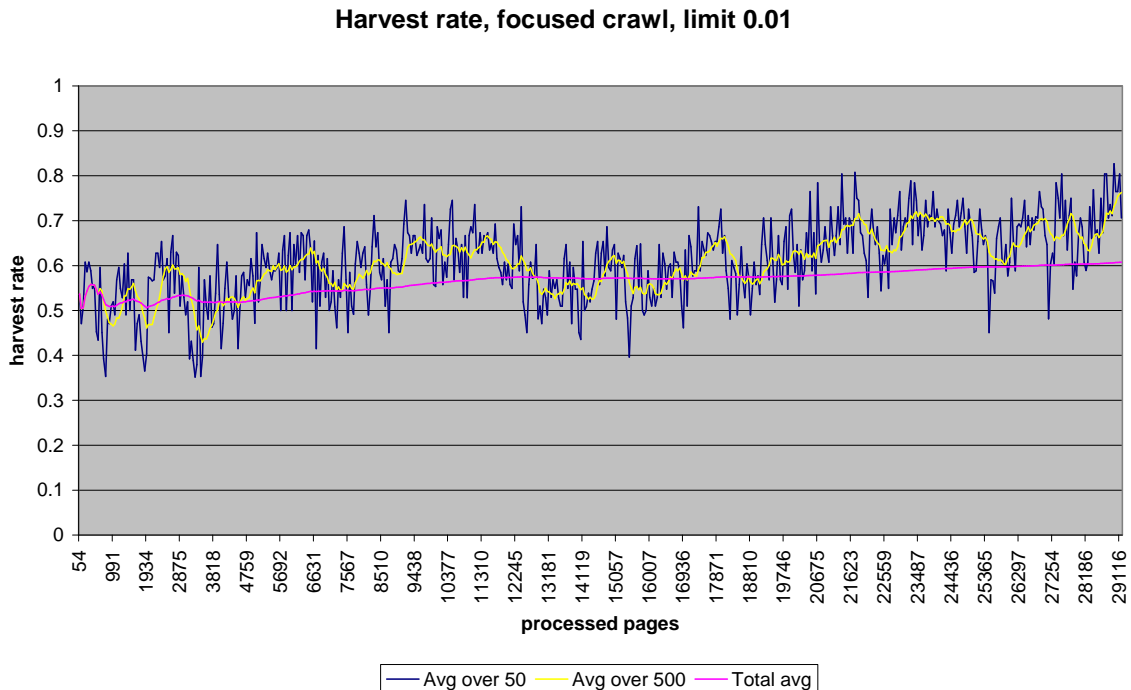
Figure 8.2 The input ontology used in the tests. Pink connections annotate Superclass-Subclass associations, while all other types of relations are purple.

### 8.3 Test results

All the diagrams in this chapter, except the last one, have three graphs. The ‘Avg over 50’-graph shows harvest rate for the last 50 downloaded pages, ‘Avg over 500’ shows harvest rate for the last 500 pages and ‘Total avg’ indicates the development of the harvest rate for all downloaded pages so far. The harvest rate sample rate can be configured in the web interface, and it was set to 50 in all the test crawls. This means that when the RelevancyCalculator has processed 50 web pages, the harvest rate of these 50 pages, and the harvest rate of all processed pages is calculated and added to a log file. A timestamp and the number of processed pages are also written to the file. The ‘Avg over 500’ values are calculated after the crawl is finished based on the ‘Avg over 50’ values. The name of the log file can be configured in the web interface.

Figure 8.3 shows the development of harvest rate on one of the focused test crawls. The relevancy limit of the crawl was set to 0.01. This means that web pages with a relevancy value less than 0.01 were considered irrelevant. Because it was a focused crawl, no links found on the irrelevant pages were followed. The graph shows a slight rise in the harvest rate, and the total harvest rate ends at about 0.6 after 29 000 processed pages. Notice that the x-axis shows the number of processed pages, and not downloaded pages. This is

because the downloaded pages number in Heritrix includes DNS lookups and robots.txt's. The processed pages only includes the URIs that are actually processed by the RelevanceCalculator, and therefore DNS lookups, robots.txt's, error pages (e.g. 404's) and pages that do not belong to the http or https scheme are not counted in this number. Pictures, text/plain and other non-HTML URIs that are not stopped by the scope filter described in chapter 8.2.1, will however be counted, even though none of these URIs ever can become relevant. These URIs constitute about 5 % of the URIs processed by the RelevanceCalculator.



**Figure 8.3 Harvest rate of focused crawl with relevancy limit 0.01**

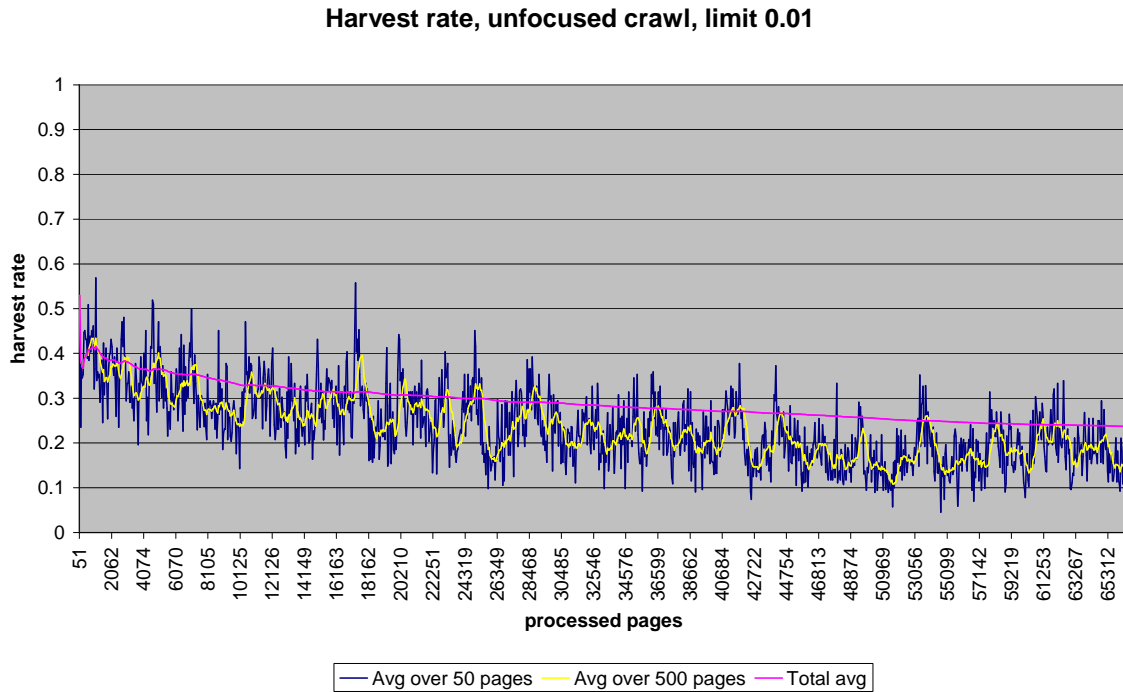
In Table 8.1 we present the most relevant web pages found by the focused crawl that generated the harvest rate shown in Figure 8.3.

Relevancy	URI
0.297761427	<a href="http://securitysoftwaresolutions.com/contact.htm">http://securitysoftwaresolutions.com/contact.htm</a>
0.286573388	<a href="http://securitysoftwaresolutions.com/_vti_bin/shtml.dll/contact.htm">http://securitysoftwaresolutions.com/_vti_bin/shtml.dll/contact.htm</a>
0.230789408	<a href="http://abanet.org/scitech/ec/isc/home.html">http://abanet.org/scitech/ec/isc/home.html</a>
0.219206044	<a href="http://visa-asia.com/secured/flash_index.html">http://visa-asia.com/secured/flash_index.html</a>
0.195066032	<a href="http://engarde.com/software/ipwatcher/risks/">http://engarde.com/software/ipwatcher/risks/</a>
0.193771758	<a href="http://safenet-inc.com/">http://safenet-inc.com/</a>
0.179166089	<a href="https://redsiren.com/infosec/samplecontent.html">https://redsiren.com/infosec/samplecontent.html</a>
0.177217099	<a href="http://anchortechnologies.com/prod_svcs.htm">http://anchortechnologies.com/prod_svcs.htm</a>
0.175182722	<a href="http://cisecurity.org/software_cert.html">http://cisecurity.org/software_cert.html</a>
0.172648746	<a href="http://networkassociates.com/us/audience/enterprise_home.asp">http://networkassociates.com/us/audience/enterprise_home.asp</a>

**Table 8.1 Top 10 web pages found by focused crawl with relevancy limit 0.01.**

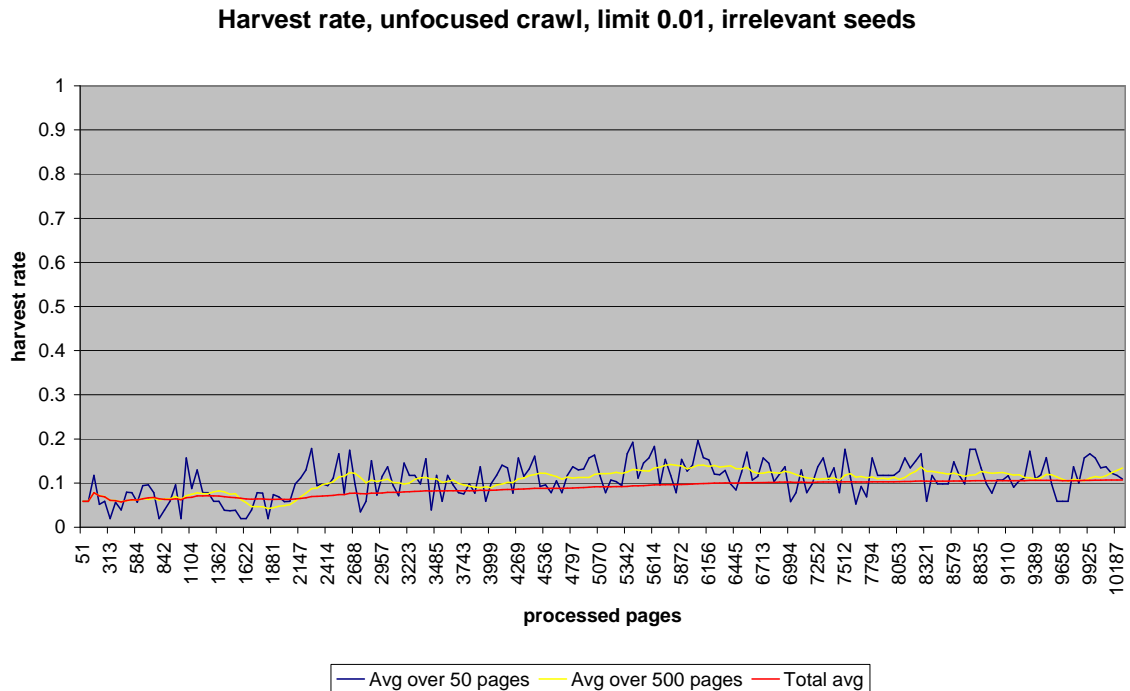


Figure 8.4 shows the harvest rate of an unfocused crawl with relevancy limit at 0.01. When the prototype is configured to crawl unfocused, all links are followed, but only the pages with relevance value above the limit will be considered relevant and added to the input topic map. The harvest rate starts off quite high, but then it keeps going lower as the crawl goes on.



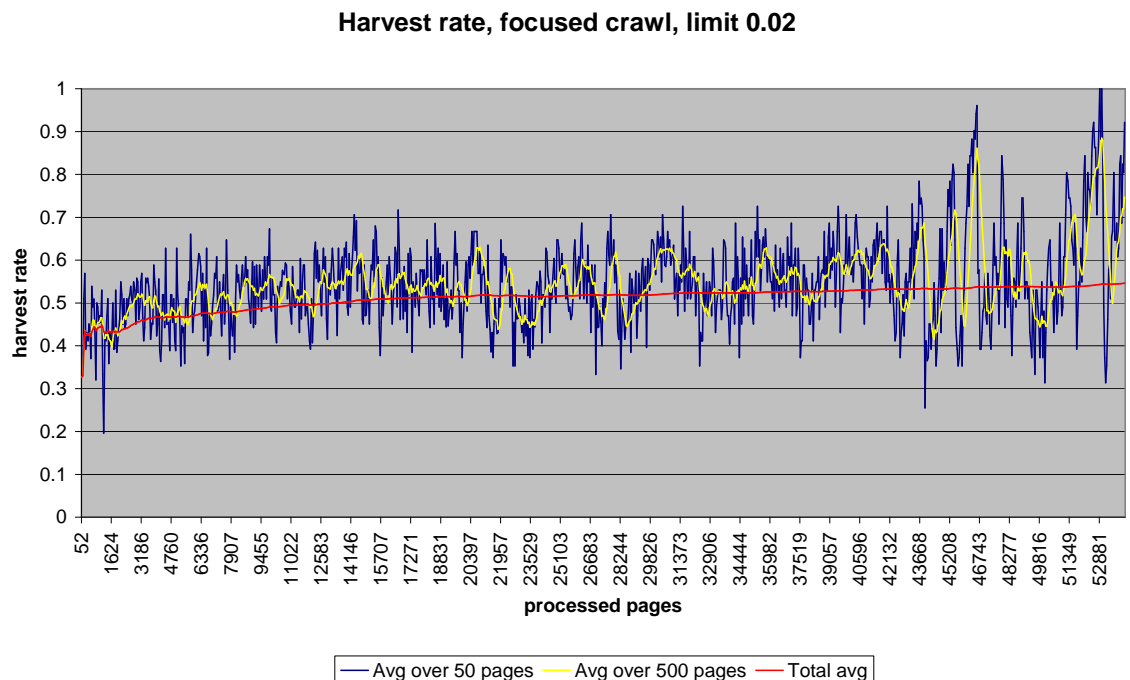
**Figure 8.4 Harvest rate of unfocused crawl with relevancy limit 0.01**

The crawl that generated the graph in Figure 8.5 was unfocused and it was started with irrelevant seeds. This test was run because we wanted to see how much effect the seeds had on the crawl and harvest rate. The harvest rate here actually increases slightly as more pages are downloaded.



**Figure 8.5** Harvest rate of unfocused crawl with relevancy limit 0.01 and irrelevant seeds

Figure 8.6 and Figure 8.7 show how the harvest rate turned out when the relevancy limit was raised to 0.02. The motivation for raising the limit was that we felt that the harvest rate in the unfocused crawls with limit 0.01 was suspiciously high.



**Figure 8.6** Harvest rate of focused crawl with relevancy limit 0.02

Harvest rate, unfocused crawl, limit 0.02

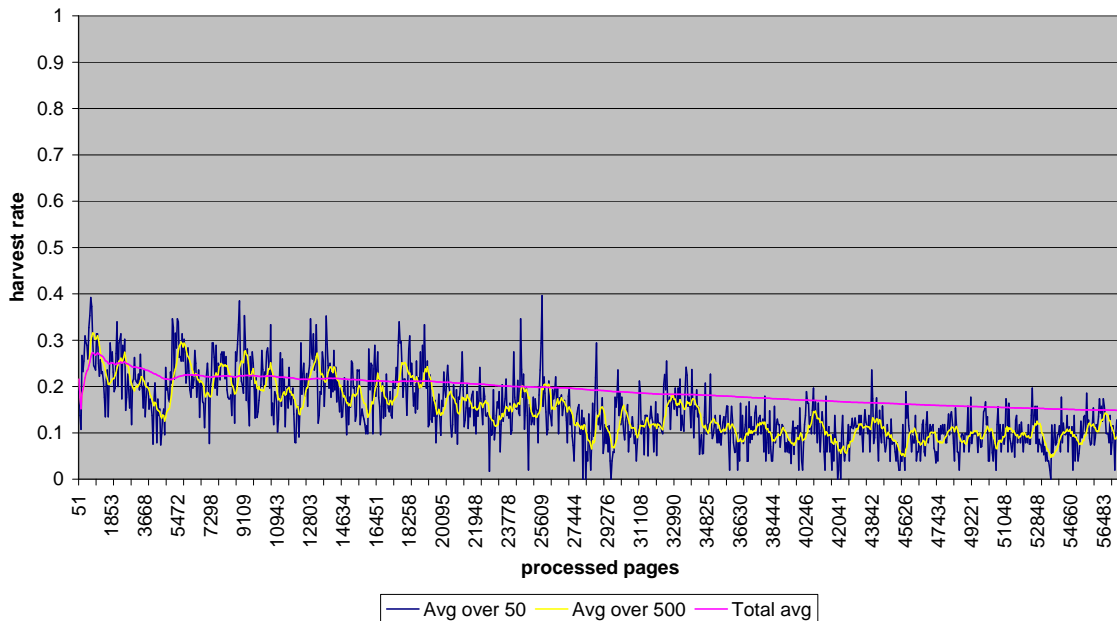


Figure 8.7 Harvest rate of unfocused crawl with relevancy limit 0.02

Harvest rate, unfocused crawl, limit 0.02, irrelevant seeds

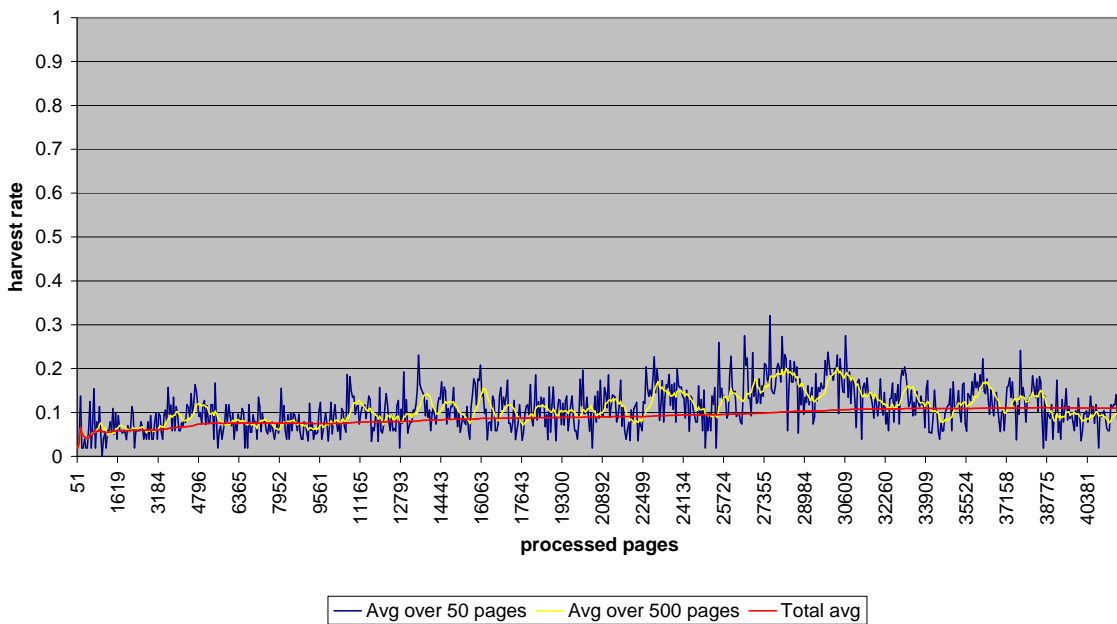
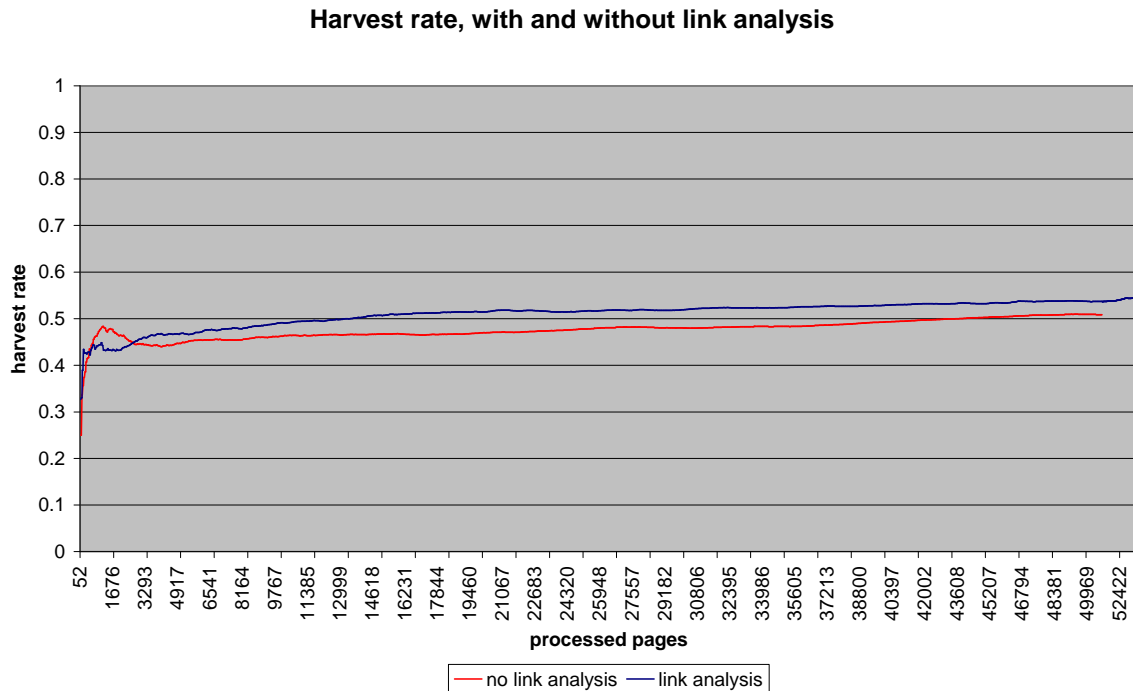


Figure 8.8 Harvest rate of unfocused crawl with relevancy limit 0.02 and irrelevant seeds

Figure 8.8 shows how the harvest rate develops in an unfocused crawl with irrelevant seed URLs and a relevance limit of 0.02. The graph starts off at about 5 % and rises slowly to around 10 % after 40 000 processed pages.



**Figure 8.9 Harvest rate of equal crawls with and without link analysis**

Figure 8.9 shows the harvest rates of one crawl with link analysis and one without link analysis. The crawl with link analysis achieves a harvest rate that is slightly higher than the harvest rate of the crawl without link analysis.

### **Robustness of acquisition**

The robustness of acquisition measure described in chapter 4.3 in [5] has also been used to describe our prototype. This was done by splitting the top 100 URIs returned when searching for the word *security* in the Computer/Security directory on dmoz.org, in half. These two subsets were then assigned as start URIs to two different test crawls with otherwise equal settings. When the crawls had been running for about 24 hours we compared the results. It turned out that of the 20 761 first relevant web pages found by the crawls, 10 322 were found by both. This gives a crawl robustness of 0.497. Because of limited time in the project, we were not able to investigate how the robustness of acquisition developed over time.

## 9 Discussion

By using ontologies to guide a focused crawl it is possible to provide the crawler with more knowledge about the target domain. Information like synonyms to the terms, and different types of relations between the terms can be used to make the crawler more capable of finding pages that are relevant.

Heritrix was easy to extend, and it has worked well as a basis for our prototype. The developers have succeeded in making an intuitive and modular crawler. By extending Heritrix we have reduced the time used on coding, but quite some time has been spent reading documentation and gaining understanding of the crawler. The solutions and strategies chosen in Heritrix have in some cases made it more complicated to implement solutions the way we wanted. Including Nutch' link analysis module in our Heritrix based prototype was complicated. Because Nutch and Heritrix have so different structures, we had to do several adjustments to both Heritrix and the link analysis module to make it work.

### 9.1 The prototype

The prototype reads the input topic map from a XTM-file during the initiation of each crawl. At first we tried to make the crawler add information about the relevant web pages it found to this topic map, and then write the topic map to a XTM-file again when the crawl was terminated or finished. This turned out to be difficult. The process of writing the topic map to a file was very demanding. Crawls that had downloaded around 50 000 URIs kept writing to the file for hours, and were still not done. It is obvious that this is not acceptable. Another problem was that at this point, the topic map was kept in the RAM. That means that the memory usage increased as more web pages were downloaded, and the `OutOfMemoryException` was unavoidable. TM4J supports different backends for containing the topic maps. In addition to the default `InMemory` backend, there is a `Hibernate-backend` that utilizes a traditional relational database, and an `Ozone-backend` that uses the Ozone object oriented database to persist the topic maps. We changed to the `Hibernate-backend` with a MySQL database, and this solved the memory problem. The `Hibernate-backend` also led to our solution of the problem of writing the topic map to file. The `TMNav` is a subproject of TM4J and therefore it can present topic maps from all backends supported by TM4J as well as topic maps from XTM-files. It turned out to take a lot less time to load the topic map directly from the `Hibernate` backend using `TMNav`, than to first write it to an XTM-file and then load the topic map from the file. The time consumption was reduced from several hours to a couple of minutes.

In addition we reduced the amount of information that was added to the topic map. In the earlier versions of our prototype, each of the topics that was found in a web page got an occurrence pointing to the URI, and all these occurrences were reified to be able to add TFIDF values for these occurrences. There was also added a topic representing the web page where the relevancy value was stored. This means that a web page mentioning 10 of the topics in the topic map will cause 10 occurrences, and 11 topics to be added to the topic map. The final version of the prototype only adds one topic and two occurrences for each

relevant web page that is found. This topic contains the relevance value of the web page, and the URI. We did this to keep the topic map as small as possible.

It was a bit difficult to find information on how to load topic maps from Hibernate backends in TMNav. In the comments in the configuration file of TMNav there was an example on how to add a Hibernate backend. By changing this example we managed to get things working, but there are some exceptions occurring when loading the topic map. We searched in search engines and relevant forums on the Internet, but we found very little information about using Hibernate backend with TMNav.

While testing our prototype we discovered that when the crawler has been running for some time, the WebDBPostselector uses quite a lot of time to do link analysis. In one of the tests we did, the crawler used 7 minutes to do link analysis when the crawler had been running for 20 hours. The link analysis started approximately every 20 minutes, so this part of the application used very much of the cpu-time. Calculating the score of all the links in the link database (WebDB) is a heavy task, and this task takes more time to complete when there are more links in the database. When the crawler has been running for some time the link database contains several thousand links, and because the computer we used when testing the prototype had a slow 800 MHz cpu, running the link analysis was also slow.

### **9.1.1 Problems running Nutch on Windows**

A lot of time was used in order to get the WebDB module from Nutch working in Heritrix. We discovered some problems while using Nutch on Windows XP. When running link analysis on the WebDB, Nutch returned error messages saying that the file "*db\webdb.new\pagesByURL*" already existed. This problem occurred because the file was not deleted the last time the link analysis was run. When running the `dbWriter.close()` method the "webdb.new" directory is copied to "webdb" and the "webdb.new" should be deleted. When running Nutch on Windows XP the "webdb.new" directory was not deleted and the next time link analysis was started, Nutch stopped because the "webdb.new" directory already existed.

To be able to delete files and directories, Nutch uses the class `LocalFileSystem` from the package `net.nutch.fs`. The method used to delete files or directories is listed below in Figure 9.1.

```
private boolean fullyDelete(File dir) throws IOException {
    File contents[] = dir.listFiles();
    if (contents != null) {
        for (int i = 0; i < contents.length; i++) {
            if (contents[i].isFile()) {
                if (! contents[i].delete()) {
                    return false;
                }
            } else {
                if (! fullyDelete(contents[i])) {
                    return false;
                }
            }
        }
    }
    return dir.delete();
}
```

**Figure 9.1** Code from `net.nutch.LocalFileSystem.java`

This is a recursive method that deletes a directory and all its subdirectories and files. For some reason this method was not able to delete the file `"db\webdb.new\pagesByUrl"` when running Nutch on Windows XP. We were not even able to delete this file manually using Windows Explorer. It seemed like the file was still in use by Nutch and therefore could not be deleted.

After a lot of testing and debugging of Nutch running on Windows XP, we tried to run Nutch on Linux. We discovered that the problem with the `"webdb.new"` directory did not occur on Linux. Because of this we switched and used Linux when running tests with our focused crawler.

As stated on the homepage of Nutch [24] Linux is preferred, but Nutch is developed using Java and should therefore have been platform independent.

### 9.1.2 Problems with DNS lookup in Heritrix

The earliest tests of Heritrix were run on a computer running Norwegian Microsoft Windows XP Professional. In these tests the crawler only worked when we disabled the PreconditionEnforcer processor, and it was very slow, only two pages downloaded per second. After some debugging we found out that the library Heritrix used to do DNS lookup (dnsjava-1.6.2) only worked on English Windows XP. The reason for this was simply that dnsjava used the `ipconfig` command to get the IP addresses of the DNS servers. The output from the `ipconfig` command was then parsed to find the correct line. The program searched for the line starting with `"DNS-servers:"`, but could not find it because the Norwegian version of `ipconfig` prints out `"DNS-servere:"`. We changed the source code to look for the Norwegian text, compiled the Java file and replaced the class file in the

dnsjava jar file. After having done this, the DNSLookup in Heritrix worked again, and the speed increased to between 20 and 30 pages per second. The crawler finds DNS servers differently on Linux, so when it was moved to the Linux server the fix in the dnsjava library was not needed anymore.

## 9.2 Test results

We believe that the harvest rate values from the unfocused test crawls are too high compared to the focused crawls with the same relevancy limit. The implementation of the IDF calculation makes it difficult to compare the TFIDF values from a focused and an unfocused crawl. IDF is calculated like this:

$$\text{idf}_t = \log\left(\frac{D}{df_t}\right)$$

where  $df_t$  is number of documents containing token  $t$ , and  $D$  is total number of documents.

	Unfocused crawl	Focused crawl
Processed documents ( $D$ )	50 000	50 000
Documents containing a relevant term $t$ ( $df_t$ )	6 000	25 000
IDF value for term $t$	0.921	0.301

**Table 9.1** IDF values for the same term in a typical focused and unfocused crawl

The problem is that in our implementation the  $D$  includes all processed web pages, including the irrelevant documents. An unfocused crawl will have a higher percentage of irrelevant pages, and therefore also a lower share of documents containing relevant terms. This could be demonstrated with an example. Let us say we have a focused and an unfocused crawl with otherwise equal settings that each has processed 50 000 web pages. The relevant term  $t$  could typically occur in 6 000 of the documents found in the unfocused crawl, and in 25 000 documents in the focused crawl. These numbers would result in IDF values for the unfocused and focused crawls at respectively 0.921 and 0.301. This is in a way correct, because the relevant term is rarer in the unfocused crawl, and hence should get a higher IDF. But it could result in the same web page getting a higher relevancy in an unfocused crawl than in a focused crawl, even though the content of the page has not changed. Table 9.2 shows a list of URIs with relevancy value from a focused and an unfocused crawl with otherwise equal settings.



URI	Focused	Unfocused
<a href="http://symantec.com/techsupp/enterprise/products/sesa/files.html">http://symantec.com/techsupp/enterprise/products/sesa/files.html</a>	0.19329686	0.35987031
<a href="http://engarde.com/software/ipwatcher/risks/">http://engarde.com/software/ipwatcher/risks/</a>	0.15190832	0.17930799
<a href="http://cve.mitre.org/press/releases.html">http://cve.mitre.org/press/releases.html</a>	0.12576312	0.24291519
<a href="http://gocsi.com/netsec/pre_post.jhtml">http://gocsi.com/netsec/pre_post.jhtml</a>	0.12397279	0.18799698
<a href="http://forum.sans.org/discus/messages/729/729.html?1091641275">http://forum.sans.org/discus/messages/729/729.html?1091641275</a>	0.11474685	0.27941003
<a href="http://engarde.com/software/manualintrusion.php">http://engarde.com/software/manualintrusion.php</a>	0.10932691	0.12981179
<a href="http://altaassociates.com/announcements/index.asp">http://altaassociates.com/announcements/index.asp</a>	0.10772106	0.22833505
<a href="http://engarde.com/aboutus/contact.php">http://engarde.com/aboutus/contact.php</a>	0.107067	0.10144658
<a href="http://redsiren.com/abouti4.htm">http://redsiren.com/abouti4.htm</a>	0.10223937	0.26713874
<a href="http://engarde.com/software/ipwatcher/paper.php">http://engarde.com/software/ipwatcher/paper.php</a>	0.09855605	0.12443033
<a href="http://engarde.com/software/ipwatcher/info.php">http://engarde.com/software/ipwatcher/info.php</a>	0.09566358	0.12175935
<a href="http://redsiren.com/">http://redsiren.com/</a>	0.09517584	0.19776214
<a href="http://engarde.com/software/ipwatcher/thanks.php">http://engarde.com/software/ipwatcher/thanks.php</a>	0.09467553	0.14540123
<a href="http://redsiren.com/soc.htm">http://redsiren.com/soc.htm</a>	0.09459465	0.25206705
<a href="http://gocsi.com/training/">http://gocsi.com/training/</a>	0.09411774	0.12418727

**Table 9.2 Relevancy of some web pages in a focused and an unfocused crawl with equal settings**

This means that if the relevancy limit of a focused and an unfocused crawl is set to the same value, for example 0.02, and all other settings are the same, many pages will be considered relevant by the unfocused crawl that are considered irrelevant in the focused crawl. When we compared the relevancy values of some web pages processed in a focused and an unfocused crawl, we found that these web pages on average got 125 % higher relevance values in the unfocused crawl. This means that the relevance limit of the unfocused crawl should be set 125 % higher than the limit of the focused crawl. The unfocused crawl in Figure 8.7 should therefore have a limit of 0.045 to be comparable to the focused crawl with relevance limit at 0.02 (Figure 8.6). Only half of the pages found to be relevant by the unfocused crawl with relevancy limit 0.02 have a relevancy above 0.045. This means that the harvest rate shown in Figure 8.7 and Figure 8.4 should have been about 50 % lower.

The harvest rate of the focused crawl with relevancy limit at 0.01 (Figure 8.3) was promising, but the fact that the harvest rate is quite high for the unfocused crawl (Figure 8.4) also, indicates that a relevancy limit of 0.01 might be too low. This is why we increased it to 0.02 in the next tests. By raising the limit, fewer web pages are considered relevant, and the harvest rate in the unfocused crawls will be lowered.

We also did a few unfocused test crawls with irrelevant seed URLs, to see how much the seeds affected the focusing and the harvest rate. The harvest rate of the unfocused crawls with relevant seeds started quite high, but got lower and lower as the crawl went on (see Figure 8.4 and Figure 8.7). In the unfocused crawls with irrelevant seeds, the harvest rate was more constant (see Figure 8.5), and it was about 10 % in the crawl with relevancy limit 0.01. The harvest rate of the test with relevant seeds is above 20 % even after more than 65 000 web pages has been processed. This means that the seeds also play an important part in the focusing.

In one of the tests we tried to measure the effect of the link analysis module. This was done by running two test crawls, one with link analysis and one without link analysis. All of the other settings were equal. The harvest rate of these test crawls are shown in Figure 8.9. The result shows that using link analysis only gives a slight increase in the harvest rate. This indicates that the prototype does not find many more relevant pages when using link analysis, but it may be that the content of the pages are better, because the link analysis can find more authoritative pages. Harvest rate is not the best way to measure the effect of link analysis, so more testing with other types of measures could therefore be done to further evaluate the link analysis module. Link analysis might not be as useful as we thought. In an interview with Doug Cutting [44], the primary developer of Nutch, Cutting says that the value of link analysis is somewhat overrated.

In all of our tests we have used our prototype to crawl pages directly on the Web. Since the tests have not been performed at the exact same time, the contents of some pages might have changed between each test. In our project we have not considered this to be a significant problem, but it is obvious that the modified pages might have participated in causing the differences in the test results. To prevent changing pages from affecting the test results, it could have been possible to test the prototype on a fixed set of pages. One way of doing this is to use a web proxy server and configure it to not download pages that have already been downloaded. In this way our prototype would only have crawled the pages that already existed in the proxy server's cache. Because of limited time in our project we did not pursue this possibility.

### 9.3 Further work

There are several ideas and methods that could be further explored in our prototype. Our prototype and algorithm contains several attributes which can be adjusted, and it is fairly easy to extend with more sophisticated algorithms. More testing of the prototype can therefore easily be done.

One thing that should be tested more thoroughly is how the weight class values affect the results. Will a higher value on the more peripheral topics (in relation to the focal topics), lead to fewer web pages about security in general (not computer security) being considered relevant?

Implementing a best-first frontier could have been an interesting approach. A best-first frontier could make sure that the links found on the most relevant pages are downloaded first. As of now, our prototype uses a kind of best-first strategy based on the link scores, but it would have been interesting to also take into account the text relevancy of the pages where the links were found. The frontier queue could in this way be sorted to download the most prominent pages first.

Enabling the crawler to find web sites instead of web pages is another interesting approach that could be examined. Ester et al. [8] proposes a method which uses an internal crawler

to view the web pages of a single web site, and an external crawler that has a more abstract view of the web as a graph of linked web sites.

One possible area of application for our prototype is to use it to create a portal focused on a specific domain or subject. In the current state, our prototype does not save the content of the crawled pages. If we add an indexing module and a search interface, it would be possible to create a search portal or search engine where users can search in a database that only contains pages relevant to a certain subject. For example researchers and others interested in it-security could then create their own web portal which crawls the Web for pages about it-security. Users in that community could then read the latest pages about it-security or search in a database that only contains pages within the domain it-security. One advantage of this is that the database is smaller and therefore easier to maintain and keep up-to-date, compared to a standard search engine which tries to crawl the entire Web. Each community could then have their own focused search engine. The task of finding relevant pages on the Web could thereby be distributed to several specialized search engines in different communities. If each of these communities had their own focused search engines, based on the same system, it might also be possible to create a common front-end for all the search engines. Users could then use this common front-end to search in all of the specialized distributed search engines at the same time.

## 10 Conclusion

In our project we have developed a prototype that uses an ontology to perform focused crawling. The prototype uses the structured information in the ontology to guide the crawler in its search for web pages that are relevant to the topic specified in the ontology. Our relevance algorithm is inspired by the relevance computation strategy proposed by Ehrig et al. [3]. Their algorithm tries to map the content of a web page against an ontology to gain an overall relevance-score.

In this thesis we have evaluated several Java-based open source crawlers, but Heritrix stands out as the most extensible and best suited crawler for our purpose. Our prototype is therefore built upon the Heritrix open source crawler. Even though we experienced some minor problems with Heritrix, we feel that Heritrix is a stable and powerful crawler, and it is well suited for developing extensions.

In addition to using an ontology to measure the relevancy of web pages, we have also used link analysis to determine the importance of a link before it is downloaded. In our prototype, link analysis is achieved by using the WebDB module from the open source search engine Nutch. The test results show that the use of link analysis in our prototype gives a slight increase in the harvest rate, but it did not give as much improvement as we had hoped.

When we started working on this project, we did not have much experience with ontologies, but throughout this thesis we have shown that ontologies are a suitable technology for creating focused crawlers. In the tests we have performed our prototype shows good results, but there are still several adjustments that could be done on the settings of the prototype and on the relevance algorithm itself. More testing could also be done in order to evaluate how well the prototype works when it is set to focus on other topics.

## Bibliography

- [1] ISC Internet Domain Survey, Available from:  
<http://www.isc.org/index.pl?/ops/ds/reports/2005-01/> [Accessed on May 2005]
- [2] G. Salton, C. Buckley, "Term weighting approaches in automatic text retrieval," *Information Processing and Management*, 24(5):513-523, 1988
- [3] M. Ehrig, A. Maedche, "Ontology-Focused Crawling of Web Documents," in Proc. of the 2003 ACM symposium on Applied computing, Melbourne, Florida, 2003
- [4] M. Diligenti, F.M. Coetzee, S. Lawrence, C.L. Giles, M. Gori, "Focused Crawling Using Context Graphs," in 26th International Conference on Very Large Databases, VLDB 2000
- [5] S. Chakrabarti, M. van den Berg, B. Dom, "Focused crawling: a new approach to topic-specific Web resource discovery," in 8th International World Wide Web Conference, May 1999
- [6] J. M. Kleinberg, "Authoritative Sources in a Hyperlinked Environment," in Proceedings of ACM-SIAM Symposium on Discrete Algorithms, 668-677, January 1998
- [7] M. Diligenti, F. M. Coetzee, S. Lawrence, C. L. Giles, M. Gori, "Focused Crawling Using Context Graphs," in Proc. of the 26<sup>th</sup> VLDB Conference, Cairo, Egypt, 527-534, 2000
- [8] M. Ester, H. Kriegel, M. Schubert, "Accurate and Efficient Crawling for Relevant Websites," in Proc. of the 30<sup>th</sup> VLDB Conference, Toronto, Canada, 2004
- [9] S. Sizov, S. Siersdorfer, M. Theobald, G. Weikum, "The BINGO! Focused Crawler: From Bookmarks to Archetypes," in Proc. of the 18th International Conference on Data Engineering (ICDE 02), 2002
- [10] S. Chakrabarti, B. Dom, P. Raghavan, S. Rajagopalan, D. Gibson, J. Kleinberg, "Automatic resource compilation by analyzing hyperlink structure and associated text," in Proc. of the 7th World Wide Web Conference, pages 65-74, 1997
- [11] Wikipedia article on Ontology, Available from:  
[http://en.wikipedia.org/wiki/Ontology\\_\(computer\\_science\)](http://en.wikipedia.org/wiki/Ontology_(computer_science)) [Accessed May 2005]
- [12] T. R. Gruber, "What is an Ontology?," Available on: <http://www-ksl.stanford.edu/kst/what-is-an-ontology.html> [Accessed May 2005]
- [13] N. F. Noy, D. L. McGuinness, "Ontology Development 101: A Guide to Creating Your First Ontology" Available on:  
<http://www.ksl.stanford.edu/people/dlm/papers/ontology101/ontology101-noy-mcguinness.html> [Accessed May 2005]
- [14] W3C. Resource Description Framework (RDF), Available on:  
<http://www.w3.org/RDF/> [Accessed May 2005]
- [15] W3C. Web Ontology Language (OWL), Available on: <http://www.w3.org/TR/owl-features/> [Accessed May 2005]
- [16] XML Topic Maps (XTM) 1.0, Available on: <http://www.topicmaps.org/xtm/1.0/> [Accessed May 2005]
- [17] S. Pepper, "The TAO of Topic Maps," Available from:  
<http://www.ontopia.net/topicmaps/materials/tao.html> [Accessed May 2005]

- [18] L. M. Garshol, "Topic Maps, RDF, DAML, OIL," Available from: <http://www.ontopia.net/topicmaps/materials/tmrdfoildaml.html> [Accessed May 2005]
- [19] TopQuadrant Inc., Ontology Languages figure, Available on: <http://www.coolheads.com/egov/opensource/topicmap/s167/img21.html> [Accessed May 2005]
- [20] Topic Maps For Java web page, Available on: <http://tm4j.org/> [Accessed May 2005]
- [21] Ozone Database Project web page, Available on: <http://ozone-db.org/> [Accessed May 2005]
- [22] Hibernate web page, Available on: <http://www.hibernate.org/> [Accessed May 2005]
- [23] Ontopia web page, Available on: <http://www.ontopia.net/> [Accessed May 2005]
- [24] Nutch web page, Available on: <http://incubator.apache.org/nutch/> [Accessed May 2005]
- [25] Heritrix web page, Available on: <http://crawler.archive.org/> [Accessed May 2005]
- [26] dnsjava web page, Available on: <http://www.xbill.org/dnsjava/> [Accessed May 2005]
- [27] <http://www.archive.org>
- [28] WebLech URL Spider web page, Available on: <http://weblech.sourceforge.net/> [Accessed May 2005]
- [29] WebSPHINX web page, Available on: <http://www-2.cs.cmu.edu/~rcm/websphinx/> [Accessed May 2005]
- [30] JSpider web page, Available on: <http://j-spider.sourceforge.net/> [Accessed May 2005]
- [31] HyperSpider web page, Available on: <http://hyperspider.sourceforge.net/> [Accessed May 2005]
- [32] Arale web page, Available on: <http://gomba.sourceforge.net/flavio/arale.html> [Accessed May 2005]
- [33] J. E. Hasle, G. Mohr, K. Sigurdsson, M. Stack, "Heritrix developer documentation," Available on: [http://crawler.archive.org/articles/developer\\_manual.html](http://crawler.archive.org/articles/developer_manual.html) [Accessed May 2005]
- [34] G. Mohr, M. Stack, I. Ranitovic, D. Avery, M. Kimpton, "An Introduction to Heritrix," 2004, Available on: <http://crawler.archive.org/An%20Introduction%20to%20Heritrix.pdf> [Accessed May 2005]
- [35] V. A. Oleshchuk, A. Pedersen, "Ontology Based Semantic Similarity Comparison of Documents"
- [36] C. Liao, S. Alpha, P. Dixon, "Feature Preparation in Text Categorization"
- [37] S. Brin, L. Page. The anatomy of a large-scale hypertextual (Web) search engine. In The Seventh International World Wide Web Conference, 1998.
- [38] R. Khare, D. Cutting, K. Sitaker, A. Rifkin, "Nutch: A Flexible and Scalable Open-Source Web Search Engine"
- [39] CyberNeko HTML Parser home page, Available on: <http://people.apache.org/~andyc/neko/doc/html/> [Accessed May 2005]
- [40] Lucene web page, Available on: <http://lucene.apache.org/java/docs/> [Accessed May 2005]
- [41] Tom Emerson's Heritrix Blog, Available on: <http://www.dreamersrealm.net/tree/blog/heritrix/> [Accessed January 2005]

- [42] dmoz - open directory project web page, Available on: <http://www.dmoz.org/Computers/Security/> [Accessed May 2005]
- [43] Security Taxonomy, Available on: <http://www.garlic.com/~lynn/> [Accessed April 2005]
- [44] Doug Cutting Interview, Available on: [http://blog.outercourt.com/archive/2004\\_05\\_28\\_index.html](http://blog.outercourt.com/archive/2004_05_28_index.html) [Accessed May 2005]