School of Electronics and Computer Science

Faculty of Physical and Applied Sciences

University of Southampton

Alex Parker

April 15, 2012

WikiTime

A system that extracts temporal events from Wikipedia

Supervisor: Dr Adam Prugel-Bennett

Second Examiner: Professor R.I. Damper

A project report submitted for the award of

Bsc Computer Science

Abstract

The internet is a large source of information but there is no easy way to view historical events mentioned within each page in a structured way, such as on a Timeline. This project plans to extract and semantically link historical events and dates, such as the births and deaths of famous figures, and enter them into a search engine so that they can be searched and viewed on a Timeline.

This project presents a working system that extracts historical events from the full English Wikipedia and displays them on a Timeline, and evaluates the applications and effectiveness of this system in an academic and educational context.

Contents

1 Introduction

The internet taken as a whole is a massive source of information of a scale never seen before, leading to an information overload where the background noise of low quality information can block out the useful information [5].

Simple filtering and categorisation of this information is a problem which has been solved by modern web search engines through the use of the Page Rank[1] and Map Reduce[2] algorithms. However the user must still search the pages linked by the search engine for the answer to their query, again leading to information overload [5]. This is a problem that can be reduced by specialising the search interface to the user's query.

A lot of information available online is related to historical events, and the traditional web search interface does not provide access to this information easily. For example, the Battle of Hastings page on Wikipedia contains the following sentence *"The Battle of Hastings occurred on 14 October 1066 during the Norman conquest of England"*[44], but for the user to discover this fact they must open and read the page after searching for "Battle of Hastings" in a web search engine. The problem is caused by the search engine interface not being personalised to the user's needs [20].

This project presents a system that extracts dates and times mentioned within web pages within a searchable interface, effectively summarizing events in history automatically, providing the user with an overview of the topic they are searching for.

Wikipedia is used as the primary source of information for this project as it is a largely well written source of information for historical events, and it is assumed that for the purpose of this work Wikipedia is reasonably accurate. A number of academic studies have used Wikipedia as their primary data source for research, and regard it as a source worthy of research and publication in academic journals [32].

1.1 Project Goals

The goals of this project are to demonstrate a system that can parse and extract historical events from Wikipedia and render them on a searchable Timeline interface. This problem can be split into two parts; the extraction and search indexing of temporal events from Wikipedia and the display of this information in a searchable interface to generate a Timeline. The search interface should allow for queries to be run within a specified date range and with additional filters such as the page the event was extracted from.

[1]An algorithm that models a random web surfer that either clicks a link or jumps to another page and determines page ranking by the probability of a surfer ending up on that page.

[2]A distributed computing algorithm that allows a problem to scaled to clusters of machines easily.

1.2 Scope

This project will focus on articles contained within Wikipedia as it is a large source of mostly accurate and well written information [32]. Wikipedia was also selected as a source since snapshots of the entire encyclopaedia are freely available permitting the project to focus on the event extraction and display. A web crawler[3] will not be used since Wikipedia limits web crawler requests to request 1 per second [46].

1.2.1 Temporal Event Extraction

The extraction of events from Wikipedia presents many different problems. The first problem is related to the scale of Wikipedia itself. Wikipedia as of December 2011 contains 3,817,405 articles in English [45], so the system must be capable of scaling to this amount of information. To resolve this problem care will be taken to ensure the code can be run on multiple threads and possibly distributed over a cluster of machines.

Another problem is with the definition of a temporal event. Dates and times in English sentences can be relative to events mentioned earlier, and can be mentioned in many different formats with inconsistencies between them. For example the date 08/12/11 could be the 8th of December in the UK or the 12th of August in the US. The system will not attempt to process relative times such as *"last Wednesday"* as this may not be useful in a historical context. Additionally dates that are defined in a shorthand form such as *"08/12/11"* as there is too much variation in formats to assume that the correct date can be extracted. Finally to simplify the extraction the system will ignore any time references such as *"8 pm"* or *"18:30"*, again because there are many ways to represent time in English.

A further problem is related to the categorisation and extraction of the context surrounding the temporal event, i.e. what does the event actually mean? This project will attempt to extract the sentence that contains the date reference as well as the page title it came from. Additionally named entities, such as a famous person or place mentioned in the page will be extracted and stored as part of the event to attempt to provide context to the event.

1.2.2 Indexing and Display

Once all of the events have been extracted from Wikipedia they need to be made searchable by keyword and date range. The system will use a software library to do this, such as Lucene[4]. To help with the prioritisation of pages a Page Rank algorithm will be used by measuring the importance of the current page being processed.

[3]A program that browses the world wide web to gather information

[4]A high performance text engine search library

Since multiple pages will be processed by this system it is possible for events to be repeated multiple times, especially for important events in history. The system will attempt to remove these duplicates and incorporate them into the event ranking, since multiple events are likely to be more important than events mentioned once.

The final problem is the display of the Timeline results to the user, as it needs to show the ranking of the events and their position on the Timeline relative to each other.

2 Background

Since the scope of this project covers many different topics a wide range of background reading was required, including the documentation of the various software tools that were used.

Daniel Rosenburg in the book “Cartographies of Time” found that “the timeline is one of the central organizing features of the contemporary user interface”[37] and that the sheer volume of information available on the world wide web makes indexing systems highly valuable. Additionally he feels that the scope of these “Web 2.0” timelines are far larger and more extensive than those seen before, but that it remains to be seen whether these new systems can provide an adequate historical viewpoint compared with timelines found in museums.

2.1 Related Work

DBpedia is a research project that aims to produce a “semantic web mirror” of Wikipedia by converting structured information extracted from Wikipedia into RDF[5], making it freely available on the internet [4]. RDF is the data format used to link structured information available from different sources together into a “global data space”[9]. The DBPedia project extracts the information by examining the info boxes that are contained on most Wikipedia articles which describe people, places, music albums, films, video games, organisations and diseases in a structured way.

In the “Extraction of Temporal Facts and Events from Wikipedia” Master Thesis [27] Erdal Kuzey examines the extraction of historical facts from Wikipedia into an RDF ontology, specifying an algorithm that examines the info boxes, similarly to the dbpedia project [4], but with a heavier focus on the dates involved. The thesis also examines extraction from free text within the the article paragraphs, first by attempting to normalize dates within the text to a standard format, and then extracting temporal facts matching these with well known base facts previously extracted from the info boxes. The limitations of this approach are that it is very intensive, and requires consideration

[5]Resource Description Format

of every type of info box that is available on Wikipedia, so a parser must be written to process each info box type, with similar issues found in the free text extraction.

The WikiTime project aims to take a different approach to this information extraction problem by focusing on the paragraphs in the articles themselves, meaning that concepts and information not provided in an info box will be extracted, at the cost of lower accuracy.

2.2 Event Extraction

A lot of research has focused on the extraction of temporal information from business news, pulling in concepts from the fields of Information Extraction[6] and Computational Linguistics[7]. Much of the research focused on the extraction of relative time references such as "yesterday" or "last week"; the Time-Indexer tool defined a set of rules that would map these references to a standardised time format in terms of hours, days, weeks, months, years, etc. The rules were implemented as a finite state machine that would read the input text [24].

Other research dealt with the fact that the majority of temporal expressions are underspecified or fuzzy i.e. missing the time or month that the event occurred [10], or that the event occurred around a non-specific time such as "last week" [25]. For this project relative temporal expressions such as "last month" or "tomorrow") will be ignored since these terms do not generally appear in Wikipedia, however an attempt to deal with absolute events that are underspecified and fuzzy will be made.

For the extracted events to make sense they must be placed in context. The project will achieve this by extracting the sentence that contains the time reference. Sentence Extraction is a well known problem in Computational Linguistics which is commonly solved through the use of lists of hand-authored regular expressions and common abbreviations, which become specialised to the text area they are extracting [1]. An alternative method is to produce a statistical model of sentence boundaries from a corpus text [18].

To provide additional context an attempt to extract named entities will be made, where a named entity is defined as a person, place or company. This is another well known problem in Computational Linguistics which is approached similarly to the sentence boundary detection problem. Researchers approach this problem in three ways; hand crafted rules, machine learning and hybrid detection schemes [40].

Many software libraries exist to support these tasks in Computational Linguistics in multiple programming languages. The perl *Lingua::EN::NamedEntity* module used in the prototype uses rule-based techniques to extract and classify entities from free text [38]. The Apache OpenNLP [15] and the Stanford NER [23] libraries written in Java use

[6]The extraction of structured information from unstructured documents

[7]The statistical or rule-based modelling and processing of natural language

trained statistical models to extract sentences, tokenize sentences into words and extract named entities.

Important events extracted from Wikipedia are duplicated multiple times, creating a lot of noise, therefore the duplicated events must be removed. There is a lot of prior work in this area detecting the similarity between sentences for academic plagiarism and duplicate removal in search engines. Since many of the algorithms used for near-duplicate detection do not work well on short texts, as it is more difficult to extract effective features of that text [42]. Qi Zhang et. al. outline a method to apply partial-duplicate detection to the map reduce algorithm to permit the processing of large amounts of text, which will be required since Wikipedia contains millions of pages so will extract millions of events [48].

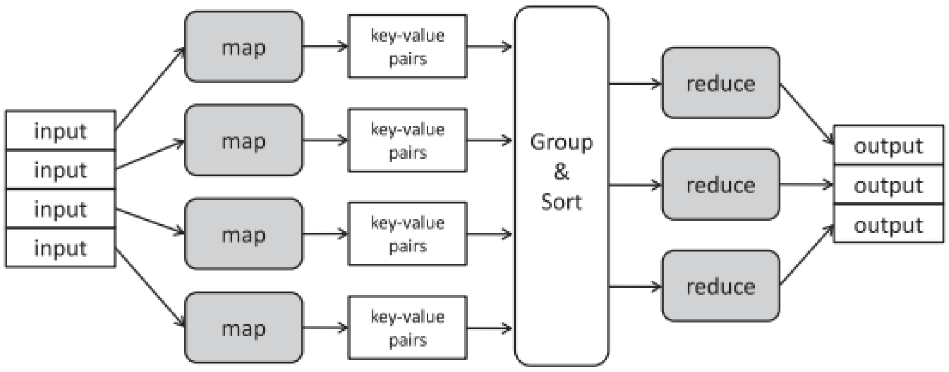


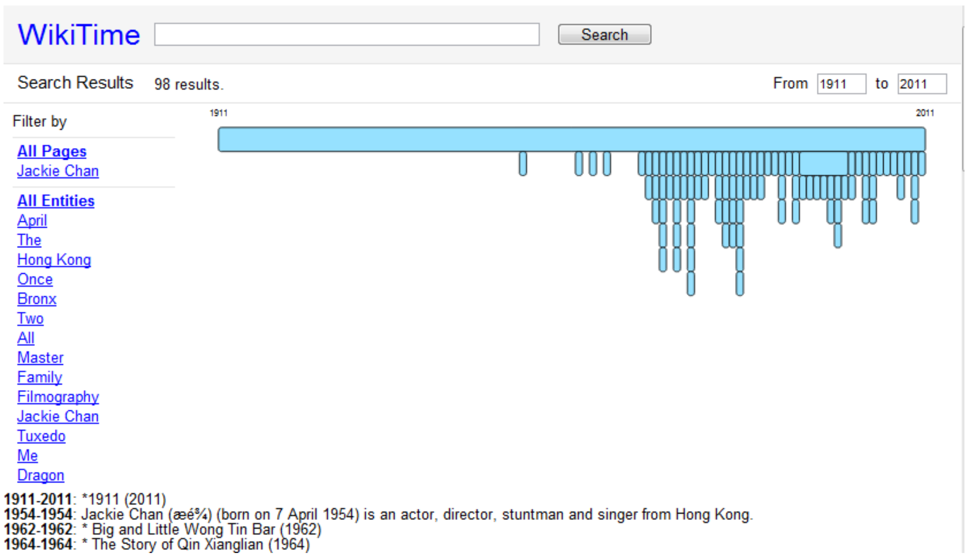
Figure 1: Map Reduce algorithm (Sourced from [48])

The map reduce algorithm allows data intensive tasks to be spread over clusters of machines easily. As shown in Figure 1 the input is split into multiple map processes which output key-value pairs. Each key-value pair is then grouped and sorted as input into multiple reduce processes, which provide the final output [43].

3 Prototype

To better understand the issues surrounding the project goals a prototype has been created that processes the Simple English Wikipedia. The Simple English Wikipedia was selected because it was smaller allowing for fast iteration on the indexing routines. Figure 2 shows the search interface that was developed displaying the events extracted from the “Jackie Chan” biography page on the Simple English Wikipedia.

The system is split into 2 parts; the extraction and indexing of events and the presentation of those events in a website. Perl was selected for the prototype because it has good support for regular expressions as well as many useful modules to solve problems such as sentence parsing, named entity extraction, search indexing and database



Search Results 98 results. From 1911 to 2011

Filter by

All Pages
Jackie Chan

All Entities
April
The
Hong Kong
Once
Bronx
Two
All
Master
Family
Filmography
Jackie Chan
Tuxedo
Me
Dragon

1911-2011: *1911 (2011)
1954-1954: Jackie Chan (æé¾) (born on 7 April 1954) is an actor, director, stuntman and singer from Hong Kong.
1962-1962: * Big and Little Wong Tin Bar (1962)
1964-1964: * The Story of Qin Xianglian (1964)

Figure 2: Prototype Timeline output for Jackie Chan's biography page.

connectivity.

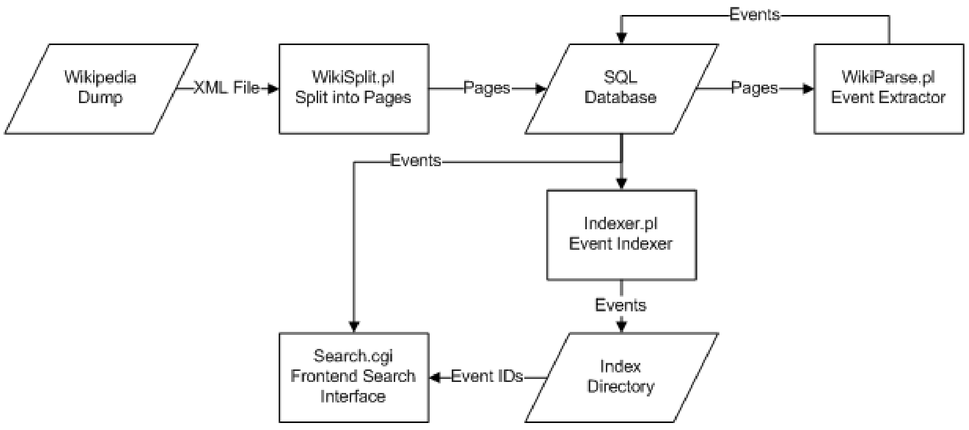


Figure 3: Prototype System Design

Figure 3 shows how the large XML snapshot of Wikipedia is processed by the system. First in the splitting script the XML file is read and each article is inserted as rows into the database. Next in the parsing script the article is converted from Wikipedia syntax into plain text and split into sentences with the perl *Lingua::EN::Sentence* module [47].

Each sentence is then inserted into the database if it contains a date reference which is implemented as a regular expression search. If a sentence contains multiple date references then only the earliest date and the most recent date are recorded. Once an event has been identified named entities are extracted from the sentence with the perl

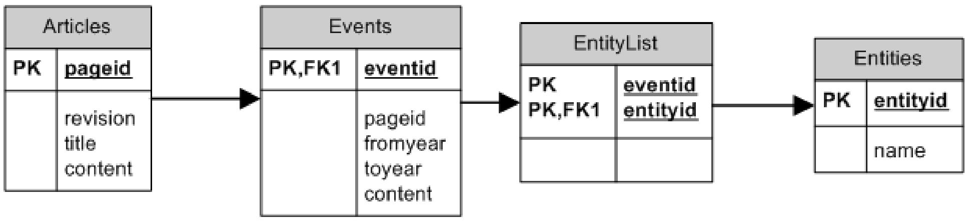


Figure 4: Prototype Database Design

Lingua::EN::NamedEntity module [38] which are inserted as rows into the EntityList and Entities table shown in Figure 4.

Once all of the events have been extracted the Event Indexer builds a Lucene index making the events searchable by the front end. The perl *Lucy* module was used to achieve this [14]. Finally the events are displayed on a Timeline on the website frontend making use of the database and Lucene index.

3.1 Map Reduce Prototype

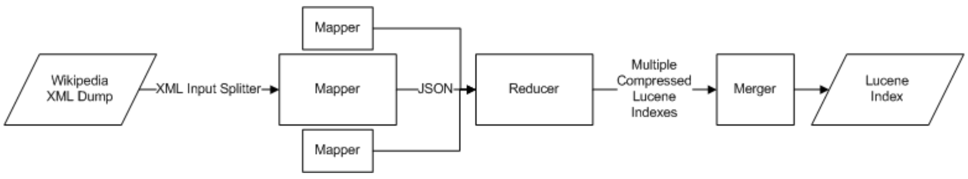


Figure 5: Map Reduce Prototype Pipeline

Additionally a prototype making use of the Map Reduce algorithm was created. This system uses the Apache Hadoop software libraryhadoop to process Wikipedia XML dumps into a Lucene index in a single map-reduce operation. It works by using an XML Input Splitter to get each Wikipedia article as input into the map operation. The mapper then outputs a JSON[8] string representing the Wikipedia page.

Each page is then input to the reduce operator which runs a perl script which parses the article into plain text, splits it into sentences and then extracts events from those sentences. Each sentence is then inserted into a single Lucene index. The reduce operation then compresses this index and copies it to the output directory. Once the map reduce operation is complete a perl script extracts the multiple compressed Lucene indexes and merges them into a single Lucene index.

The front end used in the prototype was also updated to exclusively use the Lucene index to get the data, as this was found to be more scalable than inserting the events into a separate SQL database.

[8]JavaScript Object Notation

4 Final System

The final system processes the full 20GB English Wikipedia XML Dump into a Lucene index which can be searched via a website.

Java was selected for the final system because dependent libraries can be packaged in a jar file making it more portable than Perl, where dependencies must be installed via a package manager such as *CPAN*. Additionally Java has high quality IDEs[9] such as *Eclipse* which allows for easier debugging and unit testing. Finally Java is faster than Perl in most situations [17].

4.1 Event Extraction Process

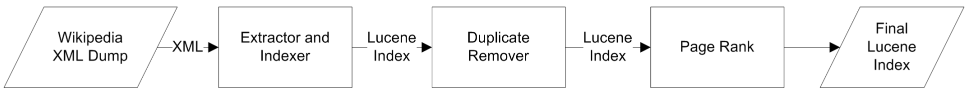


Figure 6: Event Extraction Process Overview

The event extraction process is outlined in figure 6, showing the processing stages taken to extract the events from a Wikipedia XML dump. First the XML file is processed into events and indexed into a Lucene index. This Lucene index is then processed to reduce duplicate events into a single event which is ranked higher in the search results. The de-duplicated index is then ranked so that more important Wikipedia pages are higher in the search results, where important Wikipedia pages are found using the *"Page Rank"* algorithm [7].

Due to the requirement to process large amounts of information quickly a generic framework was written to allow processing to run on multiple threads. Figure 7 shows the *Reader<T>*, *Writer<T>* and *Processor* classes which are extended to implement a flexible processing pipeline. The *Processor.Run()* method reads objects of type *T* from the *Reader<T>* class and inserts them into a queue. These items are then removed from the queue on multiple threads and processed by the *Writer<T>* class.

4.1.1 Extractor and Indexer

As shown in Figure 8 the Extractor and Indexer stage takes the Wikipedia XML Dump and splits it using the *WikipediaReader* class shown in Figure 7. This outputs a String containing the content between the *<page>...</page>* XML elements. The *EventProcessor* class takes this String and parses it into multiple *Event* classes, parsing the Wikipedia syntax into articles with the *Article* class and extracting the events with the

[9]Integrated Development Environment

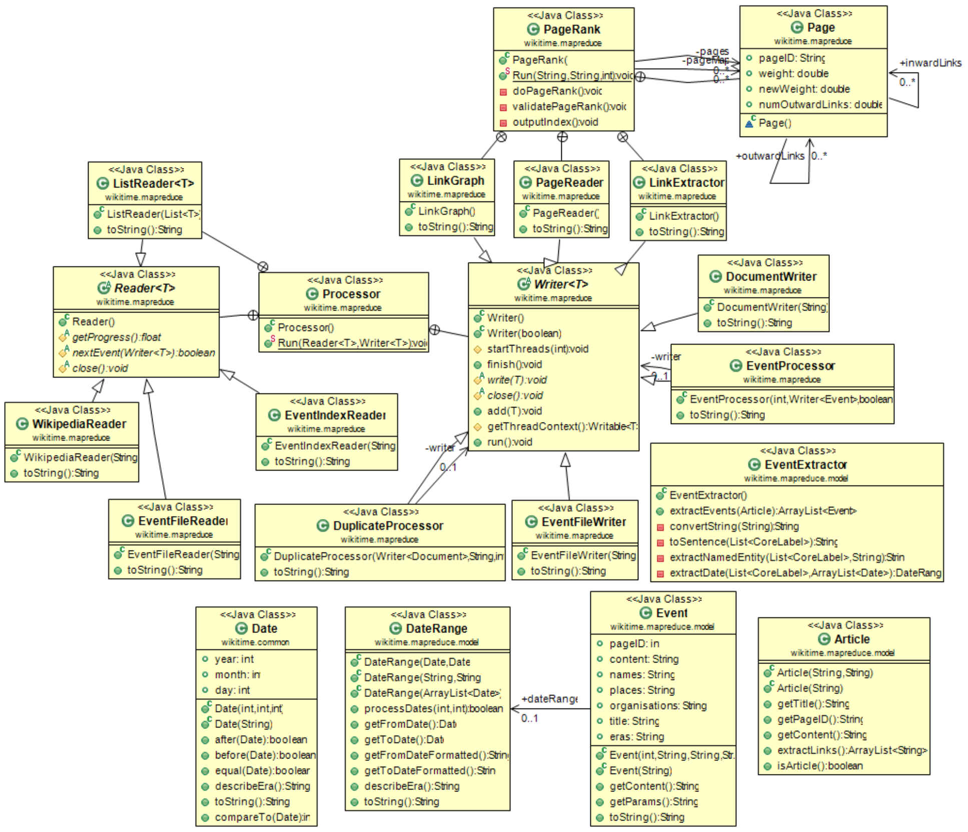


Figure 7: Event Extractor System Class Diagram

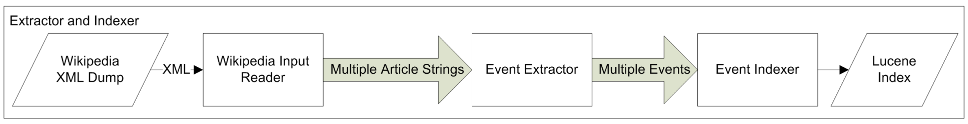


Figure 8: Event Extraction Process

EventExtractor class. Finally these events are indexed by the *EventIndexer* class into a Lucene index.

The *Article* class uses the Java Wikipedia API [22] which parses the Wikipedia markup language into a syntax tree. The custom *PlainTextConverter* class converts this syntax into plain text ignoring tables, images and bullet point lists. The Java Wikipedia API was selected because it gives better results than the Perl prototype which used regular expressions to evaluate the syntax.

The *EventExtractor* class is used to extract the article text into events using the *extractEvents()* function. The Event Extraction algorithm uses the Stanford Named Entity Recognizer [23] to split the text into sentences and tokens and find named entities. The Stanford NER is a statistical classifier which classifies text from training data and uses a three class model which finds Locations (e.g. *"London"*), People (e.g. *"Albert Einstein"*) and Organisations (e.g. *"University of Southampton"*). The classifier works by labelling each token with its classification (None, Location, People, Organisation) based upon nearby tokens.

At an earlier stage of the project the Apache OpenNLP Named Entity Recognizer [15] was used instead of the Stanford NER library. The Stanford NER was selected for the final system because it provides better extraction results and copes with multi-threaded API calls better. See section 5 for performance and extraction comparisons.

4.1.2 Date Extraction Algorithm

The date extraction algorithm shown in figure 9 is used by the *EventExtractor* class to find dates within a sentence, and operates on a single sentence which has been split into tokens. The algorithm works by remembering months (January, February, etc.) and dates (18th, 1st) seen so far, resetting them when text is seen but continuing if there is punctuation or other characters. If a year is detected then a new date is created with the stored month and day values, and the date is inverted if *"BC"* is found in the next token, permitting handling of BC dates.

This algorithm detects dates that are missing information, such as *"1983"* and *"January 1920"*, different orderings of information, such as *"March 18th 1530"* and *"18th March 1530"*, and dates that are separated by punctuation such as *"(May 17th, 1935)"*.

Since the algorithm iterates over the whole sentence multiple dates can be detected which can lead to problems if more than 2 dates are detected, as shown in this extract from

```
function EXTRACTDATE($sentence$)
    $dates \leftarrow \{\}$
    $year \leftarrow -1$
    $month \leftarrow -1$
    for $word \in sentence$ do
        if $word$ matches $/\wedge[0-9][0-9][0-9][0-9]?/$ then                ▷ Match a year
            $year \leftarrow word$
            if $year < 2050 \wedge year \geq 100$ then
                if $day = -1$ then
                    $day \leftarrow 1$
                end if
                if $month = -1$ then
                    $month \leftarrow 1$
                end if
                if $word \rightarrow next = "bc"$ then                        ▷ Handle BC Dates
                    $year \leftarrow -year$
                end if
                $dates \leftarrow dates \cup date(day, month, year)$
            end if
        else if $word$ matches $/\wedge[0-9][0-9]?/$ then                      ▷ Match a day
            $day \leftarrow word$
        else if $word$ matches $/(january|...|december)/$ then                 ▷ Match a month
            $month \leftarrow word$
        else if $word$ matches $/[A-Za-z]/$ then                                ▷ Match text
            $day \leftarrow -1$
            $month \leftarrow -1$
        end if
    end for
    return $dates$
end function
```

Figure 9: Date Extraction Algorithm

the *Abraham* Wikipedia Article:

> *"Many artists have been inspired by the life of Abraham: Albrecht Dürer (1471–1528), Caravaggio (1573–1610), Rembrandt van Rijn (Dutch, 1606–1669) created at least seven works on Abraham, Petrus-Paulus Rubens (1577–1640) did several, Donatello, Raphael, Philip van Dyck (Dutch painter, 1680–1753), Marc Chagall did at least five on Abraham, Gustave Doré (French illustrator, 1832–1883) did six, Claude Lorrain (French painter, 1600–1682), James Jacques Joseph Tissot (French painter and illustrator, 1836–1902) did over twenty works on the subject ."* [2]

The final system selects the earliest and latest dates in the sentence creating an event that spans from 1471 to 1902, which is difficult to display in the user interface.

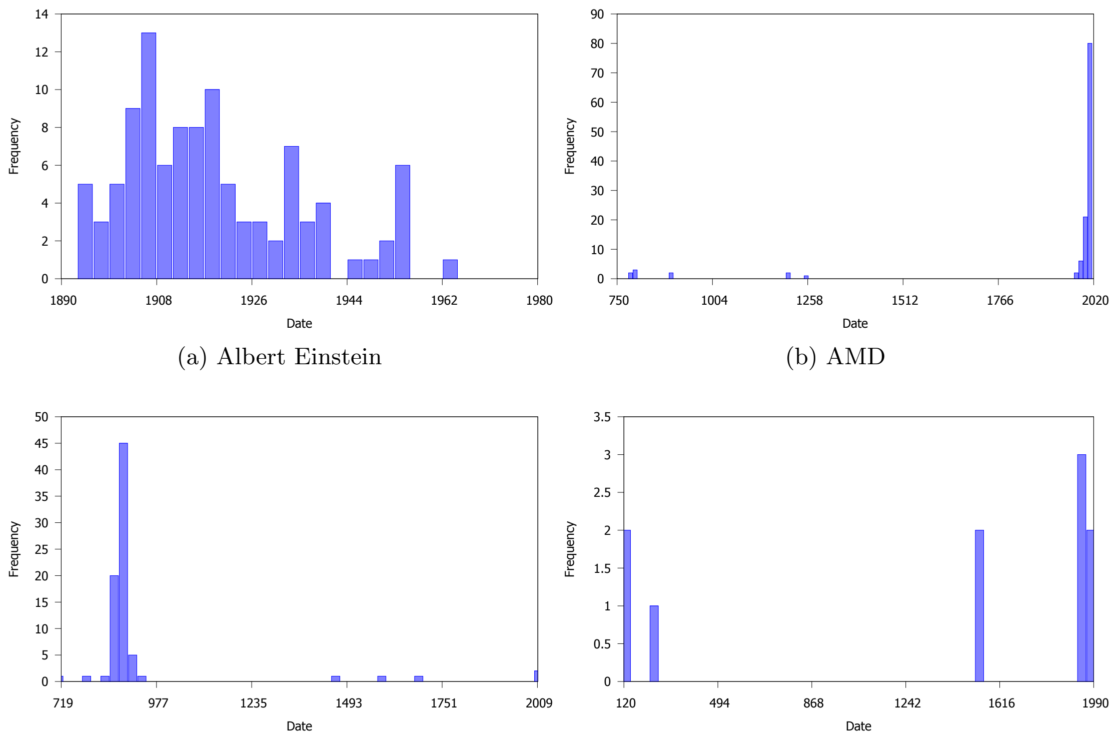
An additional problem is that years can get confused with amounts, for example a kilobyte has 1024 bytes but Pope Benedict VIII also died in 1024. Earlier prototypes attempted to resolve this problem by ignoring dates less than 1000 and more than 2050, but this still doesn't solve the 1024 problem. The final system attempts to solve this by calculating the median and average absolute deviation of all of the extracted dates in the current page, discarding events that are more than three absolute deviations away from the median.

Figure 10 shows the frequency of extracted events for different date ranges on a selection of Wikipedia articles. The AMD article has a lot of dates around 2000 which causes the numbers around 1250 and 750 to be discarded, which is correct. The King Alfred article has most of the events focused around 800 which causes the later events to be removed. None of the events are discarded in the Albert Einstein article as the total events are evenly spread out. The numbers around 1800 in the Byte article are not discarded as there are not enough correct events in 1990 to exclude them.

4.1.3 Indexing

The *EventIndexer* class indexes events into a Lucene index, using the Java Lucene Library [13]. Figure 11 shows which fields are inserted into the Lucene index.

Lucene uses text analyzers to process the text to allow for keyword searches. The *org.apache.lucene.analysis.en.EnglishAnalyzer* class is used for the content, name, place, org, eras and sig fields, whilst the *org.apache.lucene.analysis.KeywordAnalyzer* is used for the title field. The *EnglishAnalyzer* class splits the text into tokens with the *StandardTokenizer* class, converts them to lowercase with the *LowerCaseFilter* class, removes a set of common stop words with the *TokenFilter* class and finally runs the Porter Stemming Algorithm [34] which is implemented by the *SnowballFilter* class. The Porter Stemming Algorithm removes suffixes from a word returning the common stem, for example *CONNECT* is the stem word for:



(a) Albert Einstein (b) AMD

(c) King Alfred (d) Byte

Figure 10: Histograms showing extracted dates

CONNECT
CONNECTED
CONNECTING
CONNECTION
CONNECTIONS

The *KeywordAnalyzer* class returns the whole string of the document, meaning that a query must match the string precisely. This allows for direct searches by page title, making it possible to link from Wikipedia pages back into the WikiTime page.

The eras field is generated from the start and end date and allows for searches such as *"18th Century Painters"* and *"1990s Films"*. It generates three descriptions of the current date; the century (*"20th Century"*, *"1900s"*) and the decade (*"1960s"*).

4.1.4 Duplicate Remover

Once the Wikipedia XML Dump has been extracted into events and indexed into a Lucene index the next stage reduces duplicate events into a single event. The duplicate

Name	Description	Stored	Analyzed
content	Extracted sentence	Y	Y
pageid	Unique article number	Y	N
title	Article title	Y	Y
fromdate	Event start date in format *“YYYY,MM,DD”*	Y	N
todate	Event end date in format *“YYYY,MM,DD”*	Y	N
name	List of people mentioned in the sentence	Y	Y
place	List of locations mentioned in the sentence	Y	Y
org	List of organizations mentioned in the sentence	Y	Y
eras	List of era descriptions (e.g. *“19th Century, 1800s”*)	Y	Y
sig	Spot Signature used for duplicate removal	Y	Y

Figure 11: Indexed Fields

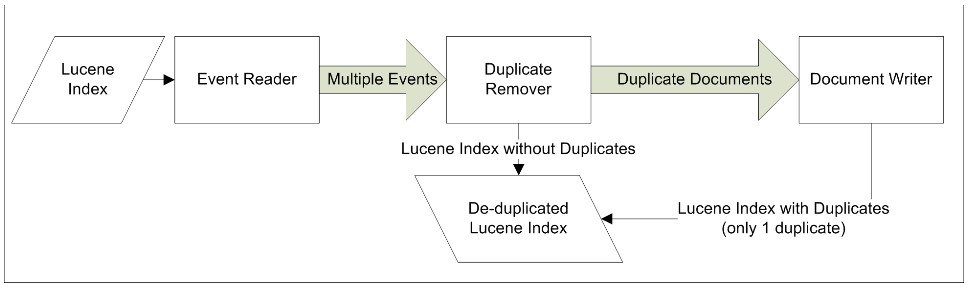

Lucene Index
Event Reader
Multiple Events
Duplicate Remover
Duplicate Documents
Document Writer
Lucene Index without Duplicates
De-duplicated Lucene Index
Lucene Index with Duplicates (only 1 duplicate)

Figure 12: Duplicate Removal Process

removal process is outlined in Figure 12 and starts by reading the events from the Lucene index using the *EventIndexReader* class shown in Figure 7 which outputs *Event* objects into the *DuplicateProcessor* class. The *DuplicateProcessor* class outputs a single event for each duplicate event which is indexed into an additional, temporary index called *index.dups* by the *DocumentWriter* class. Once indexing is complete the *index* and *index.dups* indexes are merged to form the final de-duplicated Lucene index.

The duplicate removal process uses an implementation of the *spot signatures* algorithm that detects near duplicates in large web collections [41]. It works by generating a *spot signature* of the text you wish to detect duplicates with by extracting words surrounding stopwords[10] in the text. Some examples are shown in Figure 13.

The *DuplicateProcessor* generates a *spot signature* for the current event, and then searches the index for events that have similar spot signatures with the same date range. This produces a list of duplicate results which are deleted if the event's weight[11] is ≥ 10 and then the first event is output to the *DocumentWriter* with its weighting set to the number of duplicates it replaced.

[10]Frequently occurring words in the english language, e.g. “and”, “the” and “this”
[11]The event's similarity score

“Maria Matilda Ogilvie Gordon (30 April 1864 – 24 June 1939) was a 19th century Scottish scientist.“

```
[maria:maria:matilda:ogilvie, was:19th:century:scottish]
```

“The Aragonese Crusade, or Crusade of Aragón, was declared by Pope Martin IV against the King of Aragón, Peter III the Great, in 1284 and 1285.”

```
[the:aragonese:crusade:or, of:aragn:declared:pope,
the:king:aragn:peter, the:great:1284:and]
```

“The Dutch post-Impressionist painter Vincent van Gogh lived in Arles in 1888-1889 and produced over 300 paintings and drawings during his time there.”

```
[the:dutch:postimpressionist:painter, in:arles:18881889:and]
```

“The first University of Southampton degrees were awarded on 4 July 1953, following the appointment of the Duke of Wellington as Chancellor of the University.”

```
[the:first:university:southampton, were:awarded:on:4,
the:appointment:duke:wellington]
```

Figure 13: Spot Signatures generated by the algorithm

4.1.5 Page Rank

Figure 14 shows the page ranking process, which is implemented by the *PageRank* class shown in Figure 7. First the Wikipedia XML Dump is read into the *PageReader* class which creates a hash table mapping page identifiers to *Page* objects. Next the Wikipedia XML Dump is read into the *LinkExtractor* class which extracts the outward links from each page using the hash map from the first step to build a list of pointers to the *Page* objects in the map. Each page object is also added to a list of all pages, and the hash table mapping is deleted.

Next the page list is read using the *ListReader<T>* class into the *LinkGraph* class which turns the outward links list into an inward links list, and counts the total number of outward links from each page. At this point the *PageRank.doPageRank()* function runs the Page Rank algorithm on the link graph. Finally, the *PageRank.outputIndex()* function searches through the existing *index* for each page and inserts each event into a new *index.ranked* folder, multiplying each event’s weight by the current page Page Ranking.

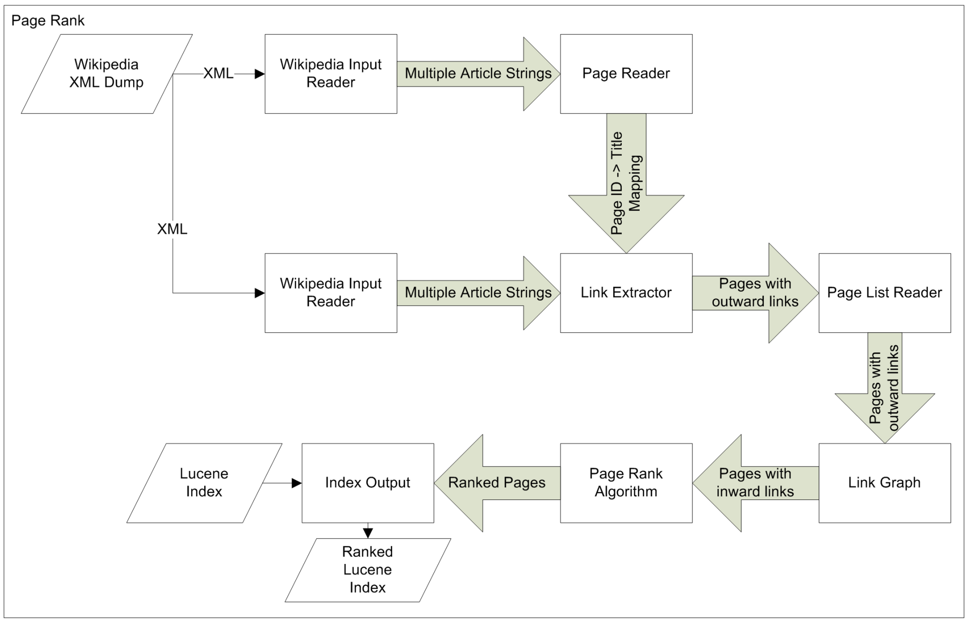


Figure 14: Page Rank Process

4.1.6 Page Rank Algorithm

The Page Ranking algorithm is used to rank important events higher in search results over less important events. A Page Rank is the probability that a random web server clicking on links ends up on a particular page, and the Page Ranking algorithm works by measuring the “link popularity” between pages where the pages with the most incoming links get a higher search ranking [7].

The algorithm starts by assigning an equal probability to each webpage. So the initial page rank for each page p_i for N pages when iteration $t = 0$ is:

$$PR(p_i, 0) = \frac{1}{N}$$

Next the algorithm iterates over t with the total number of links for the page defined as $L(p)$ and the links for the page defined as $M(p)$. A random surfer value d is added to allow the random web surfer to move to a random page, which is usually set to 0.85.

$$PR(p_i, t+1) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j, t)}{L(p_j)}$$

The algorithm iterates until for some small ϵ:

$$\sum_{\forall p} |PR(p_i, t+1) - PR(p_i, t)| < \epsilon$$

4.2 Website

The Website displays the events on an interactive timeline that the user can zoom through and navigate easily. The website server is written with the Java Restlet web framework [28] which enables implementation of a REST[12] API allowing the user interface to make AJAX calls improving the user experience and the use of "Cool URIs" [6] such as *"http://wikitime.ecs.soton.ac.uk/search/?newton"* providing flexibility for the future expansion of the project website.

Figure 15 shows the main classes used for the web server. The *WikiTimeSite* class starts up the internal web server and registers the REST search API and website pages. The *Page* class represents a single page on the website and loads the page contents into memory on startup to allow for fast response times.

The *SearchResource* class implements the search API ("*/api/search/newton*") and formats and outputs events returned from the *Searcher* class in JSON format. JSON was selected as the search API output format because it can be easily parsed in JavaScript via the *"eval()"* method. The *Searcher* class opens the Lucene text index and builds suitable queries using the Lucene *MultiFieldQueryParser* class.

4.2.1 User Interface

The user interface consists of two pages; the home page and the search results page which are written in HTML, CSS and Javascript. For the search page the JQuery library[36] is used to make the AJAX calls to the REST server API which is combined with the JQuery history plugin [29] to provide browser navigation. The Simile Widgets Timeline library [21] is used to display the events on a Timeline and the JQuery UI library [35] implements the scroll bar at the bottom of the screen. Finally the labels javascript library [3] displays the *"Search"* placeholder text in the search fields found on the home page and search results page and the CSS reset tool [30] is used to unify the CSS layout between browsers.

The Timeline layout provided by the Simile Timeline Widget didn't display the ordering of events by priority meaning that less important events would end up at the top of the results page instead of the bottom and vice versa. This was resolved by writing a custom layout algorithm which positions the most important events first, pushing subsequent events down only if they overlap with an existing event. The overlap detection works by storing the minimum and maximum date found on each track.

[12]Representational State Transfer

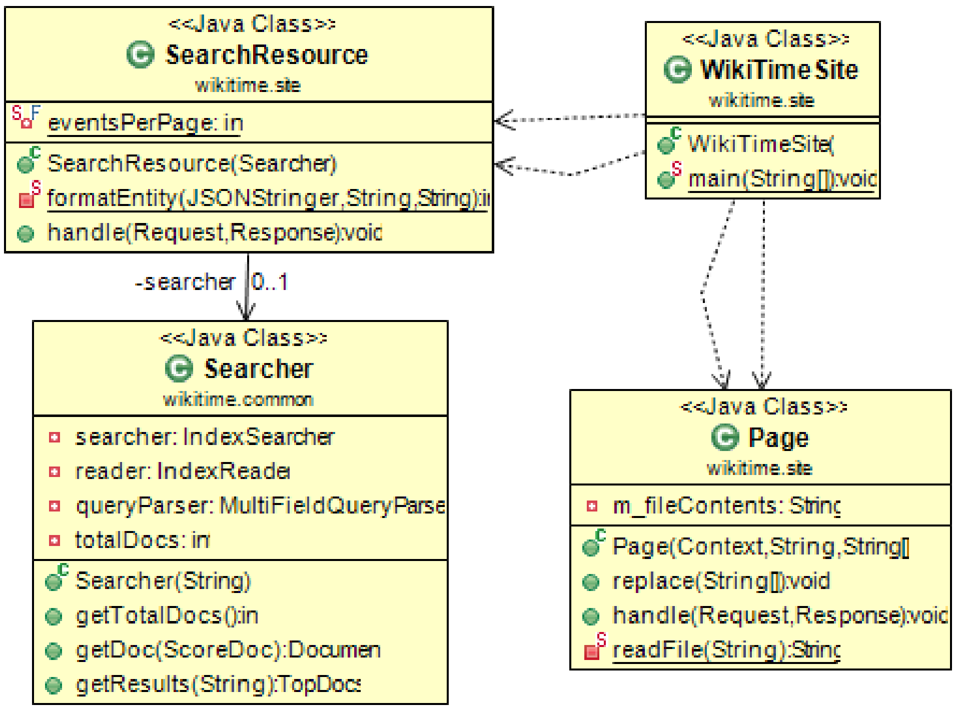
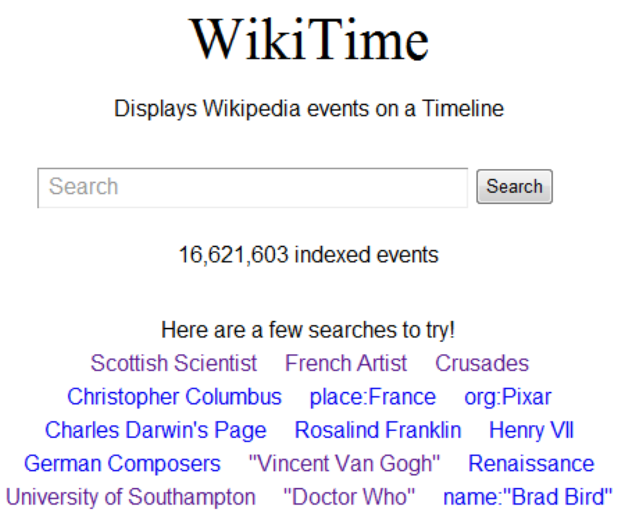


Figure 15: Server Class Diagram

An additional problem with the interface was that a small number of unimportant events would appear far away from the majority of the results with a large horizontal spacing between them, forcing the the display to zoom out. This was resolved by calculating the average year and standard deviation for the event results which was used to select the initial view position and zoom level.


WikiTime

Displays Wikipedia events on a Timeline

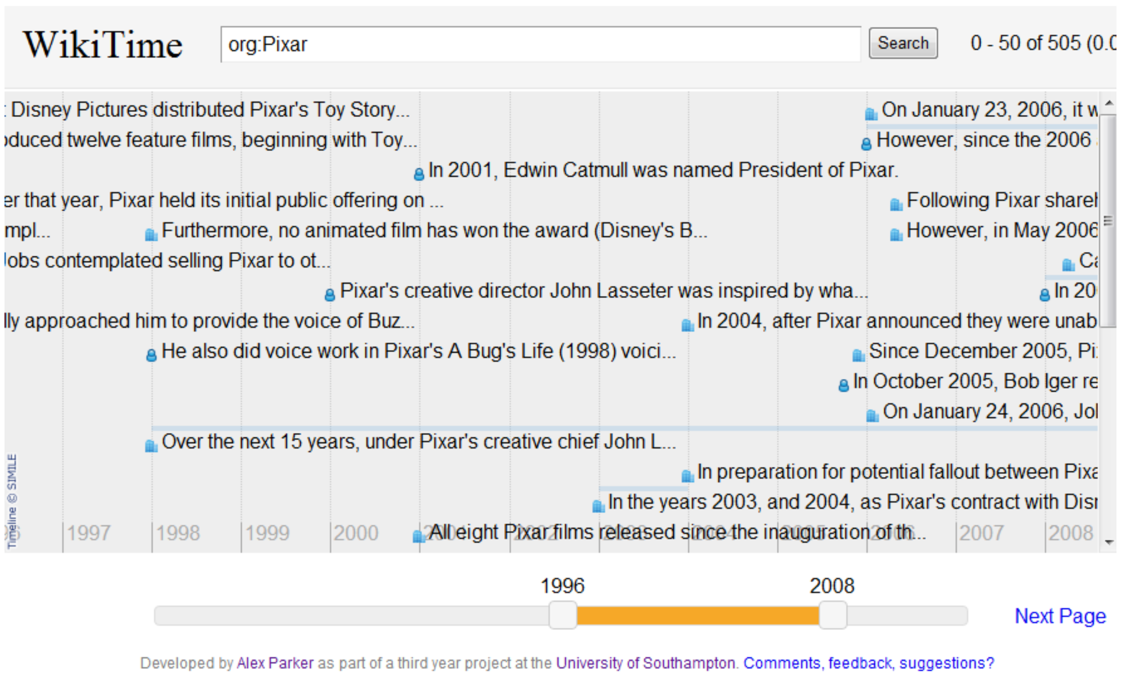
Search Search

16,621,603 indexed events

Here are a few searches to try!

Scottish Scientist French Artist Crusades
Christopher Columbus place:France org:Pixar
Charles Darwin's Page Rosalind Franklin Henry VII
German Composers "Vincent Van Gogh" Renaissance
University of Southampton "Doctor Who" name:"Brad Bird"

(a) "/" - Homepage


WikiTime org:Pixar Search 0 - 50 of 505 (0.0

: Disney Pictures distributed Pixar's Toy Story...
On January 23, 2006, it w
›duced twelve feature films, beginning with Toy...
However, since the 2006
In 2001, Edwin Catmull was named President of Pixar.
er that year, Pixar held its initial public offering on ...
Following Pixar share
mpl...
Furthermore, no animated film has won the award (Disney's B...
However, in May 2006
Jobs contemplated selling Pixar to ot...
Ca
Pixar's creative director John Lasseter was inspired by wha...
In 20
lly approached him to provide the voice of Buz...
In 2004, after Pixar announced they were unab
He also did voice work in Pixar's A Bug's Life (1998) voici...
Since December 2005, Pi
In October 2005, Bob Iger re
On January 24, 2006, Jo
Over the next 15 years, under Pixar's creative chief John L...
In preparation for potential fallout between Pix
In the years 2003, and 2004, as Pixar's contract with Dis
All eight Pixar films released since the inauguration of th...

Developed by Alex Parker as part of a third year project at the University of Southampton. Comments, feedback, suggestions?

(b) *"/search?"* - Search Results Page

Figure 16: Website Screenshots

5 Testing

Various aspects of the final system were tested to ensure it met the functional and performance requirements of the system design. Unit testing was employed to ensure functional requirements were met for the various parts of the system. Additionally performance comparisons between the various prototype event extraction algorithms were made both in terms of performance in pages per second and in the accuracy of the extracted information.

5.1 Unit Testing

Unit test code was written with the JUnit unit testing framework[26]. Tests were written for the website server and for the event extraction and processing framework.

5.1.1 Event Extraction Unit Tests

Name	Description
ArticleTest.testSimpleArticle()	Tests that the Article class parses the Wikipedia XML dump correctly into plain text with the correct fields (pageid, title, content, external links) ensuring that the content doesn't contain any Wikipedia markup.
ArticleTest.testArticle()	Tests the Article class alternative constructor that takes the page title and content only
DateTest.testEqual()	Tests the Date.equals(Date) function
DateTest.testAfter()	Ensures the Date.after(Date) function works correctly
DateTest.testFormat()	Tests that the Date.format() function returns dates in the expected format, e.g. "1984,05,01" for the 1st of May 1984
DateTest.testString()	Tests that a Date can be converted to and from a String correctly
DateTest.testEra()	Tests that the Date's era (e.g. "19th Century" and "1890s") is extracted correctly
EventTest.testEvent()	Tests that the event information is extracted correctly
EventTest.testString()	Tests that an event can be converted to and from a String correctly
ExtractorTest.testExtraction()	Tests that events are extracted from a sample Wikipedia article
ExtractorTest.testBabylon()	Tests event extraction from the Babylon Wikipedia article ensuring that all extracted events are BC events
ExtractorTest.testByte()	Tests event extraction from the Byte Wikipedia article
ExtractorTest.testAMD()	Tests event extraction from the AMD Wikipedia article
ExtractorTest.testAlfred()	Tests event extraction from the King Alfred article
ExtractorTest.testEinstein()	Tests event extraction from the Albert Einstein article
ExtractorTest.testDateExtraction()	Tests date extraction producing a histogram of extracted dates
DuplicateRemoverTest.testSpotSig()	Test the spot signature generation algorithm implementation
IndexerTest.testIndexer()	Tests the EventIndexer class and ensures that articles can be found using the frontend Searcher class

5.1.2 Website Unit Tests

To test the website server functionality the HTTP unit test framework "httpunit" [19] was used which allows a unit test to check that the web server returns valid HTML when sent various requests. For website load testing the Apache HTTP Server Benchmarking Utility *"ab.exe"* [12] was used which makes multiple concurrent requests to a single URL to test how the server software performs when under heavy load from multiple users. The results of this load test can be seen in Figure 20.

Name	Description
SiteTest.testInterface()	Tests that the server presents the home-page and that the server presents the search interface
SiteTest.testAPI()	Tests that the server presents JSON via the REST[13] search API
SiteTest.testFail()	Tests that the 404 error is displayed for an invalid URL
LoadTest.loadTest()	Uses the Apache HTTP Server ab Utility to test performance

5.2 Event Extraction Accuracy

An attempt to compare event extraction accuracy was made over four different Wikipedia articles; "Christopher Columbus", "Advanced Micro Devices", "Alfred the Great" and "Byte". First the total number of events in each article was estimated by hand, and then compared with the total number of extracted events output from the Perl prototype and the final extractor to produce a percentage. The error measurement was measured by hand looking for extracted events that contained only the date or treated a number as a year, e.g. *"18th November 1826."* and *"AMD Virtualization is also supported by release two (8200, 2200 and 1200 series) of the Opteron processors"*.

Figure 17 shows the percentage of events extracted for the Perl prototype and the final system compared with the total events that have been extracted by hand for each article. The graph shows that the final extractor decreased the error rate compared with the Perl extractor for the *"Christopher Columbus"* and *"Advanced Micro Devices"* articles, though fewer events were extracted correctly. The final extractor handled the *"Alfred the Great"* page much better due to the improved date extraction algorithm allowing for dates before 1000 AD, however the weakness of this algorithm is shown in the *"Byte"* page where the final extractor makes multiple mistakes incorrectly classifying a number as a date. Since there are a large number of articles in Wikipedia the accuracy of the extracted events is more important than the quantity, meaning that the final system is an improvement over the prototype.

[13]Representational State Transfer

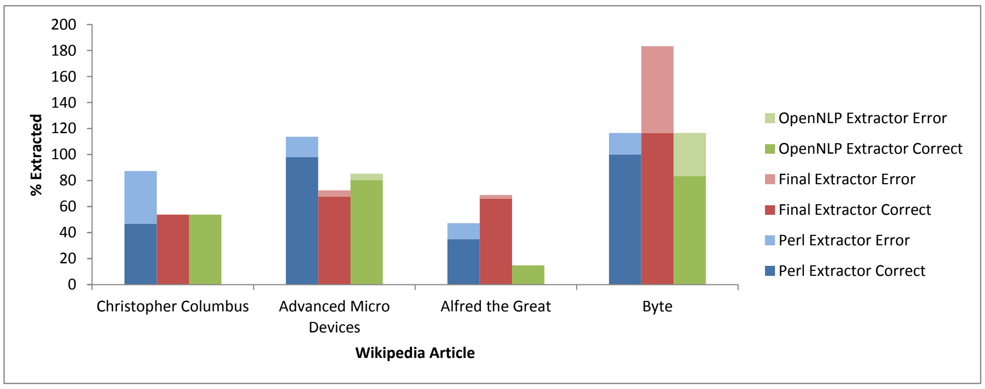


Figure 17: Event Extraction Accuracy

Comparing the OpenNLP parser with the final parser the final parser handles pages with dates below 1000 AD better than the OpenNLP parser but at the cost of a higher error rate, with many more errors on more difficult pages such as the *"Byte"* article and fewer extracted events on the *"Advanced Micro Devices"* page. However the improved extraction performance on the important historical articles *"Alfred the Great"* and *"Christopher Columbus"* means that the final system performs better than the OpenNLP parser, but clearly further improvements could be made.

5.3 Event Extraction Performance

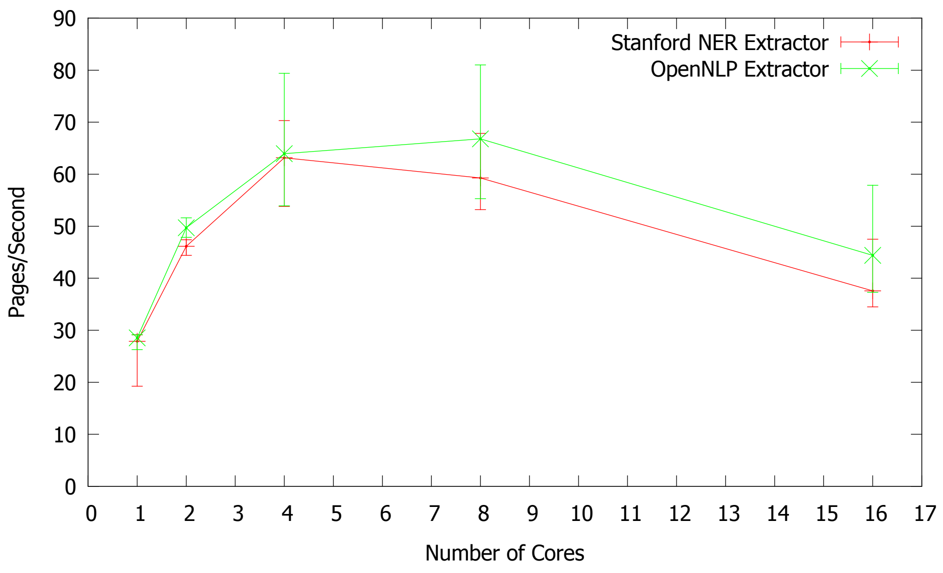


Figure 18: Extraction Performance over multiple threads

Figure 18 shows extraction performance between the OpenNLP parser and the Stanford NER parser in the final system as the number of extraction threads increase. The benchmarks were run on a 16 core Intel Xeon CPU with 47GB or RAM. The test was run 10 times with the minimum, maximum and average extraction time shown for each point.

This graph shows that for a 16 core machine the best extraction performance occurs around 8 cores. The extraction times between the Stanford NER parser and the OpenNLP parser are largely similar with the OpenNLP extractor providing slightly higher performance which is balanced by the reduced extraction accuracy on the *"Christopher Columbus"* and *"Alfred the Great"* articles shown in Figure 17.

	Final System	OpenNLP	Map Reduce	Prototype
Measure 1	66.9 s	86.3 s	41.0 s	21.6 s
Measure 2	67.5 s	83.6 s	41.0 s	19.4 s
Measure 3	64.2 s	92.5 s	40.0 s	20.7 s
Measure 4	65.0 s	101.8 s	40.0 s	22.2 s
Measure 5	69.6 s	89.7 s	38.9 s	20.4 s
Average	66.6 s	90.8 s	40.2 s	20.9 s
Result	14.1 pages/s	10.4 pages/s	23.4 pages/s	45.0 pages/s

Figure 19: Extraction Performance over all prototypes

Figure 19 shows the total extraction time for the first 940 articles from the Simple English Wikipedia extracted on a Core 2 Duo 2.2GHz machine with 2GB of memory. The extraction time was measured five times and then averaged and divided by the number of pages to arrive at a pages per second measurement. The performance measurements for the final system and the OpenNLP parser are lower than in Figure 18 because a less powerful machine was used for the performance measurements. The extraction is compared on a single thread ignoring the scalability of the processing framework and the map reduce framework which can be distributed to process pages much quicker to allow for a fair comparison with the Perl prototype.

The results show that the performance of the extraction system decreased as the complexity of the event extraction algorithm increased, with the original Perl prototype providing the best performance with the worst extraction accuracy. The Map Reduce prototype halves the performance due to the increased overheads of the map reduce framework, but this is balanced by the scalability of the map reduce framework. The final system and the OpenNLP extractor give the worst performance but the best extraction accuracy however again this is mitigated by the scalability of both systems onto multiple cores as shown in Figure 18.

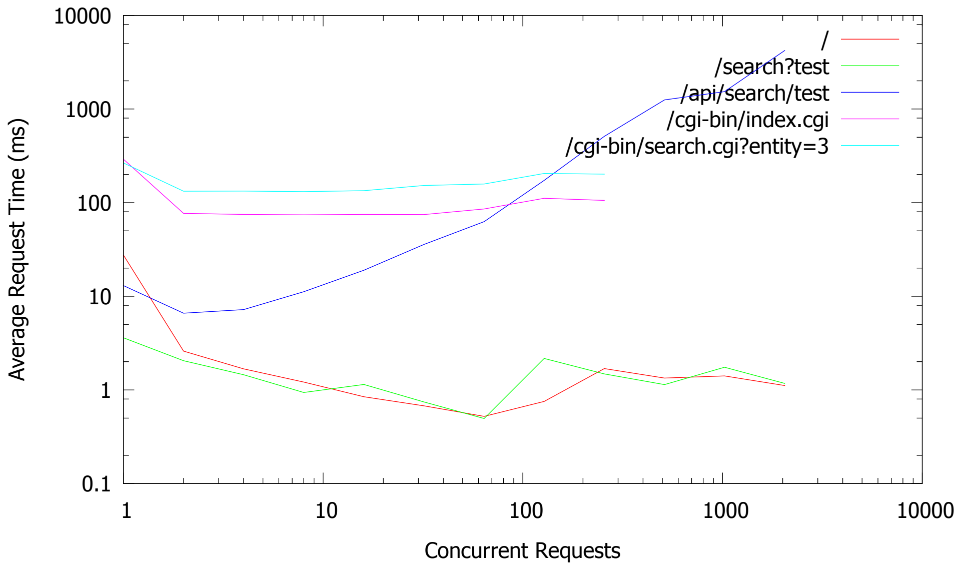


Figure 20: Frontend Performance as the number of concurrent users increases

5.4 Frontend Performance

Figure 20 shows the average response time as the number of concurrent users increases for different pages for the final system and the initial prototype. The benchmark was run using the Apache HTTP Server benchmarking tool (*ab.exe*) [12] with 1 to 2048 concurrent users making 10 requests each on a Core 2 Duo 2.2GHz machine with 2GB of memory.

The initial prototype (*"/cgi-bin/index.cgi"* and *"/cgi-bin/search.cgi?entity=3"*) is slower than the final system but it appears to scale to many concurrent users well. The final system returns the static pages (*"/"* and *"/search?test"*) in $1ms$ consistently however the search API (*"/api/search/test"*) decreases in performance as the number of users increases. Clearly this is would be the bottleneck if the website comes under heavy load, and could be resolved by mirroring the website onto multiple web servers backed up with a load balancer service [11].

6 User Evaluation

To evaluate the effectiveness of the final website a usability survey was conducted in the form of a questionnaire which was based on the open qualitative and the quantitative sample questionnaires found in the *"User-Centred Web Design"* book [8]. Members of the general public were invited to participate and were recruited from the Wikipedia user mailing list[14], the History Teacher's Discussion Board[15] and the author's personal Facebook[16] and Twitter[17] accounts. The Wikipedia and History Teacher's Forum were selected since these sites have demographics that might use the WikiTime system.

To take part in the survey users were first asked to evaluate the WikiTime website[18] taking as much time as they needed. Once they had finished evaluating the website they were asked to enter their thoughts into the survey which used the iSurvey website [39]. The entire process was conducted online to avoid affecting users by observing them.

6.1 Usability Evaluation

Figure 21 shows the usability measures for the final interface, which were measured by averaging user responses from statements such as *"I feel it enhances and enables my skills"* from Question 2.1 in the questionnaire. Users were asked to respond on a seven point scale where one meant that the user completely disagreed with the statement and seven meant that the user completely agreed with the statement. These measures were then averaged under a number of usability headings and rescaled to a percentage to be displayed on the final graph. The error bars show the standard deviation for each heading.

The graph shows that the main issues with the interface are the usefulness, i.e. the results returned matched what the user was expecting, and the presentation of those results, i.e. the user interface was easy to understand and results were presented clearly. The presentation measure could be improved through a more thorough usability study to discover what difficulties users are having, and further work to improve the timeline layout and rendering. The usefulness of the results could be improved through further research into the event extraction algorithm to remove incorrect events and a text summation algorithm to generate titles for each event. These improvements should bring the overall usability to around 70-80% which is a standard usability target for this type of system [8].

Another measure of usability is the length of time that users spent on the site, which according to the site analytics averaged at two minutes per user, with some users spending 20-30 minutes, whilst the average time reported on the survey was twelve minutes. Figure 22 shows how the average site visit time and pages/visit increased as improvements to the extraction algorithms and website were made during the project. In general a higher number of pages/visit and site visit time indicates increased user engagement, but this can be misleading since a low number of pages/visit combined with a low time spent on the site can by typical of search engine sites [33].

[14]https://lists.wikimedia.org/mailman/listinfo/wikien-l
[15]http://www.schoolhistory.co.uk/forum/index.php?act=idx
[16]http://www.facebook.com
[17]http://www.twitter.com
[18]http://wikitime.ecs.soton.ac.uk

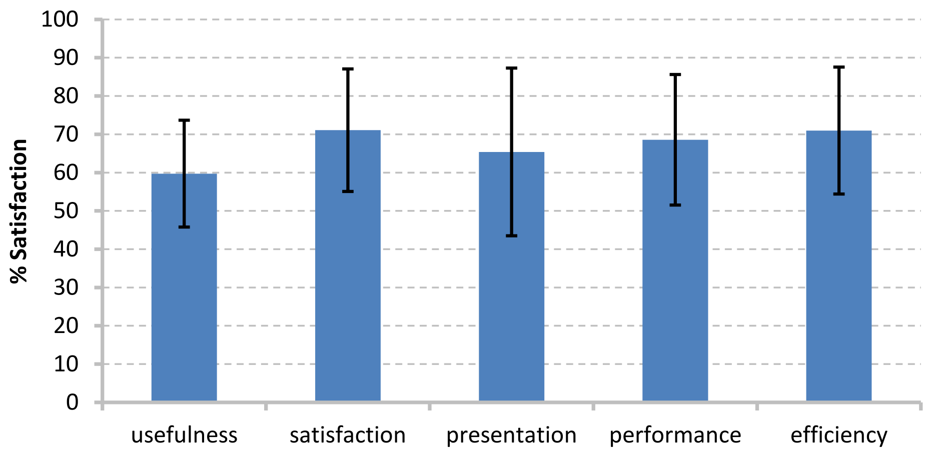


Figure 21: Usability Survey

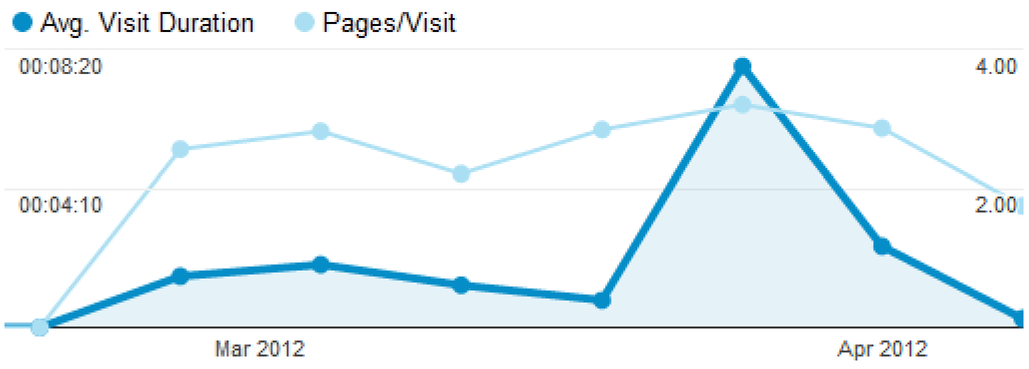


Figure 22: Average Site Visit Time

6.2 User Feedback

The final section of the questionnaire asked users about what they liked and disliked about the website, and for any further comments that they had, which helped to highlight usability issues with the user interface, some of which were fixed in later versions of the website. Additionally some users made some suggestions for future improvements to the site and of the concept.

Figure 23 shows the responses provided along with the number of users that suggested them. Most users thought that the system was a good idea producing interesting results with a good layout. The two main issues highlighted were first better integration with Wikipedia since opening a Wikipedia page loses the timeline context and second the slider at the bottom of the user interface caused confusion and difficulties with timeline navigation. Additional problems were with mouse scrolling causing confusion because it scrolls the timeline horizontally which goes against mouse scroll affordance [31] along

Statement	Tally
Produces interesting results	4
A good idea	4
Difficult to use the bottom slider	4
Could be integrated better with Wikipedia	4
Good layout of results	4
Easy to use and accessible	3
Performance is very fast	2
Scroll bar scrolling horizontally is confusing	2
Bad layout, numbers are presented as dates	2
Bad layout, events can pile on top of each other	2
Bad layout, the information is too compressed	1
The icons need a legend	1
The next page button is confusing	1
Performance is too slow	1

Figure 23: User Survey Feedback

with issues with the timeline layout where the view is too zoomed out causing events to stack on top of each other and finally issues with numbers being treated as years creating noise in the display.

7 Project Management

A Gantt chart was created at the start of the project to break down the project into smaller tasks and ensure that everything will get done. Figure 24 shows the initial Gantt chart that was created. A lot of work done on the project so far took less time than anticipated, allowing for an extension of scope from the originally envisioned project. This increase in scope was reflected in the progress Gantt chart shown in Figure 25. The completion of the project shown in Figure 26 largely went to schedule from that point, with the evaluation stage taking longer and the extraction evaluation taking less time than expected.

A risk assessment was also created at the start of the project to identify and mitigate any project risks. These risks are shown in Figure 27 which identifies each risk and the steps that have been taken to manage that risk. The Loss and Probability factors are scored between 0-5 which gives an overall risk factor between 0-10, where a higher value means higher risk.

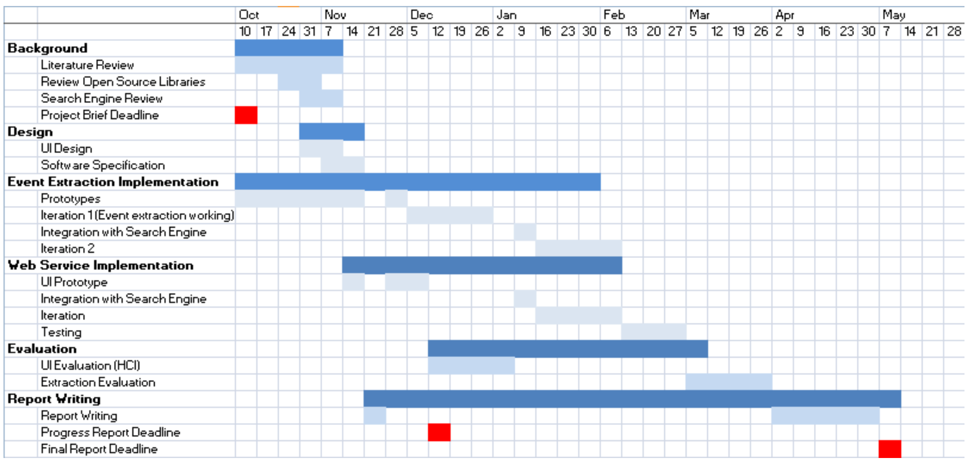


Figure 24: Initial Gantt Chart

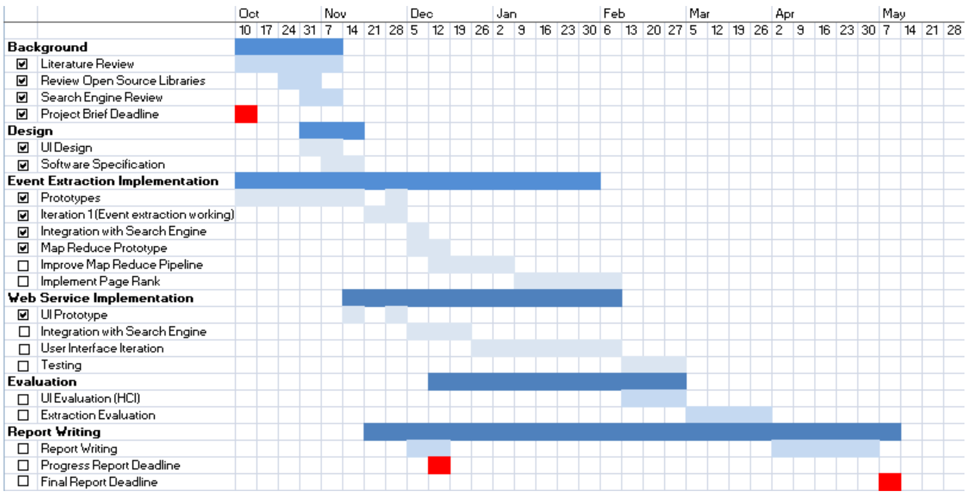


Figure 25: Progress Gantt Chart

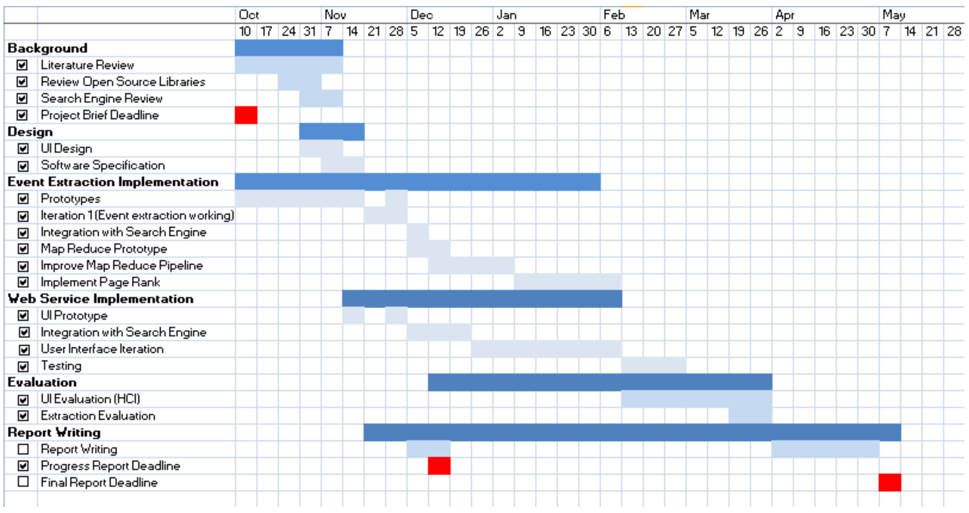


Figure 26: Final Gantt Chart

Problem	Loss	Prob	Risk	Plan
Development machines may be lost or damaged	5	1	5	Code, reports and documents will be stored on bitbucket.org using the mercurial version control system.
Wikipedia goes down or becomes inaccessible	3	3	6	The entire Wikipedia encyclopaedia can be downloaded allowing for processing offline.
Processing of Wikipedia pages takes too long meaning only part of the encyclopaedia is displayed in the final system	2	4	8	Additional computers (such as lab computers) could be used to process the Wikipedia pages. Alternatively sets of related Wikipedia pages could be processed instead.
Development of the event extraction system takes too long	4	2	8	The project can be re-scoped just to cover the event extraction system and run an analysis on the quality of the events extracted from pages.

Figure 27: Initial Project Risk Assessment

8 Conclusion

The project successfully demonstrates a final system that can extract historical events from Wikipedia and display them on a searchable timeline automatically, allowing users to gain an overview of a historical event or famous figure quickly and easily. The project presents an algorithm that can extract dates from free text within a processing framework allowing the processing of Wikipedia within a reasonable timescale, presenting the extracted information to users within an easy to understand interface.

8.1 Further Work

Further improvements could be made to the extraction algorithm, such as better handling of date recognition and extraction. One of the limitations of the algorithm is that it doesn't handle relative expressions e.g. *"3 days ago"* or *"yesterday"* which means that many relative events are missed, a good example of this can be seen in Figure 17 for the Christopher Columbus page. This could be improved by keeping track of the current time in the algorithm, then extracting an absolute event when a relative time expression is given, such as *"yesterday"* or when only a day and month are provided when the year has been mentioned earlier in the page.

A further problem is the misclassification of numbers as dates for example *"ATSC uses 188-byte MPEG transport stream packets to carry data."* is an event with the year 188 in the final system. This problem could be resolved by applying machine learning techniques to build a system that can learn the difference between numbers and years. The Stanford NER parser provides a seven class model which can recognize dates along with time, location, organisation, person, money, and percent.

Additionally further work could investigate better handling of BC dates, since the algorithm currently assumes that all BC dates have the text *"BC"* following the year which introduces problems for date ranges, e.g. *"300-500 BC"*. Furthermore, very old dates are ignored with the present algorithm such as *"320 million years ago"* which excludes many biological, geological and astronomical events.

Another problem is when lots of events are mentioned in the same sentence such as in the Abraham article extract in section 4.1.2. This could be fixed by splitting the sentence into smaller fragments adding each individually, or adding the same sentence once for each event, rather than the current implementation which selects the minimum and maximum date mentioned in the sentence. The sentence could be split into smaller fragments by separating the sentence by punctuation marks, however this would lose the sentence context making the extracted event confusing.

The final problem is that the titles of the events are not concise, for example *"Christopher Columbus (unknown; before 31 October 1451 – 20 May 1506) was an explorer, colonizer, and navigator, born in the Republic of Genoa, in what is today northwestern*

Italy." is provided as an event in the current system, which could be summed up as *"Christopher Columbus was an explorer, colonizer and navigator."* which shortens the sentence allowing more information to be displayed on the timeline overall. This process could be automated by using named entity extraction with connecting words (e.g. was, is, he, she) to generate this summarized text automatically, perhaps moving the automatically generated timelines closer to hand authored timelines.

Many issues with the user interface were identified in the user survey feedback shown in Figure 23. One of the main issues identified was the inadequate integration with Wikipedia. The current user interface provides a link back to the Wikipedia article that the event was extracted from, which forces the user to search through the article for the sentence and it also causes the timeline to disappear causing the user to lose context.

Ideas to improve this could be explored, for example the timeline could be integrated into the WikiMedia software itself as an extension [16] which would prevent the loss of context. Alternatively the current user interface could display the current Wikipedia article content in a frame at the bottom of the screen, highlighting the sentence of the current event, which would allow users to get the context around the selected event.

The timeline scroll bar at the bottom of the screen could be easier to use, as many users reported problems making sense of the scroll bar and understanding it's utility, so further work could identify these usability problems more thoroughly and attempt to create a timeline interface that displays the events more clearly.

Further ideas for extensions include providing a linked data endpoint for the information extracted from the WikiTime project, which would be provided in the RDF format. This could allow the project to link into the DBpedia project enabling timelines to be associated with the entities found within the DBpedia project, for example.

Finally the idea of producing timelines from Wikipedia could be extended to the rest of the internet, allowing users to find historical information in a new, more structured way than the present keyword search. However the size of the internet creates challenges in scaling the system to process the large amounts of information involved, as well as additional work in parsing and extracting information from the free text mentioned on web pages. A further issue would be how trustworthy and accurate the information extraction would be, for example timeline information extracted from a discussion board is likely to be less trustworthy than information extracted from Wikipedia.

8.2 Evaluation

The main issues with the project are that some of the usability issues with the user interface, such as the scroll bar at the bottom of the screen and some of the event layouts were not resolved. The usability survey in Figure 21 gave an average satisfaction of 65%, whereas a similar interface should expect a score of around 75-80% satisfaction

[8]. In hindsight the project should have produced an additional usability survey earlier in the project using the feedback to show an improvement in usability between the prototype and final user interface.

Another issue with the project was that more automated testing to evaluate the quality of the extracted events could have been used, since most of the extraction quality and performance measurements were taken during the project write up, rather than allowing these measurements to inform software choices and quantitatively show the improvement during the projects implementation.

To conclude, the major goals of the project as outlined in the project brief and progress report were met. These goals were to conduct a review of existing algorithms and libraries, to build a system that can extract temporal events from Wikipedia and to display those extracted events on a Timeline. Some extensions mentioned in the progress report were also implemented, such as duplicate event detection and removal, an implementation of the page rank algorithm, and showing that the system could be scaled to the full English Wikipedia rather than the Simple English Wikipedia.

References

[1] John Aberdeen et al. “MITRE: description of the Alembic system used for MUC-6”. In: *Proceedings of the 6th conference on Message understanding*. MUC6 ’95. Columbia, Maryland: Association for Computational Linguistics, 1995, pp. 141–155. ISBN: 1-55860-402-2. DOI: http://dx.doi.org/10.3115/1072399.1072413. URL: http://dx.doi.org/10.3115/1072399.1072413.

[2] *Abraham*. 2012. URL: http://en.wikipedia.org/wiki/Abraham.

[3] Stefano J. Attardi. *Labels behind form fields (Attardi.org)*. 2012. URL: http://attardi.org/labels/.

[4] Christian Becker. “DBpedia - Extracting structured data from Wikipedia”. In: Presentation at Wikimania 2009, Buenos Aires, Argentina, 2009.

[5] Hal Berghel. “Cyberspace 2000: dealing with information overload”. In: *Commun. ACM* 40 (2 1997), pp. 19–24. ISSN: 0001-0782. DOI: http://doi.acm.org/10.1145/253671.253680. URL: http://doi.acm.org/10.1145/253671.253680.

[6] Tim Berners-Lee. *Cool URIs don’t change*. 1998. URL: http://www.w3.org/Provider/Style/URI.

[7] Sergey Brin and Larry Page. “The anatomy of a large-scale hypertextual Web search engine”. In: *Computer Networks and ISDN Systems* 30 (1998), pp. 107–117.

[8] John Cato. *User-Centered Web Design*. Pearson Education Limited, 2001. ISBN: 0 201 39860 5.

[9] Tim Berners-Lee Christian Bizer Tom Heath. “Linked Data - The Story So Far”. In: *International Journal on Semantic Web and Information Systems (IJSWIS)* 5 (3 2009).

[10] Robert Dale and Paweł Mazur. “The semantic representation of temporal expressions in text”. In: *Proceedings of the 20th Australian joint conference on Advances in artificial intelligence*. AI’07. Gold Coast, Australia: Springer-Verlag, 2007, pp. 435–444. ISBN: 3-540-76926-9, 978-3-540-76926-2. URL: http://dl.acm.org/citation.cfm?id=1781238.1781295.

[11] Thomas Decker. *Load Balancing*. 2012. URL: http://www2.cs.uni-paderborn.de/cs/ag-monien/RESEARCH/LOADBAL/.

[12] Apache Software Foundation. *ab - Apache HTTP Server Benchmarking Tool*. 2012. URL: http://httpd.apache.org/docs/2.0/programs/ab.html.

[13] Apache Software Foundation. *Apache Lucene*. 2012. URL: http://lucene.apache.org/.

[14] Apache Software Foundation. *Apache Lucy*. 2011. URL: http://incubator.apache.org/lucy/.

[15] Apache Software Foundation. *Apache OpenNLP*. 2012. URL: http://opennlp.apache.org/.

[16] Wikimedia Foundation. *Manual: Developing Extensions*. 2012. URL: http://www.mediawiki.org/wiki/Manual:Developing_extensions.

[17] Brent Fulgham. *Perl speed vs Java 7 speed — Computer Language Benchmarks Game.* 2012. URL: http://shootout.alioth.debian.org/u32q/perl.php.

[18] Dan Gillick. "Sentence boundary detection and the problem with the U.S." In: *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers.* NAACL-Short '09. Boulder, Colorado: Association for Computational Linguistics, 2009, pp. 241–244. URL: http://dl.acm.org/citation.cfm?id=1620853.1620920.

[19] Russell Gold. *HttpUnit.* 2008. URL: http://httpunit.sourceforge.net/.

[20] Shih-Ting Huang, Tsai hsuan Tsai, and Hsien tsung Chang. "The UI issues for the search engine". In: *Computer-Aided Design and Computer Graphics, 2009. CAD/Graphics '09. 11th IEEE International Conference on.* 2009, pp. 330 –335. DOI: 10.1109/CADCG.2009.5246883.

[21] David François Huynh. *SIMILE Widgets — Timeline.* 2012. URL: http://www.simile-widgets.org/timeline/.

[22] *Java Wikipedia API.* 2012. URL: http://code.google.com/p/gwtwiki/.

[23] Trond Grenager Jenny Rose Finkel and Christopher Manning. "Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling". In: *Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics (ACL 2005)* (2005), pp. 363–370. URL: http://nlp.stanford.edu/~manning/papers/gibbscrf3.pdf.

[24] Paweł Kalczyński et al. "Time-Indexer: a Tool for Extracting Temporal References from Text Documents". In: ed. by Khosrow-Pour Mehdi. Information Resources Management Association. Philadelphia: Information Resources Management Association, 2003, pp. 832–835.

[25] Pawel Jan Kalczynski and Amy Chou. "Temporal document retrieval model for business news archives". In: *Inf. Process. Manage.* 41 (3 2005), pp. 635–650. ISSN: 0306-4573. DOI: 10.1016/j.ipm.2004.01.002. URL: http://dl.acm.org/citation.cfm?id=1063266.1063280.

[26] David Saff Kent Beck Erich Gamma. *JUnit.* 2012. URL: http://junit.sourceforge.net/.

[27] Erdal Kuzey. "Extraction of Temporal Facts and Events from Wikipedia". In: (2011).

[28] Jérôme Louvel. *Restlet - RESTful web framework for Java.* 2012. URL: http://www.restlet.org/.

[29] Benjamin Arthur Lupton. *JQuery History.* 2012. URL: https://github.com/balupton.

[30] Eric Meyer. *CSS Tools: Reset CSS.* 2012. URL: http://meyerweb.com/eric/tools/css/reset/.

[31] Donald Norman. *The Design of Everyday Things.* Basic Books, 1988. ISBN: 978-0-465-06710-7.

[32] C. Okoli. "A Brief Review of Studies of Wikipedia in Peer-Reviewed Journals". In: *Digital Society, 2009. ICDS '09. Third International Conference on.* 2009, pp. 155 –160. DOI: 10.1109/ICDS.2009.28.

[33] Panalysis. *Web Analytics - How to interpret time on site.* 2012. URL: http://www.panalysis.com/web_analytics_time_on_site.php.

[34] M.F. Porter. "An algorithm for suffix stripping". In: *Program* 14 (1980), pp. 130–137. URL: http://tartarus.org/~martin/PorterStemmer/.

[35] JQUERY UI PROJECT. *jQuery UI - Home.* 2012. URL: http://jqueryui.com/.

[36] The JQuery Project. *JQuery: The Write Less, Do More, Javascript Library.* 2012. URL: http://jquery.com/.

[37] Daniel Rosenburg and Anthony Grafton. *Cartographies of Time: A history of the timeline.* Princeton Architectural Press, 2011. ISBN: 978-I-56898-763-7.

[38] Alberto Manuel Brandão Simões. *Lingua::EN::NamedEntity Perl Module.* 2011. URL: http://search.cpan.org/dist/Lingua-EN-NamedEntity/NamedEntity.pm.

[39] University of Southampton. *iSurvey - Online Questionaire Generation from the University of Southampton.* 2012. URL: https://www.isurvey.soton.ac.uk/.

[40] Ashish Sureka, Pranav Prabhakar Mirajkar, and Kishore Indukuri Varma. "A rapid application development framework for rule-based named-entity extraction". In: *Proceedings of the 2nd Bangalore Annual Compute Conference.* COMPUTE '09. Bangalore, India: ACM, 2009, 25:1–25:4. ISBN: 978-1-60558-476-8. DOI: http://doi.acm.org/10.1145/1517303.1517330. URL: http://doi.acm.org/10.1145/1517303.1517330.

[41] Martin Theobald, Jonathan Siddharth, and Andreas Paepcke. "SpotSigs: robust and efficient near duplicate detection in large web collections". In: *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval.* SIGIR '08. Singapore, Singapore: ACM, 2008, pp. 563–570. ISBN: 978-1-60558-164-4. DOI: 10.1145/1390334.1390431. URL: http://doi.acm.org/10.1145/1390334.1390431.

[42] Enrique Vallés and Paolo Rosso. "Detection of near-duplicate user generated contents: the SMS spam collection". In: *Proceedings of the 3rd international workshop on Search and mining user-generated contents.* SMUC '11. Glasgow, Scotland, UK: ACM, 2011, pp. 27–34. ISBN: 978-1-4503-0949-3. DOI: http://doi.acm.org/10.1145/2065023.2065031. URL: http://doi.acm.org/10.1145/2065023.2065031.

[43] Jason Venner. *Pro Hadoop.* Springer, 2009. ISBN: 978-1-4302-1943-9.

[44] Wikipedia. *Battle of Hastings.* 2011. URL: http://en.wikipedia.org/wiki/Battle_of_Hastings.

[45] Wikipedia. *Special:Statistics.* 2011. URL: http://en.wikipedia.org/wiki/Special:Statistics.

[46] Wikipedia. *Wikipedia:Database download.* 2011. URL: http://en.wikipedia.org/wiki/Wikipedia:Database_download.

[47] Sholomo Yona. *Lingua::EN::Sentence Perl Module.* 2011. URL: http://search.cpan.org/~shlomoy/Lingua-EN-Sentence-0.25/lib/Lingua/EN/Sentence.pm.

[48] Qi Zhang et al. “Efficient partial-duplicate detection based on sequence matching”. In: *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval.* SIGIR ’10. Geneva, Switzerland: ACM, 2010, pp. 675–682. ISBN: 978-1-4503-0153-4. DOI: `http://doi.acm.org/10.1145/1835449.1835562`. URL: `http://doi.acm.org/10.1145/1835449.1835562`.

Protocol

Study Title: WikiTime Evaluation

Researcher: Alex Parker (ajp3g08@ecs.soton.ac.uk)

Funder: Unfunded
Sponsor (if known):

Revision History

Revision	Date	Narrative
1.0	26th February 2012	Initial submission

Background

The WikiTime project is a software system that extracts historical events from Wikipedia and makes these events available from a searchable timeline interface. The intent of this study is to evaluate the usability of the WikiTime project and its usefulness.

Method

To achieve this, a user evaluation will take place by asking participants to evaluate the system online in their own time. Once the participant has evaluated the system they are asked to answer a questionnaire about their experience using the tool. This entire process will take place on the participant's personal computer online.

Materials

The questionnaire that will be conducted for this evaluation can be found in the appendix.

Participants

Members of the general public will be invited to participate in the questionnaire. The survey will be targeted at participants who might be interested in the site, such as those who use or write for Wikipedia or have an interest in history.

A link to the survey will be emailed in the Wikipedia user mailing list https://lists.wikimedia.org/mailman/listinfo/wikien-l since this project might be of interest to the English Wikipedia community, as well as the History Teacher's Discussion Board http://www.schoolhistory.co.uk/forum/index.php?act=idx since WikiTime could be of interest to history teachers as a teaching resource, finally if recruitment is insufficient I will post survey links on my personal Facebook and Twitter account.

Procedure
Participants are asked to evaluate the system in their own time by following a link from the consent page on iSurvey. Once they have completed the evaluation of the site they answer the questionaire. The entire process is done on the participants computer without any assistance or interference from any assistants or collaborators.

Statistical analysis
The answers given for the quantitative parts of the questionnaire will be aggregated under categories to gives score for the usability of the system. A content analysis of the textual responses for the qualitative parts of the questionnaire will be done to provide a summary of the feedback overall. An attempt to look at the correlation between the participant's level of interest in history and level of education will be made.

Ethical issues
There are possible ethical issues with asking for the participant's level of education, which is mitigated by the study size and the anonymous collection of the study data.

Data protection and anonymity
The level of education and interest in history is collected from each participant, which might affect anonymity of the data, however the iSurvey software is used which permits collection of anonymous unlinked data since there is no processing of emails or names by the researcher.

The data collected will only be disclosed to myself and possibly my supervisor for the purpose of analyzing and understanding that data. The results of this analysis, including a selection of the opinions given will be published in my final report. The data once collected will be encrypted during the study and deleted once the results have been analysed at the end of the study.

WikiTime Evaluation Questionaire

Welcome Notice

Hi there,

Thanks for your interest in WikiTime! WikiTime is a website which presents all of the events in history that are mentioned in the Wikipedia encyclopedia on a Timeline which you can search over.

This evaluation is intended to test the design, usability and usefulness of the site and your input will be used to discover what about the site is good, what is bad, and any other opinions that you may have. All data is collected anonymously and you must be 18 years or older to take part. There is no obligation to complete this evaluation once started and you are free to leave at any time. The data collected is anonymous and will be stored at the University of Southampton and used for research purposes. This questionnaire has been reviewed by the University of Southampton Ethics Committee (reference number: xxxxxx)

Before starting this survey please evaluate the website first by visiting:
http://wikitime.ecs.soton.ac.uk/

Thanks for your time,
Alex Parker.

Section 1. About You

Question 1.1

Text:
What is your level of Education?

Responses:
One of the following options can be selected:

1. School Level (O-Levels, GCSEs, etc)
2. Further Education (A-Levels, BTECs, NVQs)
3. Undergraduate Degree
4. Postgraduate Degree
5. Doctorate

Question 1.2

Text:
Are you a member of a historical group or society that studies History?

Responses:
One of the following options can be selected:

1. Yes
2. No

Question 1.3

Text:
How interested are you in history?

Responses:
One of the following options can be selected:

No Interest				Very Interested		
1	2	3	4	5	6	7

Section 2. About the Site

Question 2.1

Text:
Please read the statements below and score each with a number between 1 and 7 to indicate how true the statement is for you.

Response:
A response scaling from 1 to 7 is indicated for each statement.

Totally Disagree	1	2	3	4	5	6	7	Totally Agree
I feel it enhances and enables my skills.								
I never felt lost on this system.								
I always feel in control using it.								
I found it a waste of my time.								
I found nothing new or interesting in this system.								
It is very easy to understand straight away.								
I could always find what I wanted quickly.								
I will tell my friends positive things about it.								
It was always clear what would happen when I clicked on something.								
The results returned matched what I expected.								
I found the visual presentation excellent.								
The system was coherent and consistent.								
I learnt something new with this system.								
I found it very useful.								
I found the performance too slow.								

Section 3. Your Comments

Question 3.1

Text:
How many minutes were you on the website?

Response:
The participant indicates how long they spent on the website in minutes in the box.

Question 3.2

Text:
Would you expect to use the site again?

Response:
One of the following options can be selected:

1. No
2. Sporadically
3. Periodically
4. Regularly

Question 3.3
Text:
What did you like about the site?

Response:
The participant provides textual feedback.

Question 3.4
Text:
What did you dislike about the site?

Response:
The participant provides textual feedback.

Question 3.5
Text:
Do you have any further comments on the site?

Response:
The participant provides textual feedback.

Completion Notice
Thank you for taking the time to complete this questionnaire!

Extraction of Temporal Events from Wikipedia

A system that extracts events from Wikipedia

By Alex Parker (ajp3g08@ecs.soton.ac.uk)

Supervisor: Dr Adam Prugel-Bennett (apb@ecs.soton.ac.uk)

Project Brief

This document outlines the basis for the project envisioned.

Problem

Wikipedia is a large source of information that can provide you with in depth knowledge about a specific topic. However there is no intuitive way to see how events stored within Wikipedia relate to each other over time, and there is too much information for it to be categorised and linked by hand.

The relatively new fields of Information Extraction and Computational Linguistics provides a way to automatically extract this information within a reasonable timeframe. Hence this project aims to automate the extraction of times, dates and named entities from sentences within Wikipedia.

The main issues with automatic information extraction is the quality and accuracy of the information given the wide range of meanings found in English, such as how dates and times are described. Other problems are with the description of an event as events can be a single event, or can span a range of times, or have multiple occurences. This project will not be interested in events without any date or time.

The proposed system will attempt to extract and semantically link events, remove duplicates and enter them into a search engine. A website frontend will be produced that will query the search engine and display the events on a timeline, linking the events back to the relevant Wikipedia pages.

Goals

1. Conduct a review of existing algorithms and libraries that could be used to gather temporal events from free text such as the OpenNLP, FreeLing, and MALLET libraries.
2. Extract temporal events by parsing free text collected from Wikipedia, either collected by a web crawler or by downloading the encyclopaedia and processing it offline.
3. Create a web service that allows you to search and browse wikipedia by events on a Timeline.

Appendix D: Software CD

Directory Listings

```
README.txt - Describes how to use the software!

/common - Classes shared between the extraction and website
        src/    - Common source folder
        lib/    - Dependencies shared between the website and extraction system

/mapreduce - The event extraction system
        ner.ser.gz      - Classifier used by the Stanford NER parser
        Extractor.jar   - Tool that runs the event extraction processes
        src/            - Source folder
        lib/            - Java libraries and project build dependencies

/site - The website that displays timelines
        ab.exe          - Apache HTTP Server Benchmarking Tool
        index.html      - Home page HTML
        search.html     - Search page HTML
        Site.jar        - Website server executable
        src/            - Server source code folder
        lib/            - Website libraries and project build dependencies
        static/         - CSS layout, images, icons and Javascript code and libraries
```