

AUTOMATING THE EXTRACTION OF DOMAIN-SPECIFIC
INFORMATION FROM THE WEB—A CASE STUDY
FOR THE GENEALOGICAL DOMAIN

by

Troy Walker

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Brigham Young University

August 2004

Copyright © 2004 Troy Walker

All Rights Reserved

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Troy Walker

This thesis has been read by each member of the following graduate committee and by a majority vote has been found to be satisfactory.

Date

David W. Embley, Chair

Date

Dan R. Olsen

Date

Robert P. Burton

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Troy Walker in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscripts is satisfactory to the graduate committee and is ready for submission to the university library.

Date

David W. Embley
Chair, Graduate Committee

Accepted for the Department

David W. Embley
Graduate Coordinator

Accepted for the College

G. Rex Bryce
Associate Dean, College of Physical and
Mathematical Sciences

ABSTRACT

AUTOMATING THE EXTRACTION OF DOMAIN SPECIFIC INFORMATION FROM THE WEB—A CASE STUDY FOR THE GENEALOGICAL DOMAIN

Troy Walker

Department of Computer Science

Master of Science

Current ways of finding genealogical information within the millions of pages on the Web are inadequate. In an effort to help genealogical researchers find desired information more quickly, we have developed GeneTIQS, a Genealogy Target-based Information Query System. GeneTIQS builds on ontology-based methods of data extraction to allow database-style queries on the Web. This thesis makes two main contributions to GeneTIQS. (1) It builds a framework to do generic ontology-based data extraction. (2) It develops a hybrid record separator based on Vector Space Modeling that uses both formatting clues and data clues to split pages into component records. The record separator allows GeneTIQS to extract data from the complex documents common in genealogy. Experiments show that this approach yields 92% recall and 93% precision on documents from the Web.

ACKNOWLEDGMENTS

I would like to thank all those who supported me in this pursuit. I am grateful to Dr. Embley for his mentoring and advice over the years and for his careful editing of this thesis. I thank all of the professors at Brigham Young University who have helped expand my vision of the beauty of computer science.

I thank my parents, family, and wife for the stimulating environment I enjoy, for their efforts on my behalf, for their emotional support, and for making me want to be better.

This research has been supported in part by the National Science Foundation under grant #IIS-0083127.

Table of Contents

Table of Contents	vii
List of Tables.....	ix
List of Figures.....	xi
Chapter 1 Introduction.....	1
Chapter 2 Related Work	5
Chapter 3 The Extraction Ontology	9
Chapter 4 The Extraction System.....	15
4.1 User Query.....	16
4.2 URL List.....	18
4.3 URL Selector.....	19
4.4 Document Retriever and Structure Recognizer	19
4.5 Extraction Modules	20
4.5.1 Single-Record or Multiple-Record Engine.....	21
4.5.2 Specialized Extraction Modules	21
4.5.3 Other Extraction Modules	22
4.6 Data Constrainer.....	23
4.7 Result Filter	24
4.8 Result Presenter.....	25
Chapter 5 VSM-Based Record Separation	27
5.1 The Problem of Record Separation	27
5.2 Previous Work.....	30

5.3	Vector Space Modeling	32
5.4	Improved Method of Record Separation	35
5.5	Problems and Refinements	36
5.5.1	Schema Differences	37
5.5.2	Over-richness of Data	38
5.5.3	Missed Simple Patterns	38
Chapter 6 Experimental Results		41
6.1	Test Documents	41
6.2	Results	42
6.2.1	Single-record Documents	43
6.2.2	Simple Multiple-record Documents	44
6.2.3	Complex Multiple-record Documents	45
6.3	Summary of Results	47
Chapter 7 Conclusion and Future Work		49
7.1	Future Work	49
Bibliography		51
Appendix A Test Documents		55

List of Tables

Table 1: Ontology Vector.....	33
Table 2: DOM Tree Selections and Vector Scores	36
Table 3: Single-record Document Results.....	43
Table 4: Simple Multiple-record Document Results.....	44
Table 5: Results for Old Record Separator.....	44
Table 6: Complex Multiple-record Document Results	46
Table 7: Sources of Single-record Test Documents	55
Table 8: Sources of Simple Multiple-record Test Documents	56
Table 9: Sources of Complex Multiple-record Test Document	56

List of Figures

Figure 1: Extraction Ontology for Genealogy Domain.....	10
Figure 2: Regular Expressions for Matching Locations.....	12
Figure 3: Exception and Context Expressions.....	13
Figure 4: System Overview.....	15
Figure 5: Generated Query Form	17
Figure 6: Expanded Query Form.....	17
Figure 7: Extraction Results Presented to the User	24
Figure 8: Extraction Results with Relationship Expanded.....	25
Figure 9: Single-Record Document.....	28
Figure 10: Simple Multiple-Record Document.....	29
Figure 11: Complex Multiple-Record Document.....	30
Figure 12: Illustration of VSM Measures.....	34
Figure 13: Records That Should Be Recombined	47

Chapter 1

Introduction

The genealogy domain is a clear example of the growth of data on the internet and the challenge of locating data of interest. Over the years, various organizations including governments, companies, and churches have made genealogical information available online. In addition, thousands of individuals have posted the results of their personal genealogical research. This collection of data is constantly growing. In March 2003, a search for ‘Walker Genealogy’ on Google [Goo04] returned 199,000 documents. Just one year later, the same search yielded 338,000 documents. It is clear that users are no longer limited by the availability of information; instead, they are limited by their ability to find the information they want.

Currently, users have two means of finding the information they want: directories and search engines. A directory contains links to web resources for a specific topic along with a summary of the information contained in each resource. The genealogy categories in Yahoo [Yah04] and dmoz [dmoz04] list only 2,673 and 6,399 sites respectively. Cyndi’s List [Cyn04], a directory tailored specifically to genealogy, offers more information—217,950 sites. Directories are laboriously compiled by hand—content creators or moderators submit additions. Since directories require human effort, they are limited in number and tend to be general in scope. Since Cyndi’s list contains only 150 categories, researchers will have an average of 1,400 sites to look at once they have narrowed their search to one category. Users of directories must find the topic that best matches their needs and are unlikely to find a

perfect match. For this reason, directories are ideal for a new user becoming acquainted with an area of interest, but offer little help for specific information needs.

Search engines make more information available than directories at the possible cost of relevancy. We have already seen how many documents are available through Google for 'Walker Genealogy.' Some of these pages are advertisements while others are gateway pages taking advantage of Google's algorithms to appear on as many searches as possible. These pages contain no genealogical information. The majority of these results, however, represent a wealth of information. Fortunately, search engines give researchers more power in refining their search. When looking for a certain person with a given birth date, they can enter a name and birth date as keywords for a search. However, there is no guarantee that the matching date found in the result is a birth date or even that it corresponds with the matched name.

There are limitations to both of these approaches. Both offer users links to documents that may contain the information they need; however, it is up to the users to read the documents to find it. Both options are limited in their query capability. Search engine users benefit from answers made specifically for their needs, but are limited to simple keyword searches.

Various commercial genealogy sites [Anc04, Gen04, Roo04] exist to help users find the information they need. They offer directories and search engines better tuned for finding genealogical information as well as genealogical records such as census and death records to help users find information. Typically, they allow users to submit their family trees to a communal database. Although these sites contain a wealth of

information, they do not cover all of the data available on the Web. If the information needed cannot be found on these sites, users are back to browsing pages one by one.

To aid users in managing this deluge of information, we have built a system to extract genealogical information from relevant Web pages. This allows more structured, database-style queries on the information. We built our system on the foundation of previous work done in the Data Extraction Group of Brigham Young University and made improvements necessary for extracting genealogical data.

We designed a general framework for a system able to extract genealogy data from the Web. We included in this framework the components required to extract data from web pages. As time goes on, more methods of retrieving data will undoubtedly appear. We designed our framework to easily accept modules to include these sources in extraction. Although we designed it with genealogy in mind, nothing in this framework is genealogy-specific. Since Genealogy is a particularly complex domain, we argue that success in this domain will likely carry over to most domains.

Existing methods of record separation, where a document is divided into separate records where each record contains information about one object (e.g. a person), were ineffective on pages with complex formatting, like genealogy pages. We realized that improvement in this area would be essential to any extraction effort involving genealogy. We created a method to find records in a document using Vector Space Modeling scores, which finds and separates applicable data in even complex documents.

Our system for extracting genealogical information from the web is called GeneTIQS—Genealogical Target-based Information Query System. We present the

details of GeneTIQS as follows. In Chapter 2, we evaluate existing methods of data extraction and show that we have likely chosen the only extraction method that promises to work well in the genealogy domain. Chapter 3 explains the conceptual model used to specify the domain of interest for extraction. Chapter 4 describes our system and its components. Chapter 5 describes our improved method of record separation that is able to handle complex pages such as those commonly found in genealogy. Chapter 6 shows the effectiveness of our system by applying our work to experimental data. Finally, Chapter 7 concludes the thesis, discussing the limitations we found and possible future work.

Chapter 2

Related Work

To extract wanted data from a Web site, a computer agent must be able to locate the data of interest. Initial efforts at automatic information extraction from the Web involved manually writing page-specific wrappers. These wrappers were tailored to certain Web pages, were tedious to write and debug, and often broke when the format of the target Web page changed.

Because of this, researchers have worked on the semiautomatic generation of wrappers. Currently, there are at least 39 semi-automatic wrapper generators [KT02]. [LRS+02] divides these into five categories:

- Languages for wrapper development were developed because of the difficulty of writing and debugging Web extraction programs in existing programming languages. TSIMMIS [HGN+97] and Garlic [RS97] are examples of this approach.
- HTML-aware tools create wrappers that locate data based on location within the tree created by parsing the documents' HTML. They do so either semi-automatically with one example, or automatically with many example pages from one source. W4F [SA99] is a popular tool in this category
- Wrapper induction tools attempt to recognize patterns in a labeled set of training pages. Ariadne [KMA+98] automatically creates a grammar to recognize patterns from labeled examples.

- Modeling-based tools use methods similar to wrapper induction tools to match a data structure provided by the user. DEByE [LRS02] is one example.
- NLP-based tools use Natural Language Processing techniques to extract information from free text. Some are able to extract directly from HTML documents while others need plain text input.
- Ontology-based tools rely on a conceptual model of the data to be extracted. BYU's Ontos [ECJ+99] pioneered this approach. According to [LRS+02], this approach has the advantage of being adaptive and resilient.

The wrappers generated by the first three approaches will extract data from pages similar to the pages on which they were trained. They will work on pages from the same site with the same format. That means they must train on each data source. Since we are interested in extracting data from pages created by thousands of hobbyists, this would be a time consuming task. Even if wrappers were generated for all of these sites, maintaining these wrappers would be challenging. Since these approaches rely on the formatting of pages to guide extraction, they would stop working if the pages changed significantly. In addition, many of these do not support nested data while a good number can only handle one record per page. With these approaches, both setting up data extraction and maintenance is a problem

NLP-based tools are not ideal for genealogy either. These tools use clues from the structure of parsed sentences to identify data of interest. Most Web pages with genealogical information use a terse assortment of terms and fragments instead of complete sentences.

Ontology-based tools are adaptive. They have the advantage of being able to extract data without training on the format of those pages. Rather than using a wrapper for each site related to an application domain, they use an ontology that wraps all pages related to a domain. They are also resilient to changes in pages. The Data Extraction Group (DEG) at Brigham Young University has developed a data extraction method using ontologies. Instead of relying on formatting to find the data of interest, the DEG approach focuses on the data itself. Rather than creating a page-specific wrapper, we create a domain-specific wrapper (i.e. application domain, not Web domain) which we call an extraction ontology. These ontologies are conceptual models of the data that makes up a narrow domain of interest and will be discussed further in Chapter 3. Other researchers are also exploring the use of ontologies in data extraction. [dRC+98] outlines a system for extracting data in the university department domain from any Web site. [SMN01] uses the conceptual model as a framework and allows the user to create page-specific wrappers by specifying mappings between a Web page and the conceptual model.

The DEG system accurately extracts data in simple domains such as car advertisements and job postings. It uses clues from the format of the page to locate blocks of data, and the conceptual model to extract the data. The approach works well in the regularly formatted pages of newspapers' Web sites, but breaks down in the genealogy domain and others where the page formatting can be more complex (This will be discussed more in Chapter 5). The DEG has also worked on extraction from data within tables [ETL02, Tao03] and in the hidden Web behind forms [Che02]. In this

thesis, we describe our efforts to combine and improve these methods in order to produce a better data extraction system—one that works for genealogy.

Chapter 3

The Extraction Ontology

An extraction ontology is an augmented conceptual model of the information pertaining to a narrow domain of interest. We represent the conceptual model of our ontologies as Object-oriented System Model (OSM) diagrams as described in [Emb98]. In particular, we use a sub-model of OSM, the Object Relationship Model, which models objects and relationships between objects. We augment this model with data frames [Emb80], which are descriptions of the data types to which objects belong. Figure 1 shows the OSM diagram of our genealogy ontology.

ORM diagrams can contain object sets, which are drawn as boxes, and relationship sets, which are drawn as lines connecting the boxes. These make up the schema of a database. Each object set may be either lexical (represented by a solid border) or non-lexical (represented by a dashed border) depending on the contained objects. A lexical object is an object that is indistinguishable from its representation. The object set *Name* is lexical because a name is indistinguishable from its representation as a string of characters. Object sets containing numbers, dates, and even images are also lexical. Non-lexical objects are those that must be represented by a surrogate within a computer. The object set *Person* is non-lexical because there is no way to store a person in a computer. Instead, the system generates some identifier to represent the person.

In an extraction ontology, one object set is designated as the primary object set. This is the highest-level concept we are interested in extracting. An arrow and a dot

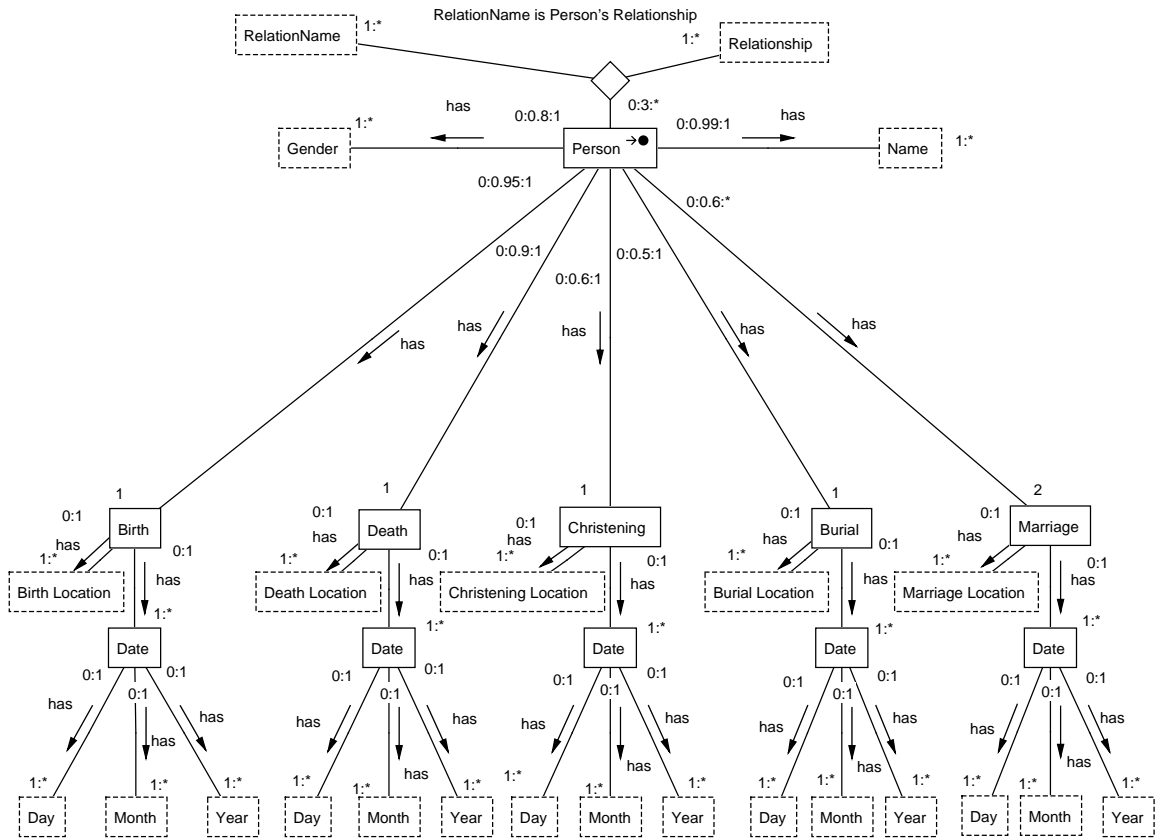


Figure 1: Extraction Ontology for Genealogy Domain

The relationships within a relationship set link one member of each object set to which the relationship set connects. On each connection to an object set, a relationship set is labeled with a participation constraint. A participation constraint indicates how many times an object in the connecting object set may participate in this relationship set. The participation constraint consists of a minimum, optional average, and maximum number separated by colons. In our notation, a star represents an arbitrarily large number. When the minimum and maximum are the same, they may be represented by one number. The participation constraint next to *Person* on the relationship set between *Person* and *Gender* indicate that a person object may be related to at most one gender. The average value of 0.8 indicates that on pages that include gender information, we expect to find a gender for 80% of the people. A person, we know, must have exactly one gender, but we must allow no gender because we may have partial data or mistakes in extraction.

In addition to the components seen in the OSM diagram, an extraction ontology also has a data frame for each object set. A data frame contains recognizers to identify data that belongs to an object set. These recognizers consist of extended regular expressions that match values, the context typically surrounding values, and keywords usually located close to a value. Our matchers support macro substitution and the inclusion of lexicons. This helps ontology engineers keep their regular expressions manageable.

Macros:

CapPhrase: ((([A-Z][A-Za-z]*)|of|the|on)(\s+((([A-Z][A-Za-z]*)|of|the|on)))){0,3}

Value Phrases:

1. {CapPhrase}\,(\s+){CapPhrase}(\.?)\,(\s+){State}
2. {CapPhrase}\,(+){State}
3. {CapPhrase}\,(+){CapPhrase}\,(+){CapPhrase}\,(+){Country}
4. {CapPhrase}\,(+){CapPhrase}\,(+){Country}
5. {State}

Figure 2: Regular Expressions for Matching Locations

Figure 2 shows the matchers used to find locations in our extraction ontology. The words within braces are labels of macros or lexicons defined elsewhere. The macro named *CapPhrase* matches strings of capitalized words while allowing some non-capitalized prepositions. *State* and *Country* are lexicons. These are external text files containing all the states and countries plus any abbreviations and variations of spelling we could anticipate. Our genealogy ontology also uses lexicons for given names, surnames, and months.

Besides the regular expressions in Figure 2, an ontology engineer optionally specifies other regular expressions to refine the matches. Figure 3 shows the expressions that correspond to the fifth value phrase in Figure 2. Exception expressions allow the system to rule out invalid matches, and right and left context expressions specify the context surrounding legitimate values. Our state lexicon contains two-letter postal abbreviations because they are commonly used to save space when city and county are given as in the first and second value phrase in Figure 2. Abbreviations are seldom used in situations that match this value phrase where the state is alone as it is in the fifth matcher. Without exception expressions in Figure 3, many English words (IN=Indiana, ON=Ontario, OR=Oregon) would be interpreted as state or province

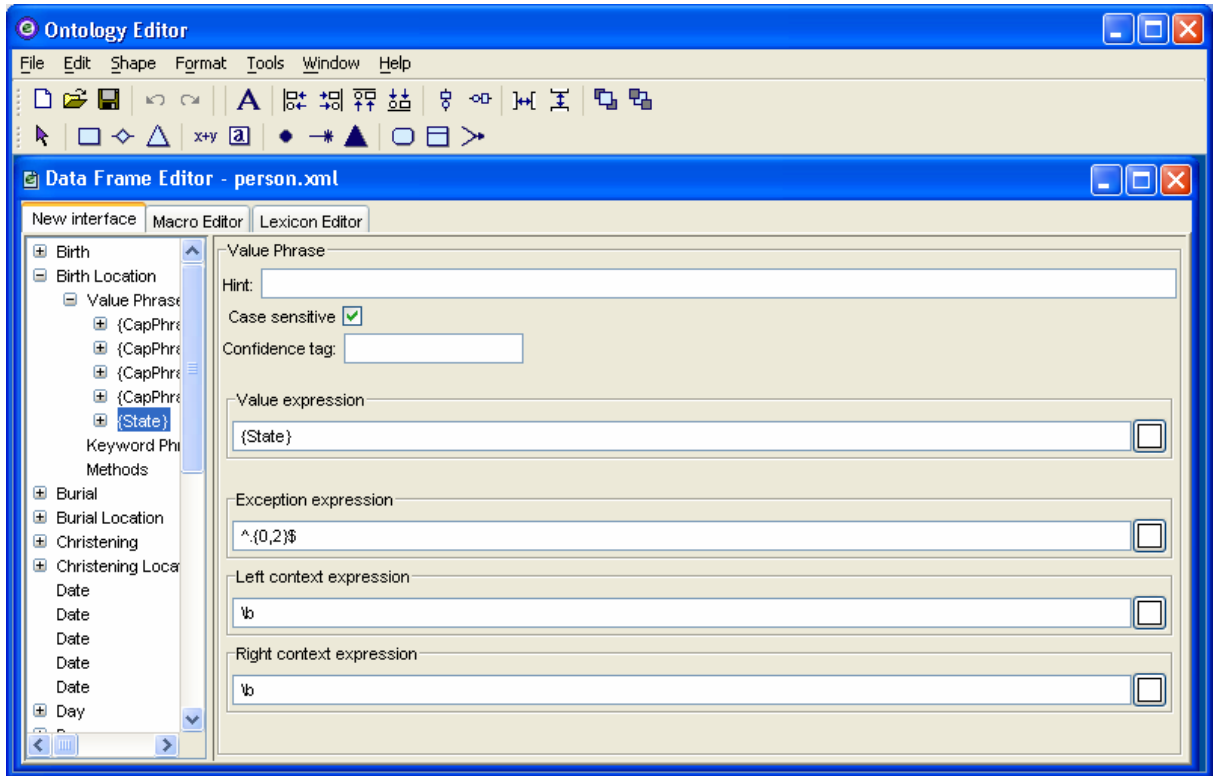


Figure 3: Exception and Context Expressions

names by this matcher. Figure 3 shows the expressions that correspond to the fifth value phrase in Figure 2. The value phrase containing {State} contains an exception expression that filters out matches that are two letters long. In Figure 3, we also have left and right context expressions that ensure the state match occurs on word boundaries.

We have previously claimed that ontology creation requires a few dozen hours of work. The DEG has recently developed tools to create and edit extraction ontologies graphically. It appears that this improvement has made it possible to create ontologies in even less time. Most of this time consists of testing and refining the ontology.

Chapter 4

The Extraction System

In this chapter, we will outline the components of our system, GeneTIQS, for extracting genealogy data. We have designed an architecture for querying extracted data from ontologically specified domains. Figure 4 shows the basic components and illustrates how data flows among the components in this architecture. We have implemented many (but not all) of these components to build GeneTIQS.

The system is initially given an *Ontology* (Chapter 3), which describes the application domain, which is genealogy for GeneTIQS. It then automatically creates a query form from the Ontology. The user can use this form to enter a *User Query* (Section 4.1). Given a *URL List* (Section 4.2), the *URL Selector* (Section 4.3) selects Uniform Resource Locators (URLs) that are likely to contain data relevant to the query. The *Document Retriever and Structure Recognizer* (Section 4.4) downloads a document for each URL from the Web and analyzes its structure to determine which extraction

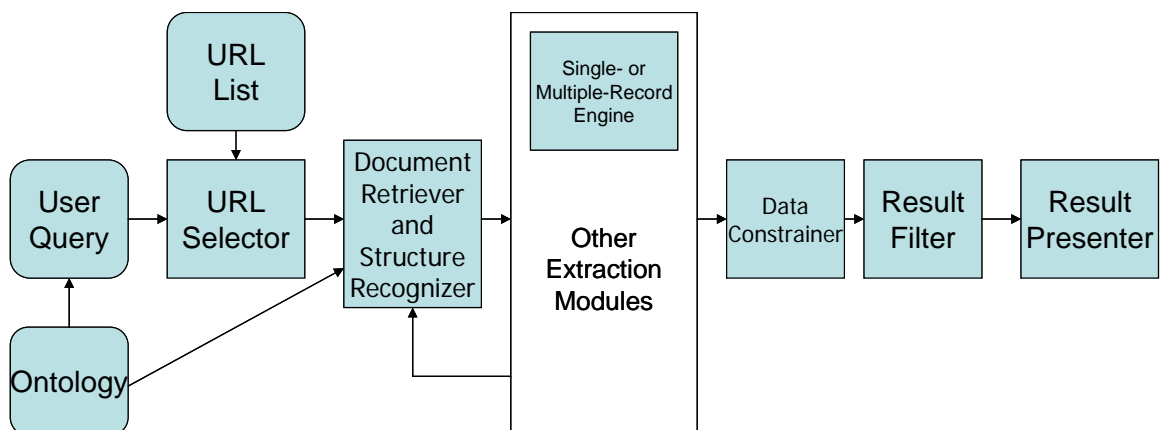


Figure 4: System Overview

module could best handle the document. The *Extraction Module* (Section 4.5) it selects then separates the document into records and produces candidate matches for all of the object sets in the ontology. In addition, the *Extraction Module* can submit URLs of additional sources of data it may find back to the *Document Retriever and Structure Recognizer*. The *Data Constrainer* (Section 4.6) takes candidate matches and converts them to objects and relationships that conform to the constraints in the *Ontology*. The *Result Filter* (Section 4.7) discards the data that does not match the *User Query*. As extraction continues for a list of URLs, the *Result Presenter* (Section 4.8) displays the information extracted so far to the user and gives the option to halt processing. In the remainder of this chapter, we will further explain these components.

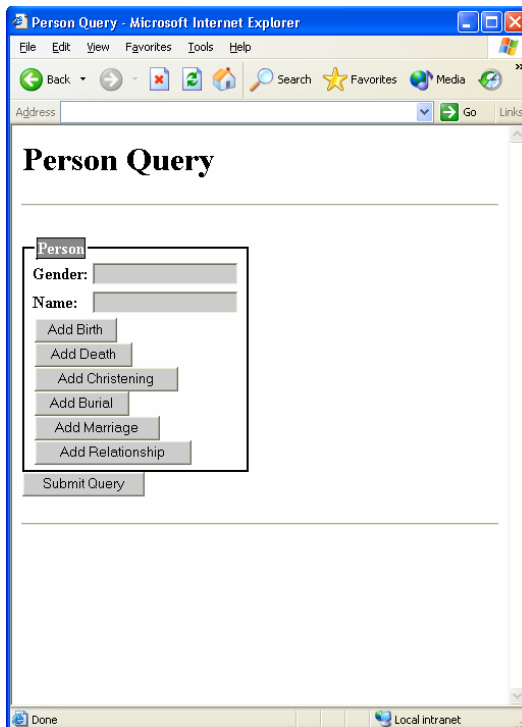
4.1 User Query

Our system will be of the most benefit in situations where many sites must be examined before filling the information need. We noted earlier that Google has indexed over 300,000 web sites dealing with Walker genealogy alone. With all of the surnames possible and allowing for overlap, there are probably tens of millions of genealogy pages on the internet. A query will allow GeneTIQS to focus its extraction on data that is likely to be of use to the user. If users are interested in extracting all of the data possible, they can simply enter a blank query.

When we initialize our system with a genealogy ontology, GeneTIQS generates an HTML form to accept user queries from the object sets in the ontology. This form allows for arbitrary conjunctive queries over the sources. Starting at the primary object set, the form generator follows the relationship sets to other object sets. When the form

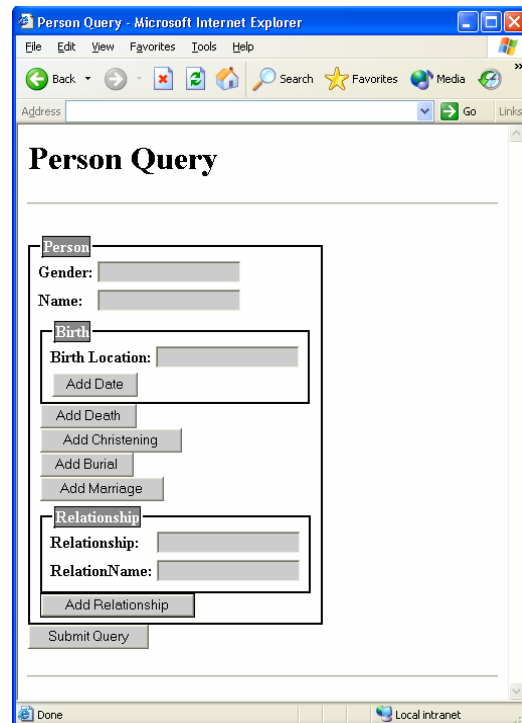
generator encounters an object set that is allowed only once, it will do one of two things. If the object set has a relationship that the form generator has not yet traversed, the form generator will generate a button for that object set. When this button is pressed, it will create a sub-form for that object set and its children. The button disappears to ensure that the sub-form only appears once. If the object set is not related to a new object set, the form generator simply creates a form field for that object set. When the form generator encounters an object set that is allowed many times, it creates a button for that object set. The button does not disappear when pressed so that it allows multiple sub-forms for that object set.

Figure 5 shows the form derived from our genealogy ontology in Figure 1. Since



The screenshot shows a Microsoft Internet Explorer browser window titled "Person Query - Microsoft Internet Explorer". The address bar is empty. The main content area has a heading "Person Query". Below the heading is a form with a section titled "Person" (highlighted with a box). This section contains two input fields: "Gender:" and "Name:". Below these fields is a vertical list of buttons: "Add Birth", "Add Death", "Add Christening", "Add Burial", "Add Marriage", "Add Relationship", and "Submit Query".

Figure 5: Generated Query Form



The screenshot shows a Microsoft Internet Explorer browser window titled "Person Query - Microsoft Internet Explorer". The address bar is empty. The main content area has a heading "Person Query". Below the heading is a form with a section titled "Person" (highlighted with a box). This section contains two input fields: "Gender:" and "Name:". Below these fields are buttons for "Add Birth", "Add Death", "Add Christening", "Add Burial", and "Add Marriage". The "Add Birth" button is expanded into a sub-form titled "Birth" (highlighted with a box). This sub-form contains a "Birth Location:" field and an "Add Date" button. Below the "Birth" sub-form is a section titled "Relationship" (highlighted with a box). This section contains two input fields: "Relationship:" and "RelationName:". Below these fields is an "Add Relationship" button. At the bottom of the form is a "Submit Query" button.

Figure 6: Expanded Query Form

Gender and *Name* do not have relationships to anything other than the primary object set and they can only occur once, they appear as form fields. Buttons allow sub-queries over the other object sets. As an example, Figure 6 shows the form as it appears after pressing the ‘Add Birth’ and ‘Add Relationship’ buttons. Since the ontology only allows only one Birth object for each Person, the ‘Add Birth’ button disappears when pressed. The ‘Add Relationship’ button, on the other hand, remains when pressed since multiple Relationship objects are allowed.

4.2 URL List

Our system needs a list of relevant information sources to help it answer user queries. We provide it with a list of URLs that point to documents that are relevant to the domain of interest. Two alternatives exist for creating this list. A human can browse the Web and manually judge relevancy, or a targeted web crawler programmed to classify pages automatically based on relevancy to the domain of interest can generate the list. A web crawler must be courteous to the owners of servers and other users by limiting frequency of access and respecting the limitations set forth by the server owners as specified in the robots.txt file. [Xu03] provides a classifier that judges relevancy with decision trees trained on ontological features and presents experimental data on cars and obituaries. In a production system, the classifier would probably need to be tuned to maximize its performance on the domain of interest. Since source discovery is not the in the scope of this thesis, we simply created our URL lists by hand.

4.3 URL Selector

Rather than search all these pages for each query, a *URL Selector* can select the URLs likely to contain information relevant to the current query. A *URL Selector* would rely on summary information stored along with the URLs during the source discovery phase to rule out improbable sources and prioritize the probable ones. This would reduce processing time, network utilization, and the impact on remote Web servers, allowing faster answers to queries while minimizing cost. Since our experimental URL lists are relatively short and do not include summary information, in our prototype, we have implemented a dummy *URL Selector* that simply selects all URLs for processing. We leave development of a smart *URL Selector* for future work.

Once URLs are selected for extraction, processing could easily be distributed across multiple computers. All this would require is a framework for dividing the selected URLs among processors and gathering the resulting data. In our experiments, GeneTIQS processed about 50 kilobits per second on an 800 MHz machine. About a thousand systems could be extracting simultaneously without overwhelming a T3 connection to the internet. Although not implemented in our prototype system, this optimization would be crucial to providing timely results in a production system.

4.4 Document Retriever and Structure Recognizer

Once the system has a URL likely to contain data relevant to the query, GeneTIQS is ready to start processing. First, the *Document Retriever* contacts the host site and downloads the document identified by the URL. The *Structure Recognizer* then determines which *Extraction Module* can best process the page.

In order to make the system easily expandable, the *Structure Recognizer* maintains a list of all *Extraction Modules* available. When presented with a document to analyze, it submits the document to each module and asks for a confidence measure. This measure represents the module's confidence that it can process the given document. The *Structure Recognizer* simply forwards processing to the module with the highest confidence.

In addition to accepting the URLs given by the *URL Selector*, the *Document Retriever* also accepts URLs from the *Extraction Modules* as they find other sources of data. For example, an *Extraction Module* that processes forms would submit links to the pages that result from form submission. This could also be used to follow links on any web page.

4.5 Extraction Modules

The *Extraction Modules* make up the core of the GeneTIQS system. They are responsible for extracting attribute-value pairs from the document and returning them grouped by record. Each module implements a simple API that exposes two methods. One analyzes the structure of a document and determines the module's confidence that it can process this document. The *Structure Recognizer* uses this method to select an *Extraction Module* for each document. The other method initializes the extraction process.

The main module we built to extract from single-record and multiple-record documents cannot extract data perfectly from every page on the Internet. Some pages employ formatting that the main module does not handle well. Other Internet resources

cannot be accessed through the usual HTTP request. To expand the system to accept these sources, other *Extraction Modules* can be added. These modules can be specializations of current modules, or provide access to a new class of information sources.

4.5.1 Single-Record or Multiple-Record Engine

The majority of web sites in the genealogy domain contain single-record and multiple-record documents. This module extracts data from these documents in two steps. First, if it is a multiple-record document—a document that contains information on multiple people—the module splits it into sections containing one record each. This process as described in Chapter 5 is a major contribution of this thesis. Once it separates the records, the module finds the candidate matches for the object sets in the ontology using the recognizers found in their data frames. Since we intend this module to be the generic processor for HTML documents, it outputs a low confidence score (0.1) for any documents that it can parse. This allows more specialized modules to override it easily.

4.5.2 Specialized Extraction Modules

A specialized extraction module extracts data from a class of pages that a more general module handles inadequately. An example would be a module to extract data from HTML tables. Some web sites, particularly those generated from databases, present data as tables containing headers that identify the data in the columns below. Each record occupies one row of the table. This configuration causes problems for our standard extractor because it cannot find the keywords that correspond to data on the

page. Pages containing tables also present an opportunity for increased accuracy: Once a column is identified as containing a given attribute, a table module can accept all cells in that column as matches for that attribute. A module specialized for tables can adapt to variations and misspellings that would break the regular expressions of the main module and can reject extraneous matches found elsewhere in the document. [Tao03] explains techniques to extract data from tables as well as techniques to distinguish a valid table from HTML tables used merely for formatting. In the future, we hope to wrap these techniques to build a specialized module for table processing.

Specialized extraction modules need not be restricted to ontology-based techniques. If the accuracy of extraction is unsatisfactory for a certain site that contains enough data, we could create a site-specific wrapper using a non-ontological technique for that site and place it in a module. This module would create its confidence score for a document simply by determining that the URL comes from a pre-identified site for which the site-specific wrapper was created.

4.5.3 Other Extraction Modules

A possible module to provide access to a new class of source information would be one to extract data from forms. Although attempts have been made at extracting all information behind a form using statistical methods [LES+02], this approach cannot handle submission of form fields that require text input. [Chen02] describes a technique to map the schema of a form to the schema of a global ontology. Once these mappings are discovered, it takes terms from a query to fill the fields of the form. For those form fields that do not match a query term, it combinatorially attempts all possibilities. In this way, it can create multiple submissions for one form. Since each

of these submissions could result in a page with any format, a module using this technique would encode each submission as a URL with an HTML GET query appended to the end and add it to the *Document Retriever's* queue. Wrapping this technique in an extraction module might allow the system to handle multiple-step forms not currently supported.

Other sources of information could be exploited in this way. A module could easily extract data from GEDCOM files found on the Web. Modules could be created to accept data from databases, web services, or the semantic web. With the exception of databases, these technologies are not widely used—particularly in the genealogical domain. As they gain popularity, they could become rich sources of information, motivating the creation of modules to support them.

4.6 Data Constrainer

Once an *Extraction Module* identifies attribute/value pairs in the document, the *Data Constrainer* binds them to objects and relationships that conform to the constraints in the extraction ontology. Two problems can occur during this binding process: conflicts and ambiguities. A conflict arises when a portion of the document matches the recognizers for two or more object sets. An ambiguity occurs when more candidate matches exist for an object than are allowed by its participation constraints. The DEG has developed heuristics for managing these problems in its Ontos system [ECG+99]. Members of the DEG are currently constructing a framework for improving these heuristics and adding new heuristics [Wes03]. We have delayed significant improvements to these heuristics pending the completion of this initiative. However,

we have made one improvement. One heuristic uses keyword proximity to identify correct matches, which fails, of course, when no keywords exist in the ontology or when none exist in the document. For this case, we refined the heuristic to take advantage of the tendency for related information to occur in close proximity in a document. When no keyword exists for an object, we use the proximity of a related object instead of keyword proximity.

4.7 Result Filter

Once the *Data Constrainer* produces objects and relationships from the matches in the document, the *Result Filter* can evaluate the applicability of each record to the query posed by the user. It discards any records that do not satisfy the query.

The screenshot shows a web browser window titled "Query Results - Microsoft Internet Explorer". The address bar shows the URL "http://bluebell8080/gene/resultframe.html". The main content area is titled "Query Results" and contains a table with columns: Gender, Name, Birth Location, Day, Month, Year, Death Location, Day, Month, Year, Christening Location, Day, Month, Year, Burial Location, Day, Month, Year, Marriage Relationship, and Source. The table lists several individuals, including Adeline Theresa Marie Adams, Lilian Holmes, Brozwyn Johnson, John Walker, and John Walker. Below the table, there is a detailed view for "Adeline Theresa Marie Adams" with the following information:

- Date of Birth : February 9, 1875
- Place of Birth : Plymouth, Devon, England
- Father : [George Abnelion Adams](#)
- Mother : [Margaret Mary Ann Hooper](#)
- Date of Marriage : September 9, 1897
- Place of Marriage : Unknown
- Spouse : [George Alfred Hulmer-Bird](#)
- Date of Death : July 13, 1953
- Place of Death : Manningtree, Essex, England
- Children : [Adeline Dorothy](#), [Ceoil George](#), [Constance Margaret](#)
- Generation Number : 4

Figure 7: Extraction Results Presented to the User

4.8 Result Presenter

Once GeneTIQS has extracted data that conforms to the ontology and the user's query, the *Result Presenter* displays this data on the user's web browser. Figure 7 shows an example. The browser window is split into two frames: The top frame contains the extraction results while the bottom frame shows the source documents. The *Result Presenter* creates the schema of the tables in the result pane with the algorithms for generating nested XML schema in [EM01, EM03]. Each row of the table contains the extracted information on one person. The table's columns consist of attributes such as *Gender*, *Name*, *Birth Location*, and so on. The values of the attributes for which only one value may exist appear directly in their column. For attributes that may occur many times, a *Show* button appears instead. When the user clicks this button, a new table emerges below the current row showing the values of this attribute. Figure 8 shows the names of relatives of Adeline Theresa Marie Adams, which appeared when we clicked the *Show* button in Adeline's row in the *Relationship* column.

The screenshot shows a web browser window titled "Query Results - Microsoft Internet Explorer". The address bar shows the URL: http://bluebell.00001/gene/service/edu.byu.dog.query.ResultPresenter. The main content area is titled "Query Results" and contains a table of query results. The first row is for Adeline Theresa Marie Adams, with columns for Gender, Name, Birth Location, Day, Month, Year, Death Location, Day, Month, Year, Christening Location, Day, Month, Year, Burial Location, Day, Month, Year, Marriage, Relationship, and Source. The Relationship column contains a "Show" button. Clicking this button has expanded the Relationship column into a sub-table showing her relatives:

Relationship	RelationName
Father	George Abzalom Adams
Mother	Margaret Mary Ann Hooper
Spouse	George Alfred
Children	Adeline Dorothy Cecil George

Below this, the main table continues with other individuals: Lilian Holmes Lilian Holmes, Bronwyn Johnson, John Walker, and John Walker. Each row has a "Show" button in the Relationship column.

Figure 8: Extraction Results with Relationship Expanded

Because a user may not wish to wait for a long extraction process involving many pages, the *Result Presenter* also includes a *Stop* button. If users find the information they want they can stop, or if they find that the query needs refinement, they can stop the process before it completes and begin again.

Chapter 5

VSM-Based Record Separation

When we initially investigated building a system for extracting genealogical information, we discovered that record separation was a major obstacle. Previous efforts at solving the problem proved ineffective on the variety of pages found in the genealogical domain. This chapter explains the problem of record separation, previous work in record separation, the basics of Vector Space Modeling (VSM) as applied to record separation, an improved method of record separation, and some refinements we have made to resolve problems we encountered.

5.1 The Problem of Record Separation

In order to extract the information related to each person in a genealogical document, the computer needs to separate the document into records. Each record should contain information on only one person. This greatly simplifies the task of selecting values and linking them together as objects and relationships. We divide web pages into three categories based on how they present information: single-record documents, simple multiple-record documents, and complex multiple-record documents. Some web pages include information on only one person per page (single-record documents). Others list many people's information on one page. The information on these pages must be split appropriately. Some multiple-record documents have a simple pattern that is constant throughout the page. One HTML tag consistently separates the records in these documents (simple multiple-record

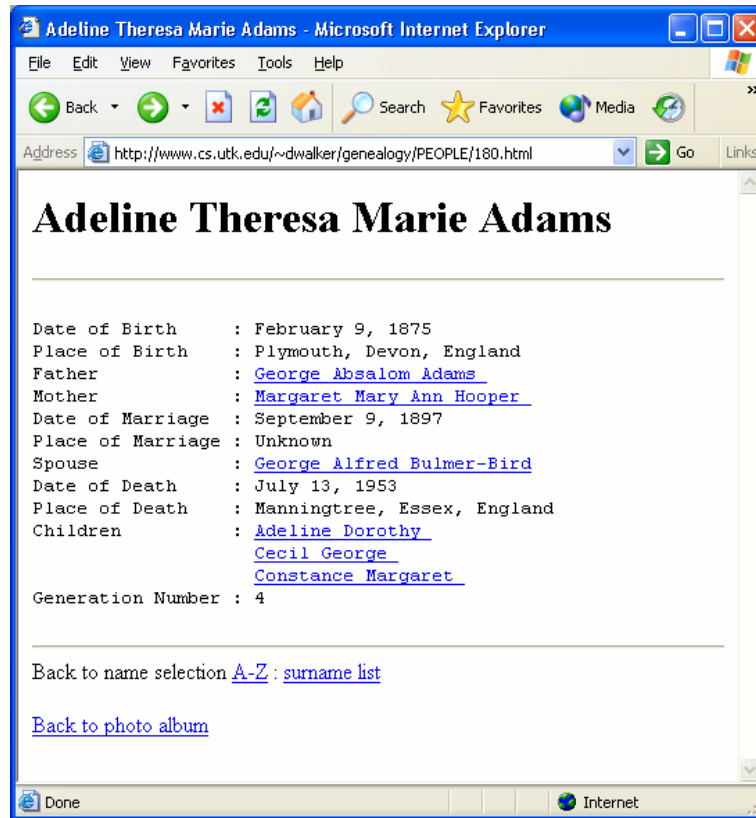


Figure 9: Single-Record Document

documents). Other multiple-record documents are not separable by one HTML tag because they have more complex formatting (complex multiple-record documents). Since all three of these categories are common in the genealogy domain, we want to be able to extract from all three categories.

Figure 9 through Figure 11 show an example of each category of document. The document in Figure 9 contains only one record. The document in Figure 10 contains many records, each separated by an `<hr>` tag. The document in Figure 11 is an example of a more complex separation. Each row in a table in the document corresponds to a generation. Each row has a different number of cells containing data. In some cases, these cells contain information about a man as well as his wife. For example, the last cell in the third row is for James Walker, but it also contains

information about his wife, including her parents and death date. On the other hand, the cell immediately to the left contains information on only one person, his sister Elyzabeth.

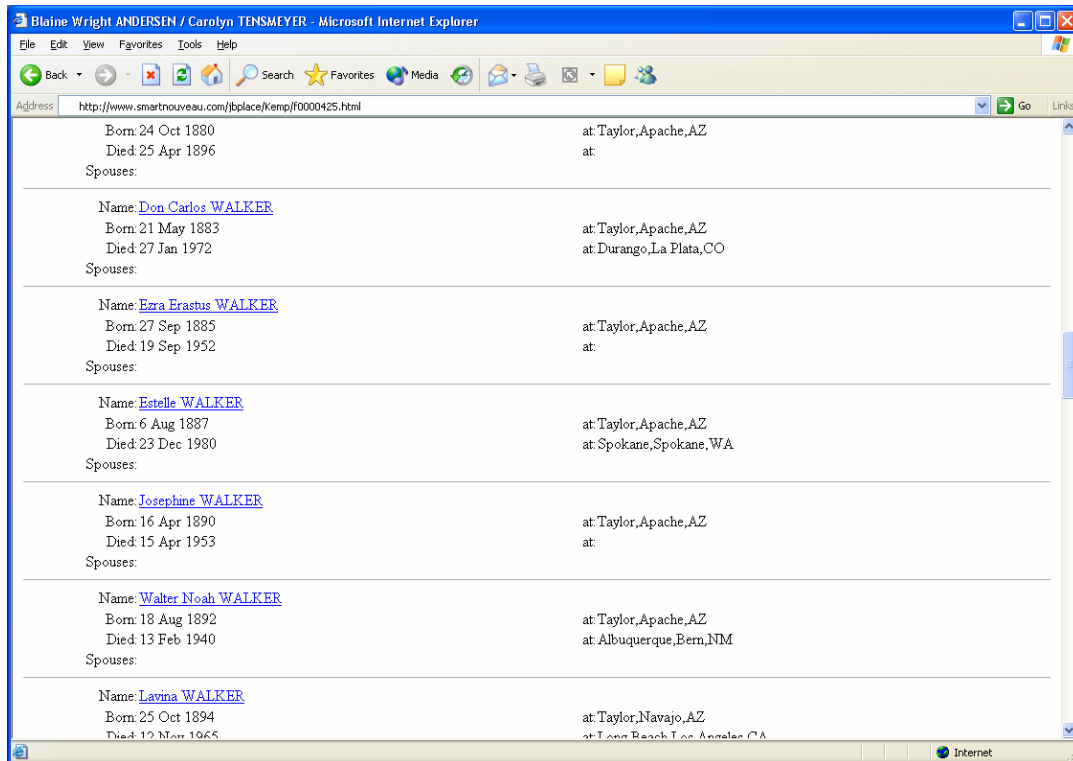


Figure 10: Simple Multiple-Record Document

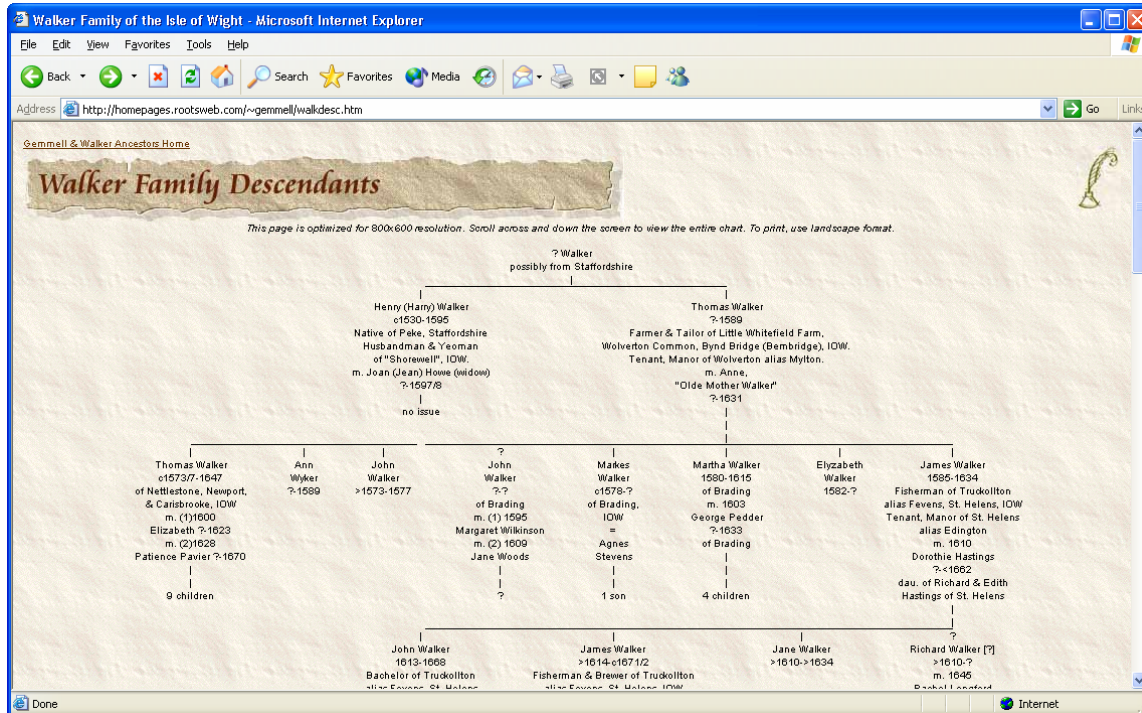


Figure 11: Complex Multiple-Record Document

5.2 Previous Work

A previous effort at record separation proved successful in certain domains [EJN99]. The record separator in [EJN99] begins by parsing the document's HTML into a Document Object Model (DOM) tree. It then locates the subtree containing the objects of interest by finding the node in the DOM tree with the most child nodes (highest fan-out). It determines a record separating tag using a combination of five heuristics and uses occurrences of this tag to partition the document into records. [BLP01] expands on these techniques with different heuristics that were more accurate on the pages in which they were interested.

There are three problems with this approach. First, it assumes that all input documents are multiple-record documents. When given a single-record document, the

record separator attempts to find some separation and thus always gives incorrect results. Second, on many pages, the highest fan-out heuristic selects a portion of the document that does not contain the data of interest. This often occurs in documents containing navigation bars with many links, or selection boxes containing many options. The problem is worse when a page contains a small number of records. Third, this approach separates on only one level of the tree. Because of the hierarchical nature of genealogical data, many Web sites use nested lists, tables, or some other formatting to denote ancestor or descendent generations. This approach cannot handle complex multiple-record documents.

Two clues aid in record separation: structure and content. A well-designed page will have a format that assists readers in distinguishing records (e.g., tree nodes in Figure 11). Readers can also determine where record divisions occur based on the data they contain (e.g., a second name and another birth date denotes a new person). The approaches mentioned above primarily use structural clues in separation, although the ontology heuristic described in [EJN99] does use counts of data frame matches to produce its candidate tag. Structural clues alone, however, do not work for genealogy because there are usually not enough records on a page to describe the complex patterns used. It would be difficult to create an accurate record separator based purely on content. In a domain where data always appears in a certain order or all data is always present, it may be straightforward, but in genealogy, this is not the case. Missing data, imperfect matchers, and unpredictable order combine to make this approach infeasible.

[EX00] introduces a technique utilizing both kinds of clues. This technique is explained further in [EX01]. The hybrid method of [EX00] and [EX01] first produces

candidate records using the techniques in [EJN99]. Next, the method refactors records based on the data they contain according to VSM measures of each record. It detects headers and footers and determines whether they contain information that pertains to all records on the page. It merges adjacent records containing complementary data and splits records containing too much data. Because it still uses the highest fan-out heuristic to locate the information of interest, the approach in [EX00] and [EX01] fails to resolve completely any of the three problems mentioned above. For single-record documents, it loses all data outside the highest fan-out node, and although it could theoretically recombine any incorrectly split information within the single-record document to recover the correct record, conflicting matches to names and names of relatives prevent it from doing so. This method does nothing to recover from cases when it chooses the wrong subtree for separating. Finally, since it discards all hierarchical information when creating the candidate records, it must split candidate records based only on content and can handle only the simplest cases.

5.3 Vector Space Modeling

Like [EX00] and [EX01], we use VSM for separating records. Unlike [EX00] and [EX01], however, we do not first seek the highest fan-out tree, neither do we limit ourselves to separation based on one HTML tag. VSM comes from the field of information retrieval. A set of features from a document makes up the values in a vector from which useful cosine and magnitude measures are derived. When doing separation, we use the object sets from the extraction ontology as dimensions in the vector space.

First, we create a vector that represents a prototypical record of genealogical data from the participation constraints in the ontology. [EX01] calls this the Ontology Vector (OV). We use the average from the participation constraints when given. When no average is given, we infer an average midway between the minimum and maximum values using a sufficiently large number (100) to replace stars. We do not include dimensions for all object sets; instead, we include dimensions for those object sets most closely related to the primary object set. These object sets give us the information most helpful for separating instances of the primary object set while more indirectly related object sets have more of a potential for ambiguity and conflicts or for being completely unrelated. The dimensions selected by this algorithm from our genealogy ontology appear in Table 1 along with the averages that make up the OV and the vector itself.

For each candidate record, another vector records the matches found in a portion of the document. [EX00] calls this the Document Vector, but since we also use this measure for various portions of a document, in this thesis we will call it the Subtree Vector (SV) to avoid confusion. The number of matches for each object set found within the subtree rooted at a node makes up the vector for that node. We judge each SV by its cosine score and magnitude score.

Large values along any dimension skew these measures, so object sets that are allowed many times tend to have too much weight in these measures. We normalize all vectors prior to finding VSM scores by dividing each value in the vectors by the

	Gender	Name	Birth	Death	Christening	Burial	Marriage	Relationship	Relation Name
Average	0.80	0.99	0.95	0.90	0.60	0.50	0.6	3.0	3.0
Ontology Vector	{ 0.8, 0.99, 0.95, 0.9, 0.6, 0.5, 0.6, 3.0, 3.0 }								

Table 1: Ontology Vector

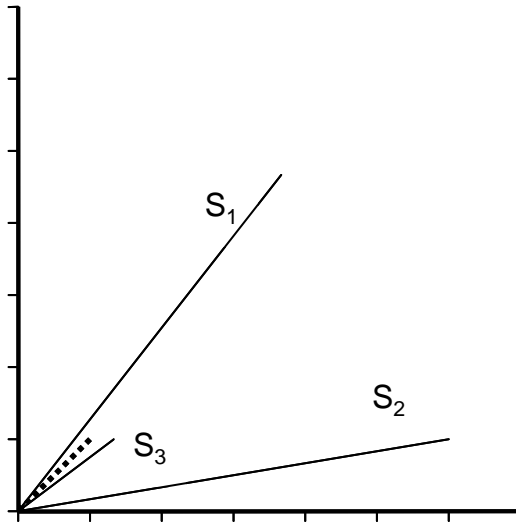


Figure 12: Illustration of VSM Measures

corresponding average in OV. This reduces the effect that attributes for which many values are expected have on record separation. We normalize OV in the same way (the value for each dimension becomes one).

The cosine of the acute angle between any two vectors in n -space is the dot product of the two vectors divided by the product of their magnitudes. This provides a reliable method of determining the similarity of SV to OV. Cosine measures close to one occur when the angle between two vectors is small. A small angle between SV and OV shows that the subtree likely contains data that relates to the ontology. The magnitude of SV divided by the magnitude of OV yields a rough estimate of the number of records in SV, which is accurate enough to decide whether to split a record.

Figure 12 shows vectors for three sections of a hypothetical web document projected into two dimensions for simplicity. The dotted line represents the OV. The angle between S_1 and OV is small, meaning that the cosine measure of S_1 is close to one

so we know that the data in S_1 closely matches the ontology. The magnitude of S_1 indicates that it contains many records (probably four or five) and needs to be split. On the other hand, S_2 has a lower cosine measure. It contains either partial information or information not pertaining to genealogy. S_3 has a high cosine measure and a magnitude close to one. These scores show that it is probably a single record related to genealogy.

5.4 Improved Method of Record Separation

Our improved method of record separation confronts the problems of [EX01] by traversing more of the tree and by maintaining format information throughout the process. Starting with the root of the tree, it evaluates the subtree rooted at each node in a depth-first traversal. If its magnitude measure is less than a threshold value (e.g., 1.8), we accept it as a record. If not, we split the subtree using the separator tag heuristics of [EJN99]. In cases where these heuristics fail to find a separator (usually when a node has fewer than four child nodes), we simply use the subtrees that are children of the current node. We then use a technique from [EX01] to recombine these subtrees where appropriate. Finally, we discard the subtrees with low cosine scores (less than 0.6) and repeat the process with the remaining subtrees.

Table 2 shows a selection of the DOM tree representing the document in Figure 11 as well as the corresponding SVs, cosine scores, and magnitude scores. At the first level, there are only two children: <!DOCTYPE> and <html>. <!DOCTYPE> has a low cosine score so we throw it out. <html> has a high cosine score so we keep it. Since <html> has a large magnitude, we split it into <head> and <body>. We discard <head> and split <body>. Because <body> has many child nodes, we find a separating tag, <div>, and divide the children accordingly. All but the second <div> tag are dropped because of their cosine scores. After repeating this process and dividing based on table rows and then table cells, we eventually start finding individual records. The <td> in the example is a table cell that happens to contain one record. Other table cells in this document actually contain multiple records and need to be split further.

5.5 Problems and Refinements

As we implemented this algorithm, we encountered three problems that limited its effectiveness. First, differences between our ontology and the schema used within documents often caused cosine measures to be too low. Second, the over-richness of

	Subtree Vector	Cosine	Magnitude
<!DOCTYPE...>	{ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 }	0.00	0.00
<html>	{ 0.0, 149.0, 89.0, 76.0, 0.0, 0.0, 48.0, 23.0, 23.0 }	0.97	111.59
<head>	{ 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 }	0.58	0.58
...			
</head>			
<body>	{ 0.0, 148.0, 89.0, 76.0, 0.0, 0.0, 48.0, 23.0, 23.0 }	0.97	111.15
<div>	{ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 }	0.00	0.00
...header...			
</div>			
<div>	{ 0.0, 146.0, 88.0, 76.0, 0.0, 0.0, 48.0, 23.0, 23.0 }	0.97	109.98
...			
<td>	{ 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0 }	0.99	1.06
...			

Table 2: DOM Tree Selections and Vector Scores

data, particularly in single-record documents, often caused magnitude measures to be too high. We introduced refinements into our algorithm to cope with these problems. Third, when separating simple multiple-record documents, our algorithm was sometimes outperformed by the old method because it did not take advantage of simple patterns in the data.

5.5.1 Schema Differences

Not everyone agrees about what attributes are needed to describe a person in genealogy. We designed our genealogy ontology to hold as much data as we were likely to find in genealogy pages on the Web. It contains many attributes such as Gender, Burial, and Christening for which most Web sites do not include data. The fact that these attributes do not appear on a page does not mean that the page is not about genealogy. However, it does affect the cosine scores of the document. This can cause valid records to have cosine scores slightly below the threshold resulting in valid records' being discarded.

The problem is that a page author's conceptual model does not always perfectly align with our conceptual model. If we detect these differences in schema at the page level and compensate for them, we can more accurately find the records within the page. We do this by pruning dimensions in our vector space. In order to detect which object sets do have matches in a document's schema, we count the object set matches and prune any dimension with no matches. Since a few erroneous matches are possible, we also prune dimensions with counts less than 5% of the average count, weighted according to the participation constraints. The document in Figure 11 is one that shows the need for this refinement. This document contains only names, birth dates, death

dates, and some marriage events. Without any genders, burial dates, christenings, or relatives, it took about five records forced together to achieve the required magnitude score to make a record. Once we pruned out these dimensions, we were able to discover the correct records.

5.5.2 Over-richness of Data

Single-record documents tend to include more complete information than multiple-record documents. Because single-record documents tend to give children and parents of an individual, it is not unusual to find seven name matches in one single-record document. Single-record documents may also repeat information or have multiple instances of one keyword. Magnitude measures for records in single-record documents are therefore much higher than measures for multiple-record documents. To overcome this, we programmed our separator to require a higher magnitude to split a document than to split within a document. For example, the single-record document in Figure 9 contains seven names and the keywords for birth and death twice. Because of these high counts, the magnitude of this record is high enough to split the record. If these counts were part of a larger document, this would probably mean multiple records

5.5.3 Missed Simple Patterns

Simple multiple-record documents are distinguished by a simple pattern in the formatting. On some simple multiple-record documents, our old technique of record separation is able to produce more correct records than this new method. In any document, some records contain more or less details than others. Sometimes our matchers do not accept all the valid data, such as when names are incomplete or contain

words not in our lexicon. At times, this variation is enough to cause our record separator to erroneously discard or split a valid record. We can take advantage of the pattern in simple multiple-record documents if we can detect them. We do so when we split a subtree by counting the ratio of records with sufficient cosine scores and low enough magnitudes to be a single record. If there are more than three records and at least two-thirds of them are single records, we consider all of them to be single records. As a further refinement, we eliminate headers and footers by discarding records with low cosine scores at the head and tail of the list. Without this refinement, for example, our record separator would discard the records for Melissa Anne Knuteson and Dennis M Knuteson in Figure 13 since they contain so little information. There are enough valid records before and after these two records for this simple heuristic to detect the pattern and include them.

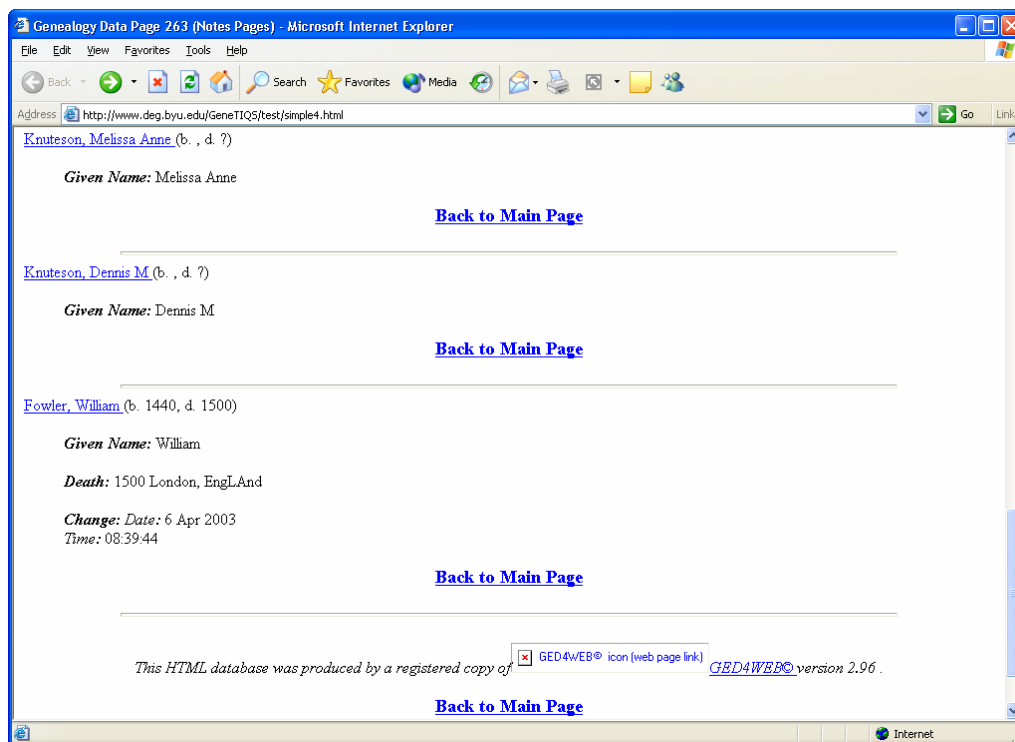


Figure 13: Document with a Simple Pattern

Chapter 6

Experimental Results

While implementing GeneTIQS, we used a few example documents to debug our code. Once our system was ready, we gathered sixteen other documents to test our algorithm and made further refinements. When our system performed adequately on this tuning set, we were confident it would perform well on any pages from the Web. Two components of GeneTIQS lend themselves to testing: the extraction module and the data constringer. Since the data constringer depends on heuristics that are currently being improved, we have focused our testing on the extraction module, specifically our improvements to record separation.

6.1 Test Documents

We gathered test documents by searching the Web for common surnames and genealogy. We took the first few results that contained data. To ensure stability and reproducibility throughout our test and to reduce load on the web hosts, we created a local cache of our test pages. Appendix A lists these pages and their sources. Since our goal for this system is for it to handle many different web pages, we focused on collecting a wide variety of pages.

When collecting pages, we found that there are about three generators commonly used for genealogical web pages. Since we were interested in evaluating our system on a variety of sources, we included only a few pages generated by each program.

Although this limited the number of documents in our test sets and made it difficult to find more, we felt this was important.

Some pages contain close to a hundred records while others contain just one. If we had tested on these documents as they were, performance on a few pages would have dominated our results even within one category. We did two things to reduce this skewing. First, we trimmed long documents to between ten and twenty records. This range allowed us to avoid truncating subtrees such as a list of children. We were careful to preserve any footers that might exist on each document. Limiting the number of records per page also made it easier for us to do our manual evaluation of results. We reasoned that if we could extract the first records from a page, we could probably extract the rest as well. Second, any time we found a source of single-record documents, we collected three documents from that site. Unfortunately, trimming the pages in our test set significantly reduced the number of records in our test set. Because of this, our test set may not appear to contain enough examples to provide a statistically significant sample. However, we felt that this did not compromise the overall accuracy of our test since the records we retained had the same characteristics of the records we removed from the same page.

6.2 Results

We divided our test documents into three groups: single record documents, simple multiple-record documents, and complex multiple-record documents. For each group of documents, we created a URL list and ran our system with an empty query to obtain all the records. To test record separation, we compared these records to what

	<i>records</i>	<i>returned</i>	<i>correct</i>	<i>precision</i>	<i>recall</i>
single1	1	1	1	100.00%	100.00%
single2	1	1	1	100.00%	100.00%
single3	1	1	1	100.00%	100.00%
single4	1	1	1	100.00%	100.00%
single5	1	1	1	100.00%	100.00%
single6	1	1	1	100.00%	100.00%
single7	1	1	1	100.00%	100.00%
single8	1	4	0	0.00%	0.00%
single9	1	1	1	100.00%	100.00%
single10	1	1	1	100.00%	100.00%
single11	1	1	1	100.00%	100.00%
single12	1	3	0	0.00%	0.00%
single13	1	1	1	100.00%	100.00%
single14	1	1	1	100.00%	100.00%
single15	1	1	1	100.00%	100.00%
single16	1	1	1	100.00%	100.00%
single17	1	1	1	100.00%	100.00%
single18	1	1	1	100.00%	100.00%
single19	1	1	1	100.00%	100.00%
single20	1	1	1	100.00%	100.00%
single21	1	1	1	100.00%	100.00%
Total	21	26	19	73.08%	90.48%

Table 3: Single-record Document Results

should have been produced. Because of the nested nature of genealogical data, this was not always simple. A name by itself in some contexts might be considered a record while in other contexts it may just be the name of a relative within a valid record. As a general rule, we considered information about a relative to be a distinct record if it contained more than just a name.

6.2.1 Single-record Documents

We tested 21 single-record documents. As Table 3 shows, our record separator correctly handled most of these documents, resulting in 90% recall and 73% precision. This success is due to the refinement we made to compensate for over-richness of data. In two cases (*single8* and *single12*), data was rich enough to overwhelm this refinement. Attempting to split these records destroyed the relationships within them

	<i>records</i>	<i>returned</i>	<i>correct</i>	<i>precision</i>	<i>recall</i>
simple1	19	20	19	95.00%	100.00%
Simple2	19	17	17	100.00%	89.47%
Simple3	11	11	11	100.00%	100.00%
Simple4	9	9	9	100.00%	100.00%
Simple5	12	13	11	84.62%	91.67%
Simple6	12	11	10	90.91%	83.33%
Simple7	14	10	10	100.00%	71.43%
Simple8	5	7	5	71.43%	100.00%
Simple9	14	14	14	100.00%	100.00%
Simple10	15	15	15	100.00%	100.00%
Total	130	127	121	95.28%	93.08%

Table 4: Simple Multiple-record Document Results

and produced a large number of incorrect records, which explains the relatively low precision of 73%. We could increase the threshold to cover more of these cases, but raising it too much would cause multiple-record documents not to be split. This refinement performed fairly well, but it is likely that a different approach would be needed to produce better results.

6.2.2 Simple Multiple-record Documents

Table 4 shows the results of our experiments on simple multiple-record documents. By using our refinement for exploiting patterns in simple documents (Section 5.5.3), we were able to return 7 more correct documents than we would have

	<i>records</i>	<i>returned</i>	<i>correct</i>	<i>precision</i>	<i>recall</i>
simple1	19	22	19	86.36%	100.00%
simple2	19	20	0	0.00%	0.00%
simple3	11	14	11	78.57%	100.00%
simple4	9	10	9	90.00%	100.00%
simple5	12	16	12	75.00%	100.00%
simple6	12	23	9	39.13%	75.00%
simple7	14	22	13	59.09%	92.86%
simple8	5	10	0	0.00%	0.00%
simple9	14	16	14	87.50%	100.00%
simple10	15	16	0	0.00%	0.00%
Total	130	169	87	51.48%	66.92%

Table 5: Results for Old Record Separator

otherwise and achieved 95% precision and 93% recall. For comparison, Table 5 shows the results of using the old record separator with the same set of documents. This comparison is only possible for simple multiple-record documents because the old record separator does not work on the other categories. The old record separator completely failed on three test documents. While processing *simple2* and *simple10*, the record separator discarded relevant data from each record. With *simple8*, the highest fan-out heuristic failed to choose the correct subtree for extraction. In most cases, our new approach was able to produce the same correct records while returning fewer incorrect records. In some documents, we lost the first record because it did not have a high enough cosine score and was misinterpreted as part of the header. In *simple7*, less than two thirds of the records were acceptable as single records so the algorithm did not detect the simple pattern. From these results, we see that our record separator compares favorably with the old record separator.

6.2.3 Complex Multiple-record Documents

Since most of the documents on the Web fall into this category, performance on complex multiple-record documents is critical. Table 6 shows our results for this category. Given the difficulty of the task, we consider 92% recall and precision to be a very good result. The most common problem we encountered stemmed from conflicting matches. While doing record separation, our system has no way of knowing whether a name is a member of the *Name* object set or the *RelationName* object set while doing record separation and must consider it a potential match for both object sets. This becomes a problem when recombining fragments of a record. Figure 14 shows two records produced from *complex4* that should have been recombined. The

first record has matches for *Name*, *Birth*, and *Marriage*. The second has matches for *Name*, *Relationship*, and *Death*. Although the names in the second record are really names of relatives of the first, they prevented our system from recombining these two records. Since it prevented the correct record from being returned and created two incorrect records, this problem affected both the precision and recall of our record separator.

Another problem arose in *complex18*. Since the document only contained four records, its magnitude measure was low enough that it appeared to be a single-record document. Our record separator did not attempt to split it so we lost three records.

	<i>records</i>	<i>returned</i>	<i>missed</i>	<i>extra</i>	<i>correct</i>	<i>precision</i>	<i>recall</i>
complex1	10	10	0	0	10	100.00%	100.00%
complex2	15	15	0	0	15	100.00%	100.00%
complex3	12	12	0	0	12	100.00%	100.00%
complex4	7	9	1	3	6	66.67%	85.71%
complex5	16	15	1	0	15	100.00%	93.75%
complex6	15	16	2	3	13	81.25%	86.67%
complex7	13	12	1	0	12	100.00%	92.31%
complex8	10	10	0	0	10	100.00%	100.00%
complex9	19	20	1	2	18	90.00%	94.74%
complex10	10	10	1	1	9	90.00%	90.00%
complex11	15	11	4	0	11	100.00%	73.33%
complex12	15	15	0	0	15	100.00%	100.00%
complex13	11	11	0	0	11	100.00%	100.00%
complex14	16	18	1	3	15	83.33%	93.75%
complex15	8	8	2	2	6	75.00%	75.00%
complex16	8	9	0	1	8	88.89%	100.00%
complex17	10	11	0	0	11	100.00%	110.00%
complex18	4	1	3	0	1	100.00%	25.00%
complex19	8	11	0	3	8	72.73%	100.00%
complex20	16	13	4	1	12	92.31%	75.00%
Total	238	237	21	19	218	91.98%	91.60%

Table 6: Complex Multiple-record Document Results

BROWN, Edwin, Born 18 Apr 1899 in Somerset, Kentucky, Married 1928
Ruth V. Rosenberg dau. of Johan N. and Anna Marie Eriksson Rosenberg, he died 4 Aug 1960 in
Toledo, Ohio at age 61

Figure 14: Records That Should Be Recombined

6.3 Summary of Results

Our record separator achieved an overall precision of 93% and a recall of 92%. This represents three improvements over the old record separator. First, the new record separator can recognize single-record documents. The old record separator always split them, but our new record separator only split about 10% in our test cases. Second, the new record separator improves the 51% precision and 67% recall of the old record separator respectively to 95% precision and 93% recall on simple multiple-record documents. Third, the new record separator can separate records in complex multiple-record documents with 92% precision and 92% recall. The old record separator always failed to separate records in complex multiple-record documents.

We encountered two problems that we believe require techniques more sophisticated than VSM to overcome. Magnitude measures alone are insufficient to correctly distinguish data-rich single-record documents and multiple-record documents with few records. In addition, unless we can resolve conflicts in matches before record separation, we need a more accurate test of whether two records can be merged.

Chapter 7

Conclusion and Future Work

This thesis documents two contributions to the automatic extraction of genealogical data from the Web. First, we implemented a general framework for ontology-based information extraction. Because the framework is based on ontological modeling, our system is able to handle Web pages taken from any source and is resilient to changes in those pages. Second, we developed a VSM-based record separator that achieved 93% precision and 92% recall in our experiments. In addition to its contributions to data extraction and record separation, this research also provides the beginning of what could be an important tool for genealogical research on the Web.

7.1 Future Work

Our extraction framework has room for improvement in several places. (1) Because improved heuristics are currently being developed in a project outside of this thesis, we did not test the overall extraction accuracy. Once these are in place, we will be able to evaluate and improve the overall accuracy of the system. (2) Development of an intelligent URL selector and parallelization will allow the system to scale to more sources and queries. (3) Currently, GeneTIQS is only capable of extracting data from static web pages. The addition of extraction modules using both existing and novel techniques will open up more sources of information to the system. (4) Sometimes not all of the information about a person is located on one page. People browsing for information follow links to other pages with more information when they find a record

interesting. Sometimes web pages also include links to other sources of information. A method of intelligently following links to either more information on a person or more sources of genealogical information would improve amount of information returned.

Our experiments have shown VSM to be a useful tool for record separation. In our experiments, we identified four points of improvement. (1) We used averages from the participation constraints to define the OV and to normalize the SVs. Thus, participation constraints were the basis for weights on each dimension in vector space. Although inferring these weights in this way proved effective, a more sophisticated method might prove more accurate. Allowing ontology designers to specify manually the importance of each object set would give them control over these weights. Alternatively, a machine-learning algorithm could automatically create these weights from labeled examples. (2) The conflict between person names and relative names caused many record separation errors. We have heuristics that resolve these conflicts, but they presuppose record separation. Heuristics able to disambiguate these matches before record separation would remedy this problem. (3) Most web pages employ some regular structure to present data. Since our basic algorithm evaluates each subtree on its own merits, it does not use this to its advantage. The refinement we added to detect simple patterns (Section 5.5.3) did so only in a small portion of pages. Many other methods can use patterns to recognize more data on a page. (4) The VSM scores we used were unable to differentiate completely between single-record and multiple-record documents. Other techniques are needed.

Bibliography

- [Anc04] Ancestry.com—Genealogy and Family History Records.
<http://www.ancestry.com>, March 2004
- [BLP01] D. Buttler, L. Liu, C. Pu. A Fully Automated Object Extraction System for the World Wide Web. In *Proceedings of the 2001 International Conference on Distributed Computing Systems (ICDCS'01)*, pages 361-370, Phoenix, Arizona, May 2001.
- [Chen02] X. Chen. *Query Rewriting for Extracting Data Behind HTML Form*, Masters Thesis, Brigham Young University, Provo, Utah, March 2004.
- [Cyn04] C. Howells. Cyndi's List of Genealogy Sites on the Internet.
<http://www.cyndislist.com>, March 2004
- [dmoz04] ODP—Open Directory Project. <http://www.dmoz.org/>, March 2004
- [dRC+98] M. De Rosa, T. Catarci, L. Iocchi, D. Nardi, G. Santucci. Materializing the Web. In *Proceedings of the Third IFCIS International Conference on Cooperative Information Systems (CoopIS '98)*, pages 24-31, New York, August 1998.
- [ECJ+99] D. Embley, E. Campbell, Y. Jiang, S. Liddle, D. Lonsdale, Y. Ng, R.D. Smith. Conceptual-Model-Based Data Extraction from Multiple-Record Web Documents. *Data and Knowledge Engineering*, 31(3): 227-251, November 1999.
- [EJN99] D. Embley, Y. Jiang, Y. Ng. Record-Boundary Discovery in Web Documents. In *Proceedings of the 1999 ACM SIGMOD International*

Conference on Management of Data, pages 467-478, Philadelphia, Pennsylvania, June 1999.

- [ETL02] D. Embley, C. Tao, S. Liddle. Automatically Extracting Ontologically Specified Data from HTML Tables with Unknown Structure, In *Proceedings of the 21st International Conference on Conceptual Modeling (ER2002)*, pages 322-337, Tampere, Finland, October 2002.
- [EM01] D. Embley, W. Mok. Developing XML with Guaranteed ‘Good’ Properties. In *Proceedings of the 20th International Conference on Conceptual Modeling (ER2001)*, pages 426-441, Yokohama, Japan, November 2001.
- [EM03] D. Embley, W. Mok. On Guaranteeing ‘Good’ Properties for XML. In *Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2003)*, pages 195-199, Orlando, Florida, July 2003.
- [EX00] D. Embley, L. Xu. Record Location and Reconfiguration in Unstructured Multiple-Record Web Documents, In *Proceedings of the Fifth International Workshop on the Web and Databases (WebDB 2000)*, pages 123-128, Dallas, Texas, May 2000.
- [EX01] D. Embley, L. Xu. Locating and Reconfiguring Records in Unstructured Multiple-Record Web Documents, In *Lecture Notes in Computer Science*, volume 1997, pages 256-276, Springer-Verlag, Heidelberg, 2001.
- [Gen04] Genealogy.com—Family Tree Maker Family History Software and Historical Records. <http://www.genealogy.com>, March 2004
- [Goo04] Google. <http://www.google.com>, March 2004

- [HGN+97] J. Hammer, H Garcia-Milina, S. Nestorov, M. Breunig, V. Vassalos. Template-Based Wrappers in the TSIMMIS System. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, Pages 532-535, Tucson, Arizona, May 1997.
- [KMA+98] C. Knoblock, S. Minton, J. Ambite, P. Ashish, I. Muslea, A. Philpot, S. Tejada. Modeling Web Sources for Information Integration. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI)*, pages 211-218, Madison, Wisconsin, July 1998.
- [KT02] S. Kuhlins, R. Tredwell. Toolkits for Generating Wrappers—A Survey of Software Toolkits for Automated Data Extraction from Websites. In M. Aksit, M. Mezini, R. Unland (Eds.): *Objects, Components, Architectures, Services, and Applications for a Networked World*, International Conference NetObjectDays (NODE 2002), pages 184-198, Erfurt, Germany, October 2002.
- [LRS+02] A. Laender, B. Ribeiro-Neto, A. Silva, J. Teixeira. A Brief Survey of Web Data Extraction Tools. *SIGMOD Record*, 31(2):84-93, June 2002.
- [LRS02] A. Laender, B. Ribeiro-Neto, A. Silva. DEByE—Data Extraction by Example. *Data and Knowledge Engineering*, 40(2):121-154, February 2002.
- [Roo04] RootsWeb.com Home Page. <http://www.rootsweb.com>, March 2004
- [RS97] M. Roth, P. Schwarz. Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources. In *Proceedings of International Conference on Very Large Data Bases (VLDB '97)*, pages 266-275, Athens, Greece, August

1997.

- [SA99] A. Sahuguet, F. Azavant. Web Ecology: Recycling HTML Pages as XML Documents Using W4F. In *Proceedings of the ACM International Workshop on the Web and Databases (WebDB'99)*, pages 31-36, Philadelphia, Pennsylvania, June 1999.
- [SMN01] H. Snoussi, L. Magnin, J.Y. Nie. Heterogeneous Web Data Extraction Using Ontology. In *Proceedings of the 5th International Conference on Autonomous Agents*, pages 99-110, Montreal, Canada, May 2001.
- [Tao03] C. Tao. *Schema Matching and Data Extraction over HTML Tables*, Masters Thesis, Brigham Young University, Provo, Utah, September 2003.
- [Wes03] A. Wessman. *A Framework for Extraction Plans and Heuristics in an Ontology-Based Data-Extraction System*, Thesis Proposal, Brigham Young University, Provo, Utah, March 2003.
- [Yah04] Yahoo! <http://www.yahoo.com/>, March 2004
- [Xu03] L. Xu. *Source Discovery and Schema Mapping for Data Integration*, PhD Dissertation, Brigham Young University, Provo, Utah, July 2003.

Appendix A

Test Documents

Cached copies of the documents we used to test our system are available at <http://www.deg.byu.edu/GeneTIQS/test/>. Table 7, Table 8, and Table 9 show the sources for these documents.

single1.html	http://www.cs.utk.edu/~dwalker/genealogy/PEOPLE/268.html
single2.html	http://www.cs.utk.edu/~dwalker/genealogy/PEOPLE/133.html
single3.html	http://www.cs.utk.edu/~dwalker/genealogy/PEOPLE/659.html
single4.html	http://www.mullgenealogy.co.uk/MullSearch.ASP?indv_no=9655
single5.html	http://www.mullgenealogy.co.uk/MullSearch.Asp?indv_no=4654
single6.html	http://www.mullgenealogy.co.uk/MullSearch.ASP?indv_no=7495
single7.html	http://www.uk-genealogy.org.uk/cgi-bin/genealogy/person.pl?496
single8.html	http://www.uk-genealogy.org.uk/cgi-bin/genealogy/person.pl?35
single9.html	http://www.uk-genealogy.org.uk/cgi-bin/genealogy/person.pl?59
single10.html	http://members.cts.com/crash/h/hindskw/KennethHinds/24755.html
single11.html	http://members.cts.com/crash/h/hindskw/KennethHinds/22838.html
single12.html	http://members.cts.com/crash/h/hindskw/KennethHinds/23647.html
single13.html	http://www.bdragon.com/cgi-bin/genealogy/tree.cgi/I0956.html
single14.html	http://www.bdragon.com/cgi-bin/genealogy/tree.cgi/I0947.html
single15.html	http://www.bdragon.com/cgi-bin/genealogy/tree.cgi/I0901.html
single16.html	http://users.erols.com/ulrich/jbt0071.html
single17.html	http://users.erols.com/ulrich/glc0074.html
single18.html	http://users.erols.com/ulrich/jeu0027.html
single19.html	http://www.geocities.com/Heartland/Meadows/2303/susannah.html
single20.html	http://www.geocities.com/Heartland/Meadows/2303/henry.html
single21.html	http://www.geocities.com/Heartland/Meadows/2303/elvira.html

Table 7: Sources of Single-record Test Documents

simple1.html	http://blairgenealogy.com/vermont/descendants_of_john_blair.html
simple2.html	http://homepages.rootsweb.com/~edburton/names3.htm
simple3.html	http://www.geocities.com/Heartland/Bluffs/6882/
simple4.html	http://www.clayton-clan.org/gedhtml/tmc/n_106.html
simple5.html	http://www.geocities.com/Vienna/5134/thomaswalkerrob.html
simple6.html	http://www.geocities.com/white_dove41/Dillon
simple7.html	http://64.235.34.221/rosehill/genthompson.htm
simple8.html	http://perrussel.netfirms.com/thompson/
simple9.html	http://users.adelphia.net/~buddy75/souther_details.html
simple10.html	http://andrsn.stanford.edu/ind0009.html

Table 8: Sources of Simple Multiple-record Test Documents

complex1.html	http://www.familyworkings.com/gedcoms/hays/dat9.htm
complex2.html	http://www.geocities.com/Heartland/Estates/8053/pedigre1.html
complex3.html	http://www.geocities.com/Heartland/Garden/7021/millardmiller/dat132.html
complex4.html	http://www.geocities.com/Heartland/Meadows/7939/edwinbro.htm
complex5.html	http://home.comcast.net/~sfburton/FamilyTree/pafg14.htm
complex6.html	http://www.angelfire.com/nd/domneal/mcmillan.html
complex7.html	http://www.hoopesonline.com/hoopes_family_genealogy/family00400.html
complex8.html	http://www.uttometer.demon.co.uk/html_genealogy/desc_0.htm
complex9.html	http://ramsey.users5.50megs.com/DESCENDA/desc07.htm
complex10.html	http://home.tampabay.rr.com/drewsmith/d0001/g0000077.htm
complex11.html	http://mcguinnessfamily.org/ancestry/p4.htm
complex12.html	http://www.angelfire.com/nc/JohnMiddleton/Pedijrm1.html
complex13.html	http://www.angelfire.com/mo/winkeler/scott.html
complex14.html	http://home.earthlink.net/~larsrbl/Gen/genealogypage.html
complex15.html	http://homepage.powerup.com.au/~ajthomps/JWT_History.htm
complex16.html	http://www.geocities.com/Heartland/Plains/4897/thomson.htm
complex17.html	http://www.familyworkings.com/gedcoms/ripley/dat22.htm
complex18.html	http://www.papachuck.org/gene/john/d4104.htm
complex19.html	http://home.cc.umanitoba.ca/~wyatt/harrison-merrickville.html
complex20.html	http://members.aol.com/ArletaHowe/Anderson2.html

Table 9: Sources of Complex Multiple-record Test Document