Imperial College London

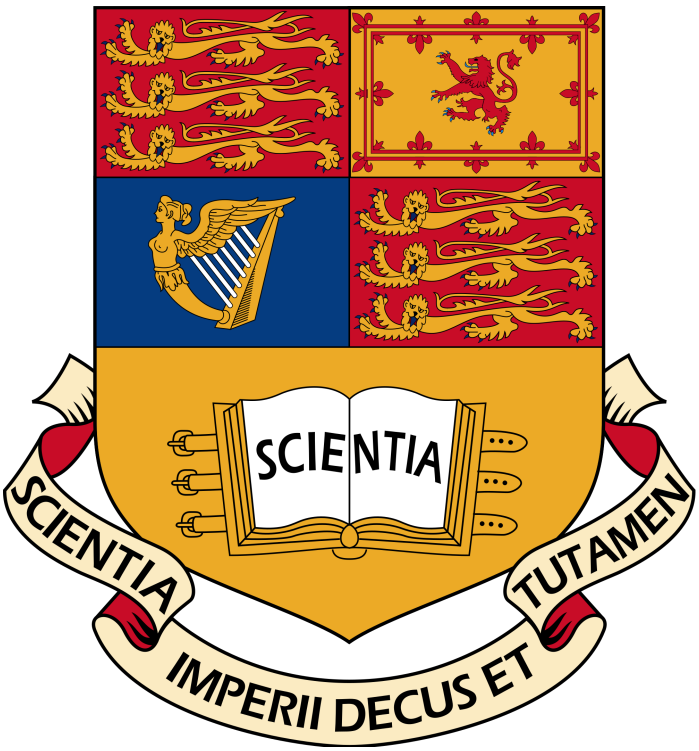
Department of Computing

MEng Individual Project

Automated Creation of Infographic Event Timelines from Text

Author:
Adam Wiles

Supervisor:
Professor William Knottenbelt



June 17, 2017

Abstract

It is often said that *a picture is worth a thousand words*. This being a saying that takes on a far more literal meaning in the context of an event timeline: "A graphical representation of a period of time, on which important events are marked" [1]. A timeline can be an incredibly insightful way to visualise the sequence of events in a piece of text, appealing to our natural ability to recognise patterns in an image to quickly reveal story sub-plots and significant character interactions. Yet the benefits of such a representation are rarely observed; at present such timelines are often painstakingly produced by hand, requiring us to first identify the events in our text, pick out any key details we wish to reflect in the timeline, and finally deduce the best layout for the resulting timeline in order to obtain the clearest insight.

However, significant progress has been made in the area of natural language processing in recent years, enabling computers to better understand human language. These continued advancements make the creation of timelines from text a potentially automatable process, and thus forms the overarching objective of this project. In the following report we describe and present a novel application for the automated creation of event timelines for *narrative* texts, significantly reducing the amount of effort required in order to observe the numerous benefits of such a representation.

Our contributions in the following work are three-fold. Firstly, we present a modified hierarchical clustering method that proves effective at identifying the boundaries between distinct events within narrative texts, shown in our evaluation to yield a set of *natural* events that align closely with those users would expect; our approach additionally allowing the user to adjust the level of detail at which events are identified. Secondly, we demonstrate the ability of force-directed graphs to create clear and expressive event timelines that highlight the dynamic interactions between characters within a text. Specifically, demonstrating the ability of force-directed graphs to balance an explicit underlying structure with the *emergent* behaviour that makes them so appealing. Finally, we present a novel end-to-end application for the *semi*-automated creation of infographic event timelines from text. As a result, enabling users to easily and effectively generate their own infographic timelines for any narrative, and providing the means to subsequently edit, explore, and understand the results through an interactive timeline. We ultimately find the inaccuracies of existing NLP tools make it difficult to achieve full automation of the process, however, with a little help from the user an accurate timeline can be obtained with far less effort than at present.

In doing all of this, we also explore and experiment with a number of the current state-of-the-art NLP tools, evaluate the current performance of these systems on narrative texts, and address some of their limitations. In particular, highlighting the need to make these systems more adaptable to different domains of language, with existing development material consisting predominantly of news articles and conversational speech.

Acknowledgements

I must take this opportunity to thank my supervisor, Professor William Knottenbelt, for his endless enthusiasm, valuable feedback, and ability to always keep the bigger picture in mind. It has been a pleasure. Additionally, I must express my gratitude to Dr David Birch for inspiring what this project has become, and sharing his wisdom from his past experience of being in my shoes.

To my family, thank you for your continuous support, love, and advice. And to my friends and colleagues, I've learnt a great deal over the past four years that wouldn't have been possible without you.

Contents

Chapter 1

Introduction

1.1 Motivation

A timeline can be an incredibly useful way to convey the sequence of events that occurred over a period of time, providing insights that may not be so apparent from reading the equivalent information in textual form. Consider the example in Figure 1.1. From this timeline we can immediately see the paths of each actor through time, their interactions with other actors, and their involvement in a number of events, but it would take us much longer to deduce this information if we had to read the entire Lord of the Rings series for ourselves first.

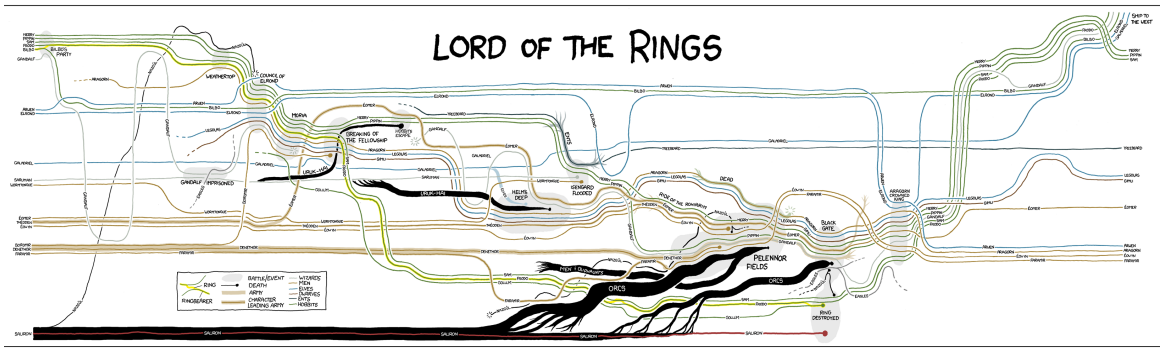


Figure 1.1: Lord of the Rings timeline hand produced by XKCD [2].

And the benefits of timelines extend far beyond this with useful applications in a number of different contexts. Firstly, in the context of teaching a timeline can prove to be an extremely useful aid in highlighting the causal relationships between events and the interactions between characters over time. As a result, helping students to link historical events to one another and to identify common themes that arise over the period of interest in a historical text [3], or alternatively to see and understand how an author introduces characters into a story and interplays the multiple sub-plots that may be described in a piece of English literature [4]. Furthermore, in the context of film scripts, novels and biographical texts, a timeline can help develop context around an event with a concise infographic representation that is far less overwhelming than the equivalent text book alone. Spotting relationships and themes from such a visual representation is far easier than making the logical deductions in your head without the visual aid.

However, at present timelines of the form shown in Figure 1.1 are often painstakingly produced by hand. The few tools that do exist for drawing timelines, discussed further in Section 2.3, currently require that the user provide a precise definition of the events to plot and typically result in far more mundane visualisations that lose the sense of dynamism that's expressed in hand-drawn plots like that of Figure 1.1. As a result, the benefit of a timeline representation like this is rarely seen.

Over the past 10 to 20 years the field of natural language processing (NLP), a field of study situated at the overlap of Computer Science, Artificial Intelligence and Linguistics, has seen significant advances in a number of areas. These dramatic advancements in NLP technologies in recent years has lead to an explosion of applications that can now, relatively robustly, interact with the user at the level of natural language, with prominent examples being Apple's Siri, Google's Assistant, and Amazon's Alexa.

Thus, with applications like these having demonstrated the potential of the current state-of-the-art natural language processing tools and techniques, this project aims to explore the viability of *automatically* constructing a timeline representation of any unstructured text using the current state-of-the-art NLP tools. As a result, enabling anyone to quickly and easily generate timelines for any piece of text, and benefit from the additional insights obtained from such a representation.

1.2 Project Overview

Naturally, the scope of such a project is incredibly wide, depending largely on the resulting visualisation to be constructed. In this project we focus on the core issues central to bridging the gap between receiving any unstructured text as input and producing the corresponding timeline as a result. To this end, we focus on 3 fundamental tasks:

1. Identifying the events of interest in the text.
2. Extracting any supplementary information required for the resulting timeline.
3. Constructing a clear, easily understandable timeline visualisation.

We take the example of Figure 1.1 as our inspiration, aiming to automatically construct a timeline that highlights the individual paths of all characters in the text through each of the events identified. While many other visualisations are possible, we feel that this actor-centric layout applies to a broad variety of input texts and has seen relatively little exploration in the past.

Event extraction, the challenge at the core of this project, is the area of NLP focussed on the identification and extraction of events from text, typically treated as a problem of *information retrieval*. A lot of work has been done on the task of event extraction over the past 10 years, focussed around the common definition of an *event* as something that *happens* or *occurs*, or a *state* or *circumstance* in which something holds true [5]. However, as we explore later, such a definition is somewhat too precise for our purposes. Humans have the natural ability to *summarise* information to varying degrees of detail. For example, if we're describing a movie we just saw to a friend, we'll typically

pick out a few of the key events from the film, which may each summarise a whole series of less significant events. If it turns out our friend has also seen the movie and loved the opening sequence, we may then discuss all the events that occurred within that opening sequence at a much finer level of detail. For this reason, in this project we treat the task of identifying an intuitive set of events from a piece of text as a *clustering* problem: grouping a text into a series of distinct clusters, each representing an *event*, and thus identifying the *boundaries* between the events described in the text.

There are many different methods of clustering data, which we discuss further in Section 2.2, however in this project we have chosen to employ a hierarchical clustering approach. This iterative method gradually groups the initial set of items into distinct clusters, grouping the most "similar" items first and continuing until all items are eventually contained within a single cluster. Unlike flat clustering methods, hierarchical clustering provides us with far more structure. This unique property being extremely useful in the context of clustering text into events, with the clusters formed at different points in the hierarchy reflecting the different levels of detail at which a user may describe the events in a piece of text. At its extremes we have the user reciting the text, sentence-by-sentence, and at the other extreme we have the user summarising the entire text as a single event.

However, in order to facilitate the extraction of a good set of events and to construct the resulting timeline, we must first retrieve all the information required from the input text. Natural language processing is an incredibly diverse field focussed on enabling computers to understand natural language; and while many significant developments have been made, it also remains in its infancy with many sub-fields of NLP still spawning their own dedicated sub-topics as the subtleties and difficulties of each area become apparent. For example, *information extraction*: the task of extracting information from text, has since been broken down into several distinct sub-tasks including temporal-information extraction, event extraction, and named-entity recognition. This decomposition has enabled the development of far more accurate tools in each of these areas, although as we shall come to see, some areas require further refinement than others. Thus, one of our first challenges has been in navigating this vast expanse and finding the best tools for the task in order to provide us with the information necessary to construct an event timeline for a piece of text. Table 1.1 provides a brief overview of some of the key areas of interest that we explore further throughout Section 2.1 and the value that they may provide.

NLP tool	**Value**
Part of Speech Annotation	A task that's fundamental to many higher-level analyses. Provides the ability to identify nouns, verbs and other grammatical arguments, which could be useful for event extraction depending on how we define an *event*.
Coreference Resolution	Identifies repeated mentions of an entity within a text and has been previously used in similar work such as in constructing actor-centric narrative event chains, as discussed in the following chapter.
Temporal Information Retrieval	The task of *identifying* and *normalising* any dates and times within a text. This information could be vital in both determining the ordering of extracted events and extracting the dates of occurrence of key events.
Named Entity Recognition	Recognises mentions of *people*, *places*, *organisations*, and other entities of interest in text. This could provide useful information to reflect in the resulting infographic.
Semantic Role Labelling	Identifies the *semantic roles* of words in text, providing potentially useful information to reflect in the resulting infographic.

Table 1.1: Some of the areas of NLP that may be of relevance to my application, along with the potential value they offer.

With the relevant events identified and any additional information extracted, the visualisation of the resulting timeline remains as the last part of the puzzle. While a number of tools exist for the drawing of event timelines, a few of which we highlight in Section 2.3, there are currently no readily-available tools for constructing timelines of the form shown in Figure 1.1. Treating this as a more general graph drawing problem we could look to incorporate or develop our own graph drawing software with sophisticated heuristics to avoid line crossings and maximise clarity. However, such tools often lack flexibility and the ability for user-interaction: two properties of particular value to our application. Thus, in this project we explore the novel use of force-directed graphs in producing the resulting timeline visualisation. Force-directed graphs employ the use of physical forces between the nodes and edges in the graph to yield an emergent structure at the equilibrium of all forces. By encoding the properties of the data being modelled as physical forces in the graph we get a natural, emergent structure without the need for any complex layout heuristics. This provides us with a clear and intuitive timeline, while also opening the door to a number of other novel visualisations that could provide a new perspective on the text being visualised.

The task of automatically constructing event timelines from text has been cited as a potential application of NLP technologies and the next step in a number of academic papers, such as [6], which also acknowledges the challenges of such a task at this moment in time. Past attempts at creating an application of this type, discussed further in Section 2.5, have seen limited success and have primarily focussed on the extraction of only the most significant event details from historical Wikipedia articles. As a result, we're yet to see a publicly available system for the automated generation of infographic event timelines from text. This project takes a different approach, instead focussing on the domain of narrative texts and the visualisation of the *dynamics* of interactions between actors as opposed to plotting the precise times of key events in factual text.

Finally, as mentioned above, for the purposes of this project we restrict ourselves to the domain of *narrative* texts, such as fairy tales and short novels. Despite aiming to make the application as domain-agnostic as possible, different textual domains exhibit different properties that then lead to slightly different requirements. For example, news articles are not always written in a chronological order potentially requiring a re-ordering of the identified events in order to obtain an accurate event timeline. Narrative documents, on the other hand, typically exhibit the property that the text ordering reflects the chronological ordering of the events described. This removes the need to additionally consider temporal information, which as we see in Section 2.1.4 is one of the more challenging areas of NLP. Additionally, the narrative style of text generalises to a wide variety of texts including fairy tales, short novels, historical accounts and witness statements, allowing us to develop an application that should provide value to a broad target audience. Section 2.4 discusses our consideration of a number of other possible input domains in more detail.

1.3 Objectives

The objectives of this project are as follows:

- Explore the potential of the current state-of-the-art natural language processing tools in facilitating the automated creation of infographic timelines from text.
- Develop a method to automatically generate a *intuitive* set of events from any narrative text at an adjustable level detail.
- Develop a clear and insightful timeline visualisation of the events extracted, and explore the potential for other novel infographic visualisations of these events to gain a new perspective on the input text.
- Create a usable end-to-end application for the automated creation of infographic event timelines from text that is of practical value to to users.

1.4 Contributions

As a result of our efforts in this project, we present four key contributions. Firstly, we provide an exploration of the current state-of-the-art tools in a number of areas of NLP (Chapter 2 and 3), and a thorough evaluation of the state-of-the-art coreference resolution tools when applied to the context of narrative text, revealing the weaknesses of current tools and demonstrating the positive impact of pre-processing narrative texts in order to maximise the performance of coreference resolution tools (Chapter 3 and 6).

We develop an effective technique for the identification of distinct events within narrative texts at an adjustable level of granularity, based upon agglomerative hierarchical clustering (Chapter 4). Our later evaluation (Chapter 7) reveals our approach yields a relatively *intuitive* set of events, particularly at coarser levels of detail[1].

We demonstrate the potential of force-directed graphs in creating a clear yet aesthetically pleasing event timeline, reflecting the paths of each character through a narrative (Chapter 5). In doing so, we demonstrate the ability to impart structure on such graphs without losing the *emergent* behaviour that makes them so appealing.

Finally, we present a novel application for the *semi*-automatic creation of infographic event timelines from text (Chapter 6). We show that despite full automation of this process remaining a difficult challenge due to the inaccuracies of the current state-of-the-art, through a combination of automated natural language processing and an easy-to-use user interface, we can still help users realise the value of timeline representations of text with a great deal less effort than would be required at present (Chapter 7). The result is a potentially marketable web application that enables the *interactive* exploration of the resulting timeline to further the insights obtained from a static image alone.

[1] In this context, we refer to events as being *fine* grained when we consider every single occurrence in the text as an event, while *coarse* grained events begin to split a text into larger "chunks" surrounding more general topics or occurrences

Chapter 2

Background

2.1 Natural Language Processing

2.1.1 Overview

Natural language processing (NLP) is a field of study situated at the overlap of Computer Science and Linguistics, concerned with giving computers the ability to analyse and understand human language. Despite huge advancements in NLP over the past two decades, Sparck Jones makes the observation in her 2001 paper, *NLP - A historical review*, that "the challenge of taking the necessary step from a focused experiment" to"a full-scale rounded-out system" had yet to be overcome [7]. However, NLP has continued to progress, and we are now seeing a number of useful systems emerging in areas such as speech recognition, in the form of virtual smartphone assistants, and also to some extent in the area of Natural Language Understanding in search engines that are now often able to answer our queries before we've even finished typing them. However, there are still a lot of challenges that remain to be overcome before we start seeing more robust systems that are able to interact with the user at the level of human language.

The broad area of NLP can be decomposed into a set of smaller, well-defined sub-tasks that each tackle a distinct challenge, with a lot of higher-level tasks being an extension of the lower-level tasks. Three tasks of particular importance, and often fundamental to a number of NLP tasks, are those of **tokenisation**, **sentence identification**, and **part-of-speech tagging**.

Tokenisation is the task of chopping up text into smaller chunks or phrases, known as tokens. This process can be thought of as breaking down a piece of text into its constituent words, however, in addition to this there is typically a *normalisation* process that follows. This process aims to identify entire phrases that may be of use to later stages of processing. That is, phrases such as "Los Angeles" should typically be grouped into a single token [8]. As a result, tokenisation provides information at a higher level for subsequent NLP processing tools to build upon, aiding further processing and understanding of the original input. [8] provides an interesting discussion as to the possible approaches to tokenisation that can be taken, and their trade-offs; a particular example being the consideration of the differing uses of punctuation in different languages.

Sentence identification extends tokenisation by building upon these results to identify the complete sentences in a given piece of text. These two processes are closely related, as sentence identification requires the consideration of punctuation marks in text; a task that tokenisation must also consider to determine whether, for example, a '.' character is included as part of a decimal number, or denotes the end of a sentence [9].

Lastly, **Part of Speech** (POS) annotation is the process of identifying and assigning a part-of-speech tag to each word in a sentence. The most common parts-of-speech being: **noun, adjective, verb, adverb, adjective, conjunction, preposition, and interjection** [10]. The Penn Treebank project created a corpus of annotated material for POS annotation, containing "approximately 7 million words of part-of-speech tagged text" [11] to help in the development of accurate POS systems, and extends these 8 core annotations to include additional, more sophisticated tags such as the **cardinal number** annotation. That is, the project provided a collection of sample texts that had been manually annotated with the expected POS tags for each word. This material can then be used as training data for attempts to apply machine learning techniques to the problem of POS annotation. In fact, the manual development of a corpus of development material has become a significant part of a number of NLP tasks, and is becoming a major contributor to the success of solutions. The information provided by part-of-speech tags can prove particularly useful in identifying the meaning of a word. For example, whether "fall" is used as a noun or a verb may alter whether we treat the word to mean the season of Autumn, or the action of falling, respectively.

To encourage and accelerate development in the area of NLP, a number of organisations and conferences have emerged over the past 20 years, proposing *shared tasks* in the various areas of NLP. A shared task is essentially a challenge proposed to the NLP community, defining a problem to overcome and typically specifying the expected output format. To support the participants, a range of development material is also often provided, such as an annotated text corpus. A number of these conferences shall be referred to during the upcoming discussion into each of the areas of NLP of relevance to this project, so we shall first briefly introduce some of the most prominent of these conferences in the paragraphs to follow.

One of the earliest of these conferences was the **Message Understanding Conference (MUC)**. As [12] describes, the MUC conference was initially created "to foster research on the automated analysis of military messages containing textual information. Although called conferences, the distinguishing characteristic of the MUCs are not the conferences themselves, but the evaluations to which participants must submit in order to be permitted to attend the conference" [12]. The MUC conferences, and others alike, certainly appear to have had a notable impact on the current state of NLP research, especially as the challenges deviated away from the military-specific challenges of MUC 1, to far broader challenges such as that of MUC 6, supported by DARPA[1], which aimed to "promote and evaluate research in information extraction" [12].

[1]Defense Advanced Research Projects Agency

The **ACE** conference was a successor to the MUC, which specified three challenges: "Entity Detection and Tracking (EDT), Relation Detection and Characterization (RDC) and Event Detection and Characterization (EDC)" [13], which drove advancements particularly in the area of event extraction. While a more recent conference series is the **CoNLL**[2] conference[3]. Similarly to the MUC, each year CoNLL presents a different task, with the 2003 task being that of Language-Independent Named Entity Recognition; this was a task of particular prominence in the area of Named Entity Recognition (NER) that is still used as a benchmark for the evaluation of many of the state-of-the-art NER[4] tools of today. A listing of the other tasks proposed by the CoNLL can be found at http://www.conll.org/previous-tasks.

Lastly, the **OntoNotes project** was not a conference, but rather a project to develop a large corpus of annotated material to enable the application of machine learning techniques to the development of solutions to these NLP tasks. This was in the hope that machine-learnt solutions, as has been largely the case, may well be able to outperform those manually constructed by following a series of pre-defined rules [14]. Again, supported by DARPA, this corpus of material has proven instrumental to advances in many areas of NLP. The OntoNotes release paper from 2005 states the motivation for the development of such a corpus very well, so I shall quote them here:

> "Natural language applications like machine translation, question answering, and summarisation currently are forced to depend on impoverished text models like bags of words or n-grams, while the decisions that they are making ought to be based on the meanings of those words in context. That lack of semantics causes problems throughout the applications. Misinterpreting the meaning of an ambiguous word results in failing to extract data, incorrect alignments for translation, and ambiguous language models. Incorrect coreference resolution results in missed information (because a connection is not made) or incorrectly conflated information (due to false connections). Some richer semantic representation is badly needed." [14]

It is also worth noting at this point that the material is primarily composed of news, broadcast, talk shows, weblogs, usenet newsgroups, and conversational telephone speech. Despite being a relatively broad spectrum of material, this will affect the results produced by solutions trained on this data. The language used in these contexts is often quite different to that used in fairy tales and other contexts, so is something to bear in mind when choosing and evaluating tools that may have been developed using this data.

Of course, there are many other organisations and conferences in addition to those noted above, and it is also not to say that without these conferences the same advancements in NLP would not have been achieved, but they have certainly helped accelerate and guide the development of the current state of NLP technologies.

[2] Conference on Computational Natural Language Learning
[3] http://www.conll.org/
[4] Named Entity Recognition

To permit the consistent evaluation of NLP tools, MUC 2 defined a set of evaluation metrics that have since become the de-facto measures of performance for an NLP tool [12]. These are **recall** and **precision**, defined as follows:

$$recall = \frac{N_{correct}}{N_{expected}} \qquad precision = \frac{N_{correct}}{N_{correct} + N_{incorrect}}$$

where $N_{correct}$ is the number of facts/items correctly identified by the system in question, $N_{expected}$ is the number of facts/items that were expected to be identified from the text, and $N_{incorrect}$ is the number of facts/items that were incorrectly picked out by the system.

Intuitively, these measures characterise how many of the expected facts/items the tool is able to identify in the text in the form of recall, while the precision then gives an indication as to how well the tool is able to discriminate between facts/items that should be extracted versus those that should be ignored. Of course, the exact definitions vary slightly depending on the NLP task being evaluated. These measures combined thus give a good idea as to the performance of a particular NLP tool.

In fact, a common approach to combining these values is to take the harmonic mean to obtain one single value that summarises the performance of an NLP tool; this is known as the $\mathbf{F_1}$**-score**, and is defined as:

$$F_1 = \frac{2 \times precision \times recall}{precision + recall}$$

These are important evaluative measures that shall be referred to throughout the rest of this chapter.

2.1.2 Named Entity Recognition

The CoNLL 2003 task defines named entities as "phrases that contain the names of persons, organizations, locations, times and quantities" [15], although this can be extended to include entities of other types too. Thus, named entity recognition is the task of identifying such named entities in plain text. The CoNLL 2003 task focused on the task of 4 label annotation: Person, Location, Organisation, and Miscellaneous; the latter intended to encapsulate any other entities of particular interest that fall outside the first 3 categories. As a result, the majority of existing NER tools focus on this 4 category task, however, the OntoNotes corpus from 2012 also provides development material that is labelled with 18 different entity tags, listed in Table 2.1. Thus, some available tools also offer a full 18 tag annotation utility, providing additional useful information.

PERSON	People, including fictional
NORP	Nationalities or religious or political groups
FACILITY	Buildings, airports, highways, bridges, etc.
ORGANIZATION	Companies, agencies, institutions, etc.
GPE	Countries, cities, states
LOCATION	Non-GPE locations, mountain ranges, bodies of water
PRODUCT	Vehicles, weapons, foods, etc. (Not services)
EVENT	Named hurricanes, battles, wars, sports events, etc.
WORK OF ART	Titles of books, songs, etc.
LAW	Named documents made into laws
LANGUAGE	Any named language
DATE	Absolute or relative dates or periods
TIME	Times smaller than a day
PERCENT	Percentage (including “%”)
MONEY	Monetary values, including unit
QUANTITY	Measurements, as of weight or distance
ORDINAL	“first”, “second”
CARDINAL	Numerals that do not fall under another type

Table 2.1: The 18 types of entity annotated in the OntoNotes corpus [14].

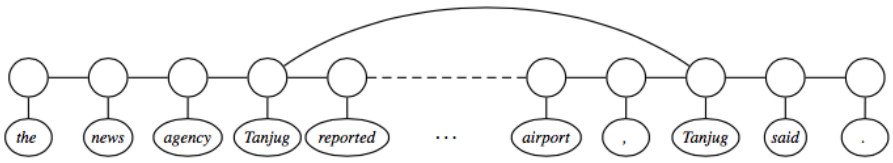


Figure 2.1: An example of the label consistency problem excerpted from a document in the CoNLL 2003 English dataset. Figure from [16].

Named Entity Recognition is an area of particular interest for my project as it provides the means to extract useful information that may be of interest to the user. In particular, identification of people, places, and organisations could prove very useful to help discover any actors and places of significance in a piece of text, and also to incorporate this information into the resulting visualisation of the event timeline. In doing so, we may be able to enhance the insights gained from a visual representation of the events in a piece of text, maximising the benefit of such an application. However, as we will discuss shortly, despite the good performance of current NER tools, there are still many shortcomings to these tools and challenges that remain to be overcome.

Two of the top performing, publicly available NER tools at the time of writing are the **Stanford Named Entity Recognizer**[5], and the **Illinois Named Entity Tagger**[6]. The Stanford NER tool is available as part of the Stanford CoreNLP software[7], a suite of NLP tools that perform both the basic syntactic analysis discussed earlier in addition to a variety of more sophisticated textual analyses. The system available online behaves and performs similarly to the baseline local+viterbi system discussed in the related paper, [16], which achieved an F-score of 85.51% on the CoNLL 2003 task [16]. However, the system available improves on this by also incorporating distributional similarity based features [16]. That is, by considering the additional context of the words surrounding a word of interest, we may be better able to identify the meaning of the word. For example, whether a "bar" is referring to a "chocolate bar" or a "wine bar"[8].

The paper also highlights a lot of the shortcomings of the current solution. Firstly, the discussion of how the inclusion of non-local structure can benefit information extraction tasks, which they demonstrate through the example in Figure 2.1. That is, the additional information from the first mention of Tanjug makes it clear that it is an organisation, while the second reference alone is ambiguous as to whether we should treat Tanjug as a person or an organisation. By using the earlier information we should be able to correctly resolve this. Unfortunately, despite the incorporation of non-local structure improving the results of NER annotation by a further 1.3% on the baseline, it also performed over 30x slower than the baseline. Due to this unacceptable overhead, the technique is not present in the currently available system.

[5] Available from http://nlp.stanford.edu/software/CRF-NER.shtml
[6] Available from https://cogcomp.cs.illinois.edu/page/software_view/NETagger
[7] Available from http://stanfordnlp.github.io/CoreNLP/
[8] An interesting overview of this topic can be found at http://www.timvandecruys.be/media/presentations/pres_cental_051110.pdf

	PER	LOC	ORG	MISC
PER	1941	5	2	3
LOC	0	167	6	63
ORG	22	328	819	191
MISC	14	224	7	365

Table 2.2: Counts of the number of times an entity sequence is labelled differently from an occurrence of a subsequence of it elsewhere in the document. Rows correspond to sequences, and columns to subsequences. Thus, diagonal entries are correctly labelled subsequences. Taken from the CoNLL training set. As reported by [16].

A second difficulty is the handling of mentions that are subsequences of earlier mentions. For example, an article may mention the football club *Sheffield Wednesday*, which should be identified as an organisation. However, later in the article we may see the club referred to more colloquially as "Wednesday". In this context, *Wednesday* should be treated as referring to the same organisation rather than the day of the week, but this is a subtle distinction to make [17]. The confusion matrix of Table 2.2 taken from the same paper quantifies just how often this sort of challenge arises, and shows that there is still a significant margin of error in the current NER tools.

The Illinois Named Entity Tagger is another state-of-the-art NER tool, that in addition to the 4-tag annotations provided by the Stanford NER, can also provide 18-tag annotations as defined by the OntoNotes project. In fact, this tool currently achieves an F-score of 90.8% on the CoNLL 2003 corpus: one of the best reported results to date [17].

A key piece of information to note here is that the Illinois tagger makes use of prior knowledge in the form of Gazetteers. These are essentially look-up lists containing typical examples of a particular entity. For example, the Annie Gazetteer[9] maintains a collection of lists, each associated with a major and minor type. For example, we may have two lists with major type: unit. However, one listing possible units of *money*, and the other listing possible units of *time*. Hence, this distinction is made by the two lists having the minor types *money* and *time*, respectively. This approach was inspired by the fact that the baseline entry for the CoNLL 2003 task essentially performed a dictionary look-up to determine what a word or phrase could be referring to, and immediately achieved a relatively high F-score of 71.91% [18]. The Illinois tagger then builds upon this with a range of other techniques, including the incorporation of non-local structure as we discussed above. The Illinois tagger has been designed to be largely domain agnostic, as the group expect it to be "applied on a diverse set of documents: historical texts, news articles, patent applications, webpages etc." [17], but by relying on the use of such world-knowledge, the system is unlikely to perform well on fictional texts, referring to fictional character names and places as these re-introduce ambiguity to the problem: a consideration we take into account when determining the corpus of text to use for development.

[9] https://gate.ac.uk/sale/tao/splitch13.html

The paper concludes that "NER proves to be a knowledge-intensive task, and it was reassuring to observe that knowledge-driven techniques adapt well across several domains" [17] . In a recent short paper published by Illinois, they report that they have since further enhanced the application by improving its reliability, memory footprint, wall-clock performance. In addition, some further tweaks have increased the F-score of the Illinois tagger to 91.06% on the CoNLL 2003 task [19]. Thus, this appears to be the most promising tool in the area of NER at this moment in time.

2.1.3 Coreference Resolution

Coreference resolution is the task of identifying all expressions in a piece of text that refer to the same entity, typically clustering the results to form a *coreference chain*: a collection of all the phrases from the text that refer to a particular entity [20]. As a concrete example, consider the text below where all the phrases coreferent to the entity named Jonathan are highlighted in bold:

> ***Jonathan*** *went over to* ***his*** *car.* ***He*** *reversed out of the driveway, and went to work.*

This task plays a significant role in a number of higher-level NLP tasks, including text-summarisation, question-answering, and information extraction [20]. It is also of particular relevance to my task of event extraction in order to identify the chain of events that involve a particular actor of interest, and to then be able to use this information to construct a timeline of the form shown back in Figure 1.1. While at first glance this is perhaps a seemingly simple task, there are a great number of subtleties to be considered, and as a result even the current state-of-the-art systems only achieve an accuracy of around 60%, as we'll see below.

Like a lot of NLP problems, before the advent of a large corpora of annotated material for development was made available, it was extremely difficult to apply machine learning to these problems. However, with the creation of the OntoNotes corpus, "a large-scale, accurate multilingual corpus for general anaphoric coreference that covers entities and events not limited to noun phrases or a limited set of entity types" [21], the CoNLL 2011 and 2012 tasks challenged participants to model "Unrestricted Coreference in OntoNotes" [21]. As a result, we have seen the performance of the state-of-the-art coreference resolution systems improve significantly over the past few years. It is also worth noting that the CoNLL 2012 task focuses primarily on intra-document coreference resolution: the resolution of co-references within a single document, while more recent efforts have gone on to explore the task of inter-document co-reference resolution: identifying coreferent mentions of a given entity across a collection of documents. It is intra-document coreference resolution systems that are of greatest interest to us at present.

Type of Mention	*Example*
Named mentions	"Adam"
Nominal mentions	"the dogs", "the little old lady"
Pronominal mentions	"she", "they"
Identical reference	"She had **a good suggestion** and **it** was unanimously accepted", where the entities in **bold** are co-referent [22].

Table 2.3: Typical categorisation of mentions.

Table 2.3 shows the typical categorisation of mentions targeted by coreference resolution tools. For my purposes, it is the former 3 categories that are of particular interest in order to identify exactly which actors are involved in which events within a given text, and thus be able to reflect that in the resulting timeline. However, the latter category may also be of use in determining whether or not multiple sentences are coreferent, possibly linked by entities other than people.

Before evaluating some of the current state-of-the-art, it is worth pointing out that along with the original MUC-defined metrics shown earlier, a number of refinements to these original definitions have also since been proposed to address some of their shortcomings. In particular, the two most commonly used metrics alongside the original MUC metric that is still used for comparative reasons, are the B^3 metrics and the CEAF metrics. These are subtle variants of the original definitions of **recall** and **precision** from the MUC. While the MUC metric only evaluates the accuracy of any coreference *chains* containing more than 1 mention of an entity and treats all mistakes as equals, the B^3 metric is designed to also evaluate the correctness of any singleton mentions: mentions that are not resolved to a chain of mentions, and also considers that some mistakes are more costly than others when it comes to coreference resolution [23]. For example, merging two long chains that refer to different entities is potentially more costly than adding one additional incorrect mention to a large chain.

The CEAF metric then proposes an altered method of aligning the mentions identified by the system under evaluation with the true mentions expected, ensuring that at most one of the system-produced mentions aligns to any one of the expected true mentions [23]. As a result, many conferences now employ all 3 of these metrics in evaluation to provide a more rounded result and for the purpose comparison. For more information, I refer you to [23].

Many of the existing solutions to the coreference resolution task operate by learning a scoring function over individual mention pairs, considering only 2 phrases at any one time. This scoring function is then used to determine whether any two mentions should be treated as co-referential [24]. Some more recent approaches have then gone a step further by incorporating additional more global

information that can be extracted from the text to improve the analysis. Both Stanford[10] and Illinois[11] universities have publicly available coreference resolution systems that attain a performance to match and even surpass the current best in class, and so we explore these tools in particular in greater detail.

The Illinois system incorporates "a novel and principled machine learning framework that pushes the state-of-the-art while operating at a mention-pair granularity", thus remaining at the level of comparing individual mentions [25]. They present a linguistically inspired Latent Left Linking model (L^3M), based largely upon the Best-Left-Link approach that has yielded competitive results in the past [26, 27]. The model employs a latent structural SVM approach to learn to recognise the most likely antecedent to the immediate left of a mention, mimicking the process a reader would typically go through when determining who a mention may refer to, subsequently constructing the resulting coreference chains through the transitive relationship between left co-references [25]. Furthermore, with the incorporation of some additional knowledge-based constraints in order to inject some more explicit world-knowledge into the system, the Illinois coreference tagger manages to achieve state-of-the-art results on the OntoNotes-5.0 corpus; this extended model is referred to as the CL^3M model. These results are shown in Figure 2.2. These results show the Illinois system to outperform the Stanford rule-based coreference system, and also highlights the additional performance gained from incorporating some additional rule-based constraints. This leads to consider a perhaps obvious but important point: while machine-learning methods can enhance the performance of coreference resolvers, it also makes the resolvers highly dependent on the training data used. With insufficient, inadequate, or over-specific training data the system may not learn the set of rules we expect it to, and may lack the ability to generalise to other texts. This is a trait that rule-based methods do not suffer from, and is something we must bear in mind when incorporating these tools into our solution.

Since this paper was published in 2013, Stanford have gone on to further improve their coreference systems and extend the performance of the current state-of-the-art. The Stanford co-reference resolution system is capable of resolving all 3 of the aforementioned mention types, and offers 3 different approaches to the problem: a deterministic rule based system, a statistical system, and a neural system [28]. The performance of these systems over the CoNLL 2012 evaluation data are summarised in Figure 2.3, where we are at present only interested in the performance on English data. Note however, the figures quoted in this table are the performance of their models trained for general purpose use rather than maximal performance on this data set. Again, highlighting the significance of a good training set.

The deterministic rule-based system was submitted to the CoNLL 2011 shared task and attained the top result of of 58.3% in the open-track challenge [29]. The open-track allowed the use of external resources, such as Wikipedia and WordNet in order to enhance the performance of applications, acknowledging the role of world knowledge in recognising entities within a text [29]. The system employs a multi-pass sieve system to identify and resolve coreferences in a two stage process. Each *sieve* acts as a filter, searching for specific items in the text that satisfy its constraints.

[10] Available from http://nlp.stanford.edu/projects/coref.shtml

[11] Available from https://cogcomp.cs.illinois.edu/page/software_view/Coref

	MUC	BCUB	$CEAF_e$	AVG
	Dev Set			
Stanford	64.30	70.46	46.35	60.37
(Chang et al., 2012a)	65.75	70.25	45.30	60.43
(Martschat et al., 2012)	66.76	71.91	47.52	62.06
(Björkelund and Farkas,)	67.12	71.18	46.84	61.71
(Chen and Ng, 2012)	66.4	71.8	**48.8**	62.3
(Fernandes et al., 2012)	**69.46**	71.93	48.66	63.35
L^3M	67.88	71.88	47.16	62.30
CL^3M	69.20	**72.89**	48.67	**63.59**
	Test Set			
Stanford	63.83	68.52	45.36	59.23
(Chang et al., 2012a)	66.38	69.34	44.81	60.18
(Martschat et al., 2012)	66.97	70.36	46.60	61.31
(Björkelund and Farkas,)	67.58	70.26	45.87	61.24
(Chen and Ng, 2012)	63.7	69.0	46.4	59.7
(Fernandes et al., 2012)	**70.51**	71.24	**48.37**	**63.37**
L^3M	68.31	70.81	46.73	61.95
CL^3M	69.64	**71.93**	48.32	63.30

Figure 2.2: F_1-scores of various coreference resolution systems on the OntoNotes-5.0 corpus. Showing all three of the MUC, BCUB, and CEAF scores. The L^3M and CL^3M are the Illinois implementations [25].

SYSTEM	LANGUAGE	PREPROCESSING TIME	COREF TIME	TOTAL TIME	F1 SCORE
Deterministic	English	3.87s	0.11s	3.98s	49.5
Statistical	English	0.48s	1.23s	1.71s	56.2
Neural	English	3.22s	4.96s	8.18s	60.0
Deterministic	Chinese	0.39s	0.16s	0.55s	47.5
Neural	Chinese	0.42s	7.02s	7.44s	53.9

Figure 2.3: Performance of the Stanford Coreference Resolution system on the CoNLL 2012 evaluation data. Figures are lower than in the related papers as the systems evaluated here have been trained for general purpose use, rather than to necessarily achieve the maximum score on the OntoNotes corpus [28].

The first stage, mention detection, aims to maximise **recall**, applying sieves in the order of highest recall to lowest recall in order to identify as many candidate mentions as possible for the second stage [29]. Missing any candidates at this stage automatically rules them out for inclusion in the next stage: coreference resolution. In contrast to the mention detection stage, this stage now priorities **precision**, again applying a number of different *sieves* from highest precision to lowest precision, ensuring that the easiest merges (i.e. an exact noun match) are made before the more challenging ones [29].

Unlike many other existing systems, including the Illinois system, that restricts evaluation of coreference to mention-pairs, the Stanford statistical co-reference resolution system goes a step further by incorporating *entity*-level information into its analysis [24]. The system proceeds in 2 stages, each presenting a novel contribution to the area of coreference resolution. Firstly, the authors make use of **model stacking**, allowing them to apply 2 distinct coreference models to the problem. The first considers the global context, identifying all possible antecedents of a mention in the text, while the other focusses on the more immediate problem of identifying the most likely nearby antecedent, similar to the Left Linking model of the Illinois system. Each model provides slightly different information that when combined complement one another to produce a far more accurate outcome [24]. Having constructed an initial set of co-reference clusters, a post-processing stage performs the pairwise comparison of these clusters, potentially merging clusters using an "entity-centric model that operates between pairs of clusters instead of pairs of mentions, guided by scores produced by the pairwise models" [24]. The result is a set of clusters that each represent the mentions of an individual and distinct entity.

The Stanford statistical system achieves an F_1 score of 60.3% on the CoNLL 2012 evaluation data. A result to match, and even surpass, the current state-of-the-art on this data, and illustrating how taking an iterative approach, utilising the results of early coreference decisions to inform later ones, can have a significant impact on the final results [24]. A useful example from the Stanford paper is that of supposing we had a mention cluster, {"Clinton", "She"}, that has been identified as co-referential. It is thus more likely that these two mentions are also co-referential with a second cluster containing "Hillary Clinton" than that containing "Bill Clinton", also found in the same text. The entity-centric model employed by the Stanford system would indeed merge the mention cluster of "Clinton" with that of "Hilary Clinton", as the additional constraint imposed by the "she" co-reference makes "Bill Clinton" an unlikely co-referent [24]. In contrast, other systems that merge clusters based only on mention pairs, may just as likely merge the cluster containing "Clinton" with the mention of "Bill Clinton" despite the additional constraint imposed by "She".

Antecedent	Anaphor
the country's leftist rebels	the guerrillas
the company	the New York firm
the suicide bombing	the attack
the gun	the rifle
the U.S. carrier	the ship

Figure 2.4: Example nominal coreferences that the Stanford neural coreference resolution system resolves, but the statistical implementation misses. From [30].

Finally, the neural system builds upon the entity-centric approach of the statistical system above introducing the use of neural-networks for encoding mention-pairs and cluster-pairs into "high-dimensional vector representations" [30], and then merging mention clusters using an *easy-first cluster-ranking* procedure [30]. The increased complexity of this implementation is certainly apparent in the increased time taken to perform coreference resolution in the example of Figure 2.3. In this case, the authors train a deep neural network to rank possible cluster merges using a learning-to-search algorithm [30]. The network then ranks possible cluster merges and proceeds in an easy-first manner [30]. This refers to the approach of merging the most likely coreference clusters first, yielding the benefit that earlier decisions can then be used to inform later, more difficult ones. This is in contrast to the common approach to merging clusters using a left-to-right strategy, building up clusters as we pass through the text.

As a result, the neural system significantly improves on the state-of-the-art with an overall F_1-score of 65.29%. A particularly useful area of improvement for our purposes in the resolution of *nominal* mentions, which they credit to their incorporation of semantic information. In this particular context, they see an improvement from 10.7% to 18.9% F_1 over the statistical implementation; still extremely low but a significant improvement, and one that could prove particularly important in the context of fairy tales and other narratives where we often have characters such as *the wolf* or *the little old man*. Figure 2.4 from [30] highlights a number of nominal coreferences which the neural system resolves correctly but the statistical system misses.

Our brief exploration of the area of coreference above highlights just a few of the complexities of coreference, and as a result the relatively poor performance of coreference resolution systems in contrast to other areas of NLP, with the current state-of-the-art achieving at best F_1 scores of 65.29%, leaving a lot to be desired. This inaccuracy will have to be accounted for in our resulting application, both in ensuring that we use a number of different features beyond coreference resolution for event clustering, and give the user the ability to correct any mistakes made by the application. In particular, we see an even lower accuracy in the case of resolving coreferent nominal mentions, not only highlighting the difficulties this type of reference but also perhaps the reliance of machine-learnt solutions on the training data used. According to [30], only about 1.2% of the positive links in the CoNLL test set is of this form, suggesting little emphasis has been put on this class of coreference as yet.

Both the Illinois and Stanford systems have online demos available, so I have also performed some of my own, less scientific, experimentation on the two systems in order to judge which solution may be better suited to my problem. However, it is worth noting that I aim to construct the system in such a way as to make it relatively easy to migrate between different systems at a later date should improved systems become available.

Figure 2.5 shows the results of executing the Illinois co-reference resolution system over a provided example, showing the system is quite capable of handling large texts containing multiple distinct entities. The system initially appears to perform well on this task, with the mention graph showing that it correctly resolves all mentions of the *bombsniffing dogs*. We also see an example **identity reference** between "it" and "The Transportation Security Administration".

However, the results on a far simpler, and better representative example of the sort of text I wish to process reveals the limitations of the system. In Figure 2.6, we can see that the system fails to recognise "John" and "a farmer" as coreferent entities. Thus resulting in a dis-join that makes it difficult to relate all co-references to the farmer back to "John", as desired. We also see that the entities "Rusky" and "a pet dog Rusky" are not correctly resolved by the system either. Thus, to use this system I would have to introduce an additional level of post-processing to attempt to correctly merge these coreference clusters, perhaps using some additional information obtained from semantic role labelling, a topic that will be discussed later, or other analyses.

The Stanford coreference system on the other hand, shown in Figure 2.7, fares far better on this simple example, correctly identifying all references to John, and all references to Rusky. Caroline is also recognised as an entity, but is not included in the coreference clusters shown in the figure as she is only a singleton mention[12] in this text. Thus, there is no coreference chain for Caroline, but she is certainly identified as an entity in the text. These results seem to illustrate the clear benefit of the entity-centric model used to merge coreference clusters in the Stanford system.

Thus, these initial experiments appear to support the evidence that the Stanford system is at present the superior tool. A deeper comparison between the alternative implementations of the Stanford coreference resolver and the adjustable parameters is provided later in Section 3.3.

[12] A singleton mention is an entity that is only mentioned once in the text

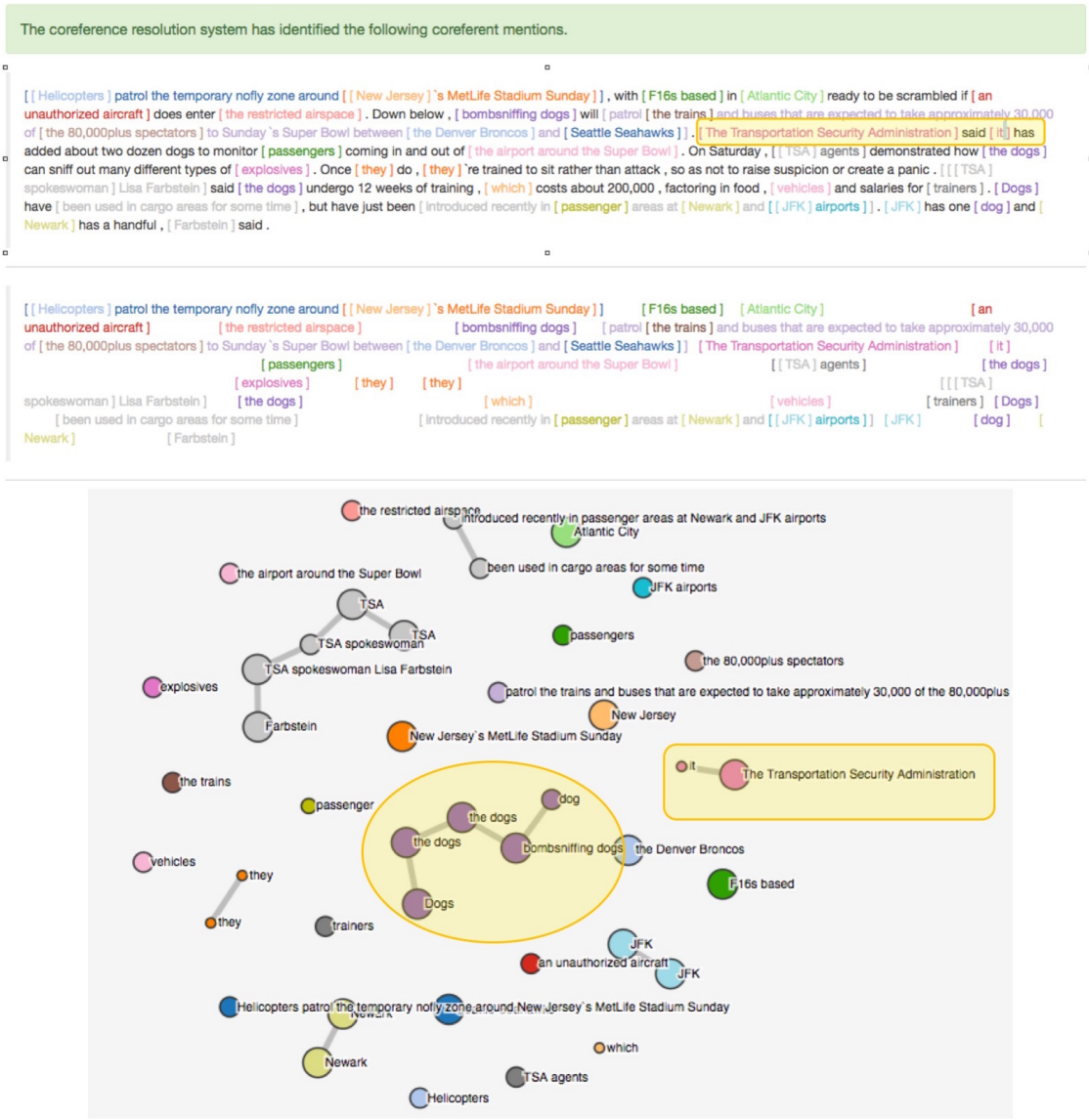


Figure 2.5: The results of executing the Illinois Coreference Resolution Demo on a large input.

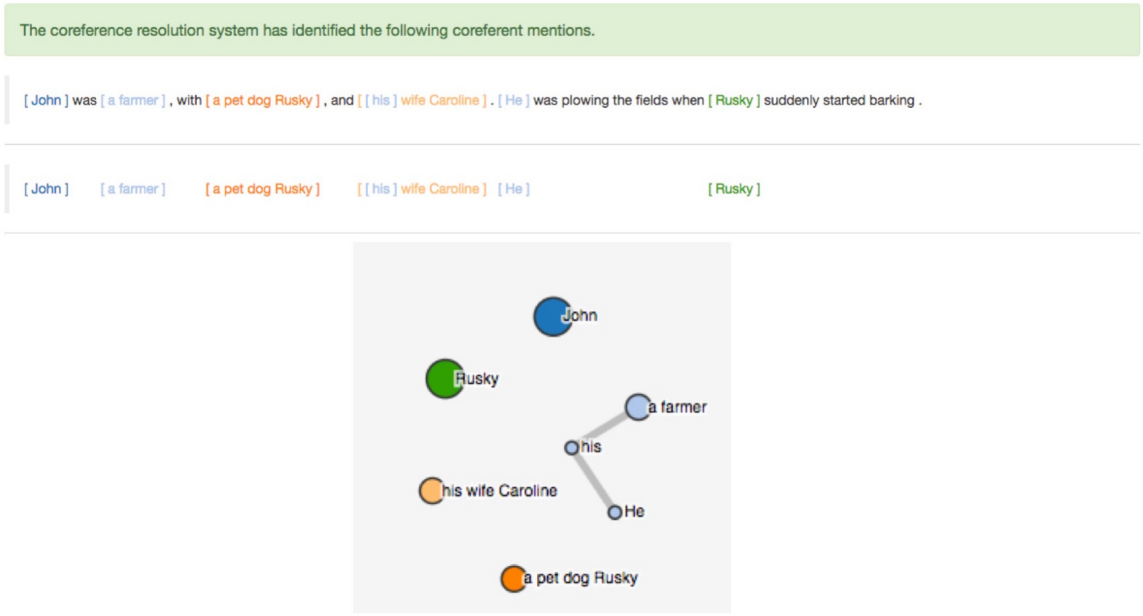


Figure 2.6: The results of executing the Illinois Co-reference Resolution Demo over a shorter, more representative sample of input text. The mention graph below illustrates the coreference clusters identified, and shows that it fails to link "John" with references to the "farmer", and fails to link the two mentions of "Rusky" as a single entity.

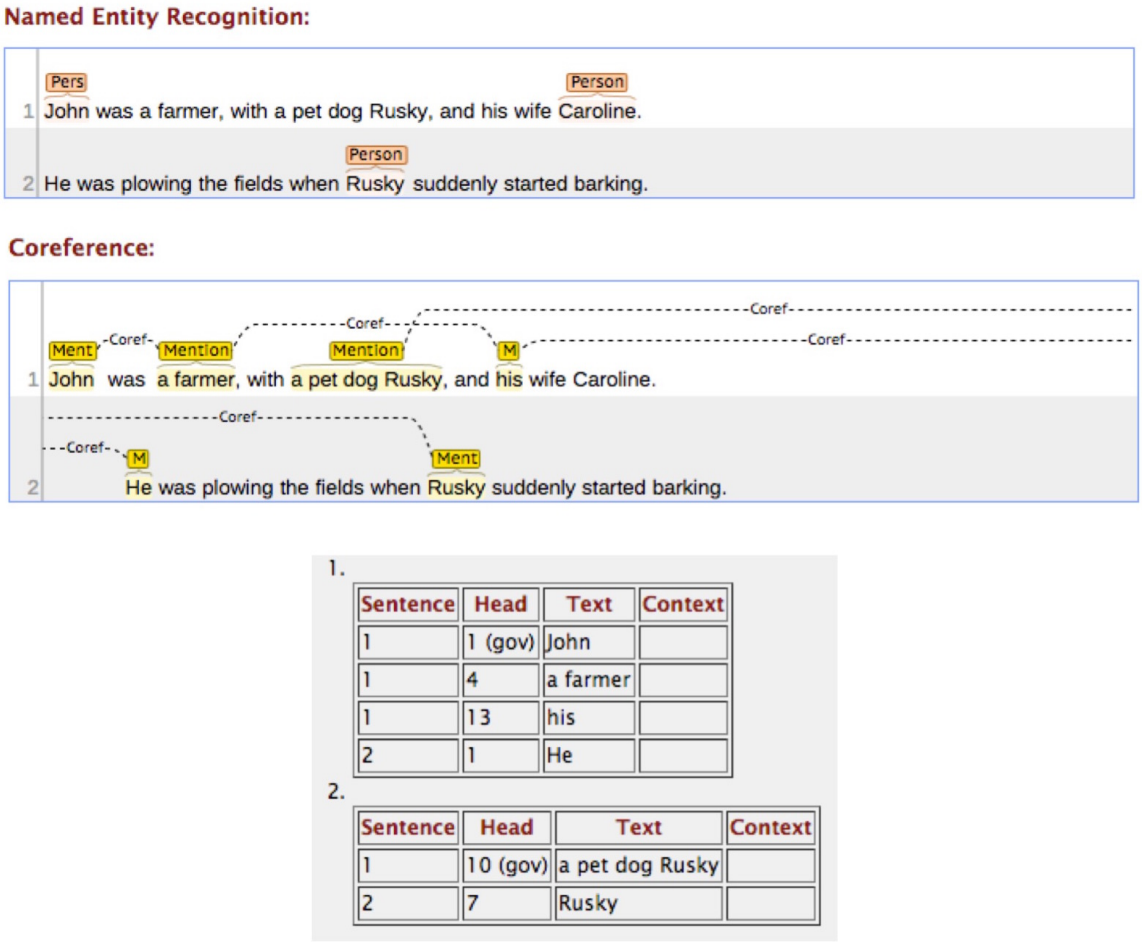


Figure 2.7: The results of executing the Stanford Coreference Resolution System as part of the Stanford CoreNLP demo over the same simple input text. We clearly see that the Stanford NER system correctly identifies the three actors in this text, and that the co-reference resolution system correctly identifies two coreference chains: one for “John”, and one for “Rusky”.

2.1.4 Temporal Information Retrieval

Temporal information retrieval is the non-trivial task of identifying temporal expressions and recognising temporal relations between times and events in text. A perhaps obvious, but intriguing nonetheless, thought by Derczynski in his book *Automatically Ordering Events and Times in Text* is the fact "that *we* can identify the nature of temporal relations easily suggests that the information required for temporal relation extraction is contained either in discourse or in world knowledge" [31]. Thus, making this a certainly achievable, if difficult, challenge. The task of temporal relation extraction can be broken down into two parts:

1. Identifying temporal entities, such as times and events, in text
2. Determining how these entities relate to one another

However, it appears that thus far, work towards extracting *events* from text, and work towards extracting *temporal expressions* from text have been largely disjoint efforts [31]. As a result, a rather damning summary of progress in the temporal information retrieval area was given by Mirza in 2014, stating that "recalling the low performances of currently available systems on the temporal relation extraction task, including the state-of-the-art systems according to TempEval-3, it is still insufficient to use the existing temporal relation extraction systems to support real world applications, such as creating event timelines and temporally-based question answering" [6].

Despite this, the increasing amount of annotated text corpora such as the TimeBank 1.2 corpus[13], a corpus consisting of 182 news articles with annotated temporal expressions, has enabled the application of machine learning to the problem; a technique that has proven extremely successful in other areas of NLP.

Temporal Information Extraction is of particular importance to my task, as identifying the time of occurrence of events, and the temporal relations between them, forms a fundamental part of constructing an event timeline from text. Table 2.4 defines the 3 distinct types of temporal expression that each present a unique challenge to T-IR[14], and are important to recognise so that we can properly test and evaluate both existing T-IR systems, as well as my application.

To reduce some degree of ambiguity, we reduce our focus to texts in the English language, better enabling us to resolve phrases such as "New Year". In addition, a further consideration to take into account is the modality of temporal phrases. That is model expressions such as "might" or "could" reduce the likelihood of an event actually happening. As such, this should perhaps be reflected in the resulting visualisation, or such events could be discarded altogether. This is a consideration that I will explore further throughout the project.

[13] http://www.timeml.org/timebank/timebank.html
[14] Temporal Information Retrieval

Type of temporal expression	**Examples**	**Notes**
Explicit temporal expressions	"September 2017", "24/02/2017"	The easiest to recognise and resolve (despite some ambiguity in the ordering of month and date)
Implicit temporal expressions	“The New Year”	Difficult to resolve due to “lack of temporal target or an unambiguously associated time point” [32]. i.e. "New Year" has different meanings in China and the USA.
Relative temporal expressions	“on Thursday”	This sort of expression may be difficult to resolve as to whether it refers to “last Thursday” or “next Thursday” relative to some reference date. This may be an inference that requires some additional context to be able to make.

Table 2.4: A common breakdown of the 3 distinct types of temporal expression in text, each presenting a different challenge to temporal information retrieval.

At the present moment, it appears that there are two superior tools for the task of temporal information retrieval: Stanford’s **SUTime**[15] and Heidelberg University’s **HeidelTime**[16]. Both of these tools achieved competitive results on the TempEval-3 challenge, a “three-part task structure covering event, temporal expression and temporal relation extraction” [33], with SUTime and HeidelTime achieving F_1 scores of 93.2% and 93.0% respectively in the extraction task[17], and scores of 67.4% and 77.6% for the normalisation process[18] [32].

SUTime, available as part of Stanford’s CoreNLP package, is a “deterministic rule-based system designed for extensibility” [33]. In fact, both SUTime and HeidelTime are rule-based systems, with [34] giving 4 key reasons for taking this approach:

> *“(1) the divergence of temporal expressions is very limited compared to other named entity recognition and normalization tasks, e.g., the number of persons and organizations as well as the variety of names referring to these entities are probably infinite, (2) the normalization is hardly solvable without using rules, (3) resources for additional languages can be added without the need of an annotated corpus, and (4) the knowledge base can be extended in a modular way, e.g., for adding events and their temporal information such as "soccer world cup final 2010" that took place on July 11, 2010”*

[15] Available from http://nlp.stanford.edu/software/sutime.shtml

[16] Available from https://github.com/HeidelTime/heideltime

[17] The task of identifying and extracting temporal expressions from text

[18] Normalisation is the process of transforming an expression such as “1st March 2012” into a standard format such as “2012-03-01”

Expression	Type	Value
October of 1963	DATE	1963-10
October	DATE	2011-10
last Friday	DATE	2011-09-16
next weekend	DATE	2011-W39-WE
the day after tomorrow	DATE	2011-09-21
the nineties	DATE	199X
winter of 2000	DATE	2000-WI

Table 2.5: Example expressions recognised by SUTime, their TIMEX3 type, and their corresponding normalised value. Using a reference date of **2011-09-19**. Taken from [35].

The additional extensibility of SUTime could prove valuable in adapting the system to different domains, such as fairy tales where temporal expressions such as "happily ever after" and "Once upon a time" could also be identified. The Stanford system also attempts to resolve the ambiguities of relative expressions such as "on Thursday" by examining the verb tense of surrounding context, helping to determine whether the phrase refers to the past or the future [35]. Table 2.5 shows some example phrases the SUTime system can recognise, the TIMEX3 type, and the corresponding normalised value.

As the final step in SUTime's execution, it filters the potential temporal expressions found, removing any unlikely candidates from the results. For example, by examining the whether the word "fall" is used as a verb or a noun allows the system to determine whether it's likely referring to the action of falling, or the season of Autumn [35]. However, despite SUTime's good performance in general, it is also important to maintain an awareness as to the current limitations of the system; in particular, "support for temporal ranges is poor. For instance, the expression from 3 to 4 p.m. is incorrectly identified as 15:57:00, while the expression 12-13 March 2011 is identified just as 2011-03" [35]. In addition, "SUTIME cannot correctly interpret temporal expressions such as a year and a half ago"[35], another current limitation of the system.

The HeidelTime system is a publicly available, open source project that appears to be continuously improved. Due to the nature of the current corpora of available annotated material, the majority of existing temporal taggers, such as SUTime, are tuned to analysing news articles. However, HeidelTime has the advantage of being "adapted not only to the news domain but also to narrative documents" [32], a trait that could be of particular benefit to my application. In fact, HeidelTime distinguishes between news-style documents and narrative-style documents (e.g., Wikipedia articles) in all languages, and supports both colloquial English and scientific language [36]. Figure 2.8 shows an example execution of the HeidelTime online demo on a simple example.

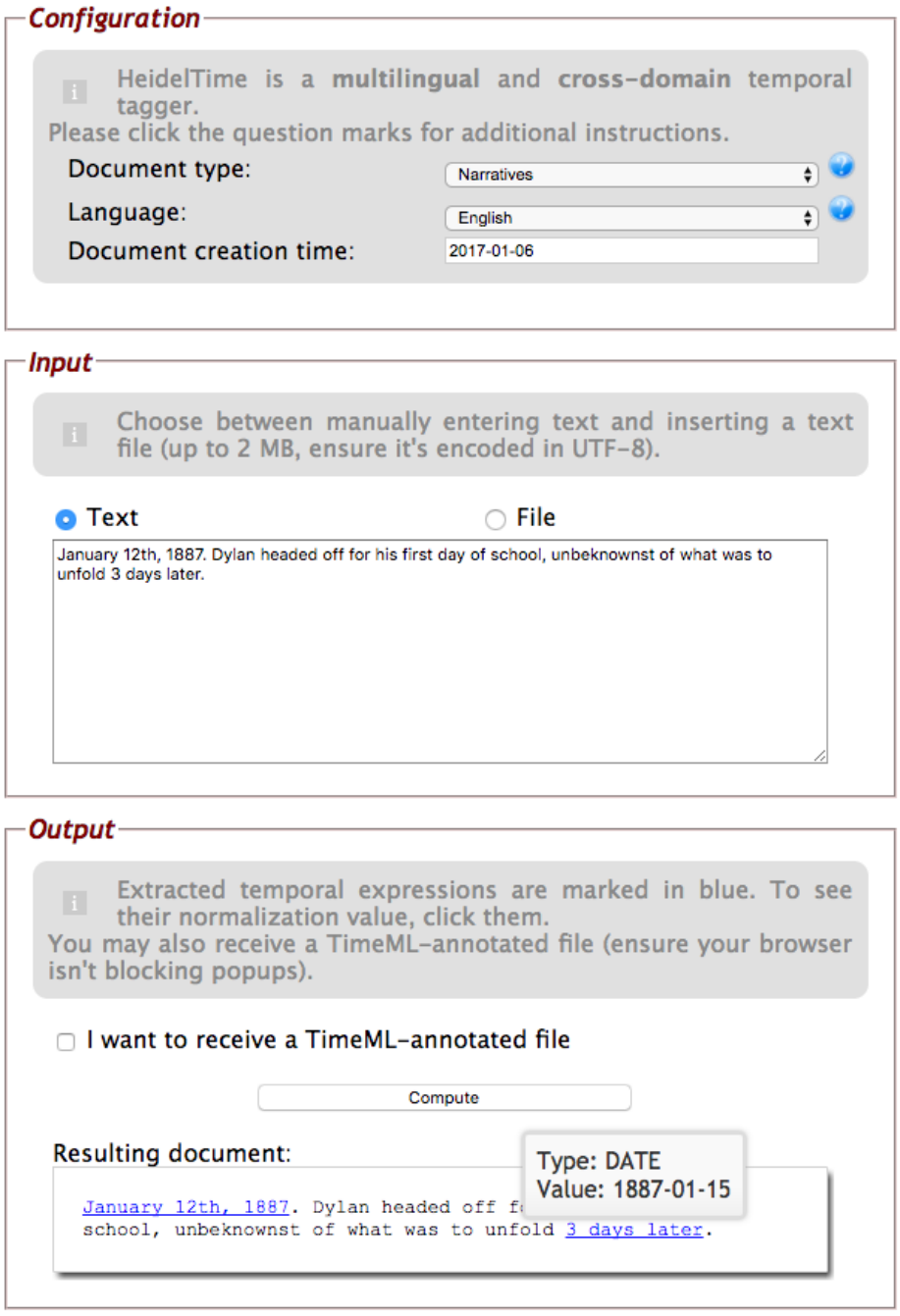


Figure 2.8: Executing the HeidelTime online demo on a simple example. Here we can see the phrase "3 days later" is correctly resolved to "1887-01-15", relative to the earlier date in the text.

The most recent extension to the HeidelTime tool is the 2016 addition of temponym tagging, where temponyms are defined in [37] as:

> *"Free-text temporal expressions [that] refer to arbitrary kinds of named events or facts with temporal scopes that are merely given by a text phrase but have unique interpretations given the context and background knowledge."*

Thus, HeidelTime is now able to resolve expressions such as the "FIFA World Cup Final 1998" to "1998-07-12". In fact, the authors "added to HeidelTime's English resources temponyms for more than 40,000 events and for more than 900,000 facts with several paraphrases" [38], currently consisting of "facts about persons (birth, death, political position, marriage; 664,000) and culture (releases, directions, authors; 253,000)" [38], as well as the identification of sporting events and historical battles [38]. So, this is another feature that could prove particularly useful to the creation of event timelines for historical texts, or other non-fictional narratives that refer to famous historical events.

It is also worth noting the **Illinois Shallow Temporal Reasoning System**, an extension of the HeidelTime system from 2012 that extends the system in 3 ways:

1. The system extends the extraction task to "capture complex temporal expressions" [39], a novel feature that is able to identify phrases of the form "**since** it entered production in February 1947", normalising the expression to the interval [1947-02-01 00:00:00, 1947-03-20 23:59:59] when given a reference time[19] of March 20th 1947.
2. The system normalises temporal expressions to time intervals as opposed to the time points
3. The system is able to perform temporal comparison with respect to multiple relations such as *before, before-and-overlap, contains, equals, inside, after and after-and-overlap*.

This is achieved by combining the results of HeidelTime's annotations with the results of POS[20] tagging. However, it is again worth noting some of the limitations of the system. Firstly, the current set of complex temporal expressions that can be extracted are restricted to the temporal connectives: *since, between, from, before and after* [39]. Secondly, it is likely that HeidelTime has been further improved since the development of this software. As such, it may well be the case that HeidelTime provides the better solution at this moment in time, however a deeper practical evaluation of all 3 tools shall be required prior to the incorporation of one of these into my application. Figure 2.9 illustrates the results of the Illinois online demo over a short example *complex temporal expression*. Figures 2.8 and 2.9 illustrate the clear difference between the single time *point* value provided by HeidelTime system versus the time *interval* provided by the Illinois system, where a time *interval* representation could offer a clear benefit to a timeline visualisation of events.

[20] Part-of-speech

Extracted Expression	Normalized Interval
Since June 2015	2015-06-01T00:00:00.000/+INF

Figure 2.9: Results of the Illinois Temporal Tagger demo correctly extracting the time interval corresponding to the complex temporal expression: "Since June 2015".

2.1.5 Semantic Role Labelling

Semantic role labelling (SRL) is the task of identifying and assigning roles to the constituents of a sentence, typically the arguments of a predicate in the text. As a result, aiming to extract the information required to recognise “"Who did what to whom" (and perhaps also “when and where”)” [40]. These roles can be very specific, such as BUYER, or far more abstract, such as AGENT. For example, in the sentence “XYZ corporation bought the stock”, we could extract the tuple *bought(XYZ corporation, the stock)*, where *XYZ corporation* is identified as the BUYER, and *the stock* as the ITEM_BOUGHT [40]. The key power behind semantic role labelling is that we can extract the same predicate-argument information regardless of how it is expressed. Some alternative expressions of the previous example are given below:

- *XYZ corporation bought the stock*
- *The stock was bought by XYZ corporation*
- *They sold the stock to XYZ corporation*

Figure 2.10 shows a concrete example of this using the Illinois Semantic Role Labelling Demo[21].

Figure 2.10: The results of the Illinois Semantic Role Labelling Demo on two different representations of the same information.

[21] Available from http://cogcomp.cs.illinois.edu/page/demo_view/srl

Thematic Role	Definition
AGENT	The volitional causer of an event
EXPERIENCER	The experiencer of an event
FORCE	The non-volitional causer of the event
THEME	The participant most directly affected by an event
RESULT	The end product of an event
CONTENT	The proposition or content of a propositional event
INSTRUMENT	An instrument used in an event
BENEFICIARY	The beneficiary of an event
SOURCE	The origin of the object of a transfer event
GOAL	The destination of an object of a transfer event

Figure 2.11: Some commonly used roles along with their definitions. Taken from [40].

Thematic Role	Example
AGENT	*The waiter* spilled the soup.
EXPERIENCER	*John* has a headache.
FORCE	*The wind* blows debris from the mall into our yards.
THEME	Only after Benjamin Franklin broke *the ice...*
RESULT	The city built a *regulation-size baseball diamond...*
CONTENT	Mona asked *"You met Mary Ann at a supermarket?"*
INSTRUMENT	He poached catfish, stunning them *with a shocking device...*
BENEFICIARY	Whenever Ann Callahan makes hotel reservations *for her boss...*
SOURCE	I flew in *from Boston.*
GOAL	I drove *to Portland.*

Figure 2.12: Some commonly used roles with examples. Taken from [40].

Figure 2.11 defines some of the roles commonly assigned in a piece of text, while Figure 2.12 provides some more concrete examples of such roles. Such additional semantic information could prove useful in identifying further information of interest for display in the resulting timeline. For example, highlighting any *instruments* or *beneficiaries* involved in an event could prove useful in the context of witness accounts or historical texts, allowing us to draw attention to certain artefacts.

The **Illinois Semantic Role Labeller** appears to be among the current state-of-the-art, achieving the highest F_1 score of 77.92% on the CoNLL 2005 shared task[22] [41]. Figure 2.13 shows an example annotation from the online demo of the system, where we can see that the system correctly identifies "A squirrel" as the *storer* of "a lot of nuts" as it prepares for a "seasonal change". However, my own experiments with the system executing locally on my machine reveal the Illinois SRL tool to be an extremely resource intensive and time consuming process, requiring 4GB of RAM for verb SRL and a further 1GB of RAM for noun SRL[23].

[22] `http://www.cs.upc.edu/~srlconll/st05/st05.html`

[23] All system requirements of the Illinois NLP Pipeline, which includes the semantic role labelling tool, can be found at `https://github.com/CogComp/cogcomp-nlp/blob/master/pipeline/README.md`

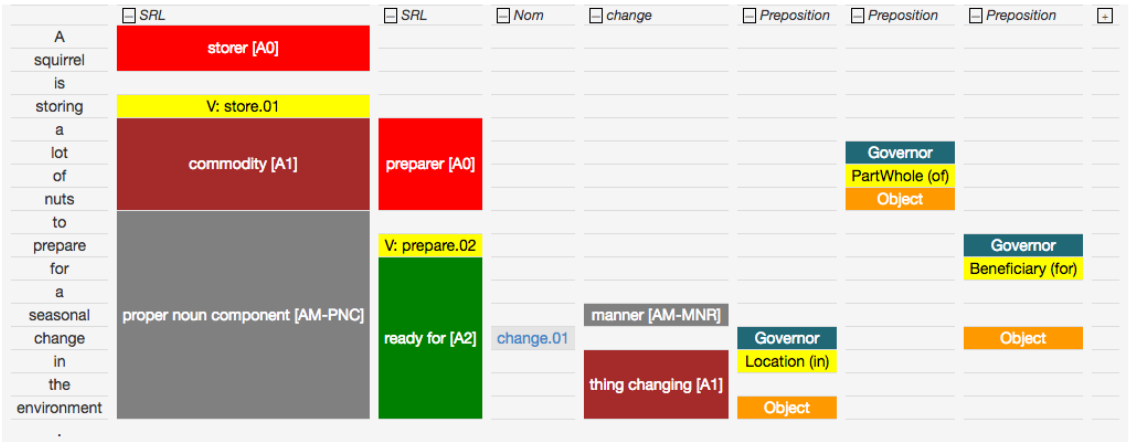


Figure 2.13: Demo of the Illinois Semantic Role Labelling Demo

Evaluating the results, I see little reason to incorporate this tool into my application at present, as it provides little value to the application initially. We do not necessarily require natural language *understanding* to construct a timeline, nor do we require information retrieval in the question-answering sense. As a result, the semantics of the text we're processing could be deemed largely irrelevant to the construction of an event timeline. Our task is the identification of *events* that happen at different moments in time, who's involved, when, and possibly where. All issues that can be solved with NLP tools related to other NLP tasks. However, semantic role labelling does provide interesting supplementary information that could be used to enhance the visualisation of the resulting timeline, and provides the means to later incorporate semantic information into our event extraction process; as such, I shall keep it in mind.

2.1.6 Event Extraction

Event extraction is another developing area of NLP that lies at the heart of this project. Current solutions to this problem typically consist of a combination of all the NLP tools and techniques we have already discussed in addition to machine learning techniques, using the information extracted by previous techniques as features to inform further inference. The results of extraction also largely depend upon how we define an *event*. Two common event models being the TimeML description [5], focussing on things that *happen or occur* or predicates describing *states and circumstances* that may hold true at a given point in time, and the definitions of the ACE 2005 event extraction task, which identifies the eight types of event shown in Table 2.6 [42].

The TempEval-2 task has proven to be one of the most significant tasks in semantic event extraction, helping push the state-of-the-art by presenting three challenges "that are relevant to understanding the temporal structure of a text: (i) identification of events, (ii) identification of time expressions and (iii) identification of temporal relations" [43].

Event Type	Event Sub-types
Life	Be-Born, Marry, Divorce, Injure, Die
Movement	Transport
Transaction	Transfer-Ownership, transfer-Money
Business	Start-Org, Merge-Org, Declare-bankruptcy, End-Org
Conflict	Attack, Demonstrate
Contact	Meet, Phone-Write
Personnel	Start-Position, End-Position, Nominate, Elect
Justice	Arrest-Jail, Release-Parole, Trial-Hearing, Charge-Indict, Sue, Convict, Sentence, Fine, Execute, Extradite, Acquit, Appeal, Pardon

Table 2.6: The eight distinct types of event of interest as defined by the ACE 2005 task on event extraction.

It goes without saying that a precursor to the visualisation of text as an event timeline is the effective extraction of *events* from a given text. As such, this is an especially salient task in the development of an application for the automatic creation of infographic event timelines from text. However, it appears that at present there are very few publicly available event extraction tools. As such, this forms a significant part of the project, requiring the development of a custom event extractor using the current best performing techniques from previous works and perhaps some additional novel contributions.

Stanford OpenIE[24] is a publicly available information extraction tool, designed to extract relation tuples from plain text [44]. It is certainly not designed for the specific task of event extraction, but could provide a source of useful information. For example, from the sentence *"Born in Honolulu, Hawaii, Obama is a US Citizen."*, the system is able to extract the tuples (Obama; is; US Citizen) and (Obama; born in; Honolulu, Hawaii). These are certainly useful facts, and explains how the system "outperforms a state-of-the-art open IE system on the end-to-end TAC-KBP 2013 Slot Filling task" [45], however this information definitely appears more relevant to the problem of question-answering than that of event extraction; useful facts but not necessarily meeting the TimeML and ACE definition of an event as "something that happens/occurs or a state that holds true, which can be expressed by a verb, a noun, an adjective, as well as a nominalization either from verbs or adjectives" [6].

[24] Available from http://nlp.stanford.edu/software/openie.html

EveSem, "an automated NLP pipeline for semantic event extraction and annotation" [46], presented in 2013, *is* an event extraction system, and provides a competitive solution to the problem. Using the results of POS annotation and SRL[25] information, the system manages to identify and order the events in a given text. An evaluation of the performance of the system over a subset of the TimeBank corpus showed the system to achieve a recall of 89.13, and a precision of 85.42; comparable to that of the previous state-of-the-art system, **TIPSem** [46]. Unfortunately however, EveSem does not appear to be publicly available as yet.

TIPSem[26], however, does appear to be available online. TIPSem is a semantic event extraction system for both English and Spanish texts, achieving the highest F_1 scores in tasks B[27] (event extraction) and D (event - document creation time ordering) of the TempEval-2 challenge, and appearing amongst the top scorers in the other sub-tasks [47]. Again, by employing semantic information the tool generally outperforms the equivalent TIPSem-B system that performs the same analysis based solely on syntactic information. However, despite a good performance on the task of identifying and classifying events in text, it does not perform so well on task (iii) of Temp-Eval-2: that of identifying temporal relations between events and temporal expressions. In identifying the temporal relationship between events (Task E), TIPSem achieved an F_1-score of 55%, and although this was not the best score, it was only 1% behind it [47]. Thus, it appears that one of the biggest challenges in this area remains the temporal ordering of events extracted from text.

Despite this, an interesting paper on the *unsupervised learning of narrative event chains* from Stanford presents a three stage process to extracting and ordering the series of events[28] contained within a narrative, with the result achieving a 72.1% accuracy at the binary task of determining if one event is before another on the Timebank corpus [48]. The paper describes the construction of a narrative event chain, defined as "a partially ordered set of events related by a common protagonist" [48], in effect yielding a timeline of events related to the protagonist of interest. As a result, this approach is referred to as *entity-centric*, exploiting the idea that although a narrative typically has multiple actors, there is typically one main protagonist that the narrative follows. Thus, the project uses this protagonist as the *hook* to extract a narrative event chain from a given text.

To construct this narrative event chain, the authors make the assumption of **narrative coherence**: "verbs sharing co-referring arguments are semantically connected by virtue of narrative discourse structure" [48]. An idea intuitively justified by the authors as "a series of argument-sharing verbs is more likely to participate in a narrative chain than those not sharing" [48].

[25] Semantic role labelling

[26] Available from http://www.cognitionis.com/tipsem/

[27] The recognition and classification of events as defined by the TimeML EVENT tag

[28] Defined simply as a verb and its participants

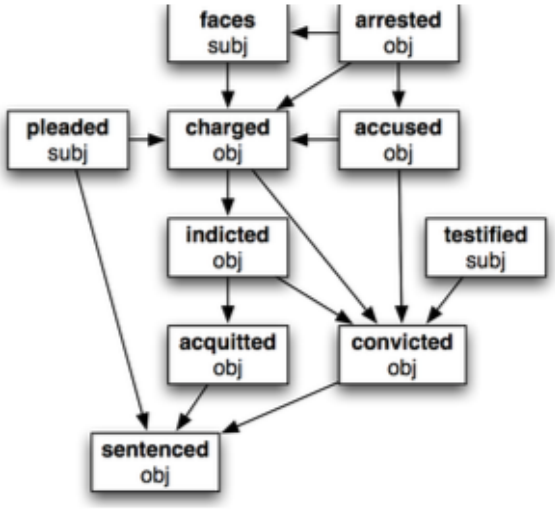


Figure 2.14: An automatically learned prosecution chain [48]

The two stages of particular importance to my project are those of identifying events to be included in the narrative event chain, and subsequently identifying a partial ordering of the extracted events subject to the *before* relation. In order to construct the initially unordered narrative event chain, the authors first define a pairwise (event-event) comparison score using point-wise mutual information over the number of grammatical arguments the two events share, following from the narrative coherence assumption. With these pairwise scores defined, a narrative chain can be constructed by first extracting all events directly related to the protagonist of interest, and then building up the chain based upon the pairwise scores of candidate events against the events already included in the chain [48]. The second stage utilises the temporal attributes of the extracted events, such as tense, grammatical aspect, and aspectual class, in addition to other linguistic features in order to classify the temporal relationship between events, restricted solely to identifying the *before* relation for the purpose of this project [48].

As a result, the system achieved an 89% accuracy on identifying the partial ordering of event chains containing 10 or more pairs of ordered events, i.e. chains that most progress through time. Figure 2.14 shows the rather impressive results of combining the two stages above to yield a directed graph reflecting the partial ordering of events discovered by the system. Again, despite the system's lack of availability, the paper describes an effective technique for the construction of narrative event chains that could prove useful to the development of my application.

So, despite event extraction being one of the "most investigated tasks of information extraction" [6], it appears that the prevalence and availability of solutions is still somewhat limited, and the problem remains a challenge. However, recent studies have identified a range of effective strategies, all building upon the additional information that can be obtained using the state-of-the-art NLP solutions for other syntactic and semantic NLP tasks, namely part-of-speech, co-reference resolution and semantic role labelling, among others, and so provide the means to develop an effective solution to the problem.

2.1.7 Other Areas of Interest

Finally, in addition to these areas of NLP that are of immediate interest, it is worth mentioning that there are a whole host of other tools and active areas of NLP research that could prove useful in the development of an effective application. Three areas of particular relevance are those of text summarisation, information extraction, and cross-document co-reference resolution.

Text summarisation aims to process and convert a large piece of text into a shortened summary that maintains only the most important pieces of information. Again, how we define *importance* makes a big difference to the output, with a typical approach being to keep information regarding frequently mentioned entities in the text. Such a tool could prove useful in my application by reducing the amount of information to process, potentially reducing the difficulty of event extraction as the summarised information already clusters related information. Alternatively, it could be a useful post-processing step to construct more concise event descriptions to display in the resulting timeline. However, it is important that I am able to locate the original source of any summarised information for two key reasons:

1. To be able to inform the reader exactly where in the original text they can find the details of the summarised event
2. To be able to apply other NLP techniques to the original text to extract additional supplementary information related to an event that can be displayed in the resulting timeline

Google's TensorFlow is one example of such an application, and has been trained to perform text summarisation on news articles to a high standard, with some examples shown in Table 2.7.

Information extraction, the broad task of extracting useful information from text, could also prove to be useful. Named entity recognition, temporal information extraction, and event extraction can all be viewed as sub-tasks of information extraction, however, in the broadest sense, information extraction is essentially a slot-filling task: the task of extracting factual information from a piece of text, such as the extraction of the tuple (Obama, born in, Honolulu, Hawaii) by the Stanford OpenIE system, an example seen in our discussion of event extraction tools. Facts like these could prove to be useful supplementary information to display along with the associated events in the resulting timeline, potentially allowing further insight to be drawn from the visualisation.

Finally, a relatively new area of research is that of **cross-document coreference resolution**. Extending the task of coreference resolution to deal with the identification of entity coreference across a collection of documents, rather than just one. This could also prove to be an interesting extension to the project, allowing us to construct timelines consisting of events from multiple texts.

Input: Article 1st sentence	**Model-written headline**
metro-goldwyn-mayer reported a third-quarter net loss of dlrs 16 million due mainly to the effect of accounting rules adopted this year	mgm reports 16 million net loss on higher revenue
starting from july 1, the island province of hainan in southern china will implement strict market access control on all incoming livestock and animal products to prevent the possible spread of epidemic diseases	hainan to curb spread of diseases
australian wine exports hit a record 52.1 million liters worth 260 million dollars (143 million us) in september, the government statistics office reported on monday	australian wine exports hit record high in september

Table 2.7: Results of Google's TensorFlow text summarisation tool [49].

2.2 Clustering Techniques

Clustering is the process of grouping a number of related items into a collection of disjoint sets. A variety of techniques exist for such a task and have each been successfully applied in a number of different contexts. We discuss a few of these techniques below, and evaluate their potential with respect to our use case of text clustering.

2.2.1 K-Means Clustering

K-means clustering is a simple and extremely popular clustering method. This is an iterative method for grouping items into K disjoint clusters, requiring the user to first specify the value of K they desire. The method begins by randomly choosing K points distributed throughout the feature-space of the items to be grouped, these points being referred to as the initial *cluster centroids*. Each item is then moved into the cluster of its nearest centroid using some pre-defined distance metric, typically Euclidean distance. Each of the K cluster centroids are then re-calculated as the average value of all the items contained within the cluster, and the process repeats. This iterative cycle continues until we reach a stopping condition, typically when the clusters no longer change between an iteration.

This is a relatively simple method, and while having a theoretically high upper bounding complexity, in practice the process typically finishes quickly [50]. However, the major disadvantage of this approach is the requirement to specify K in advance; how do we know how many events we expect to extract from the text? It would also be a fairly expensive operation to repeat the process for various values of K, requiring a complete re-computation for each value. Another significant drawback of this approach is the fact that the choice of initial cluster centroids can also alter the resulting clusters

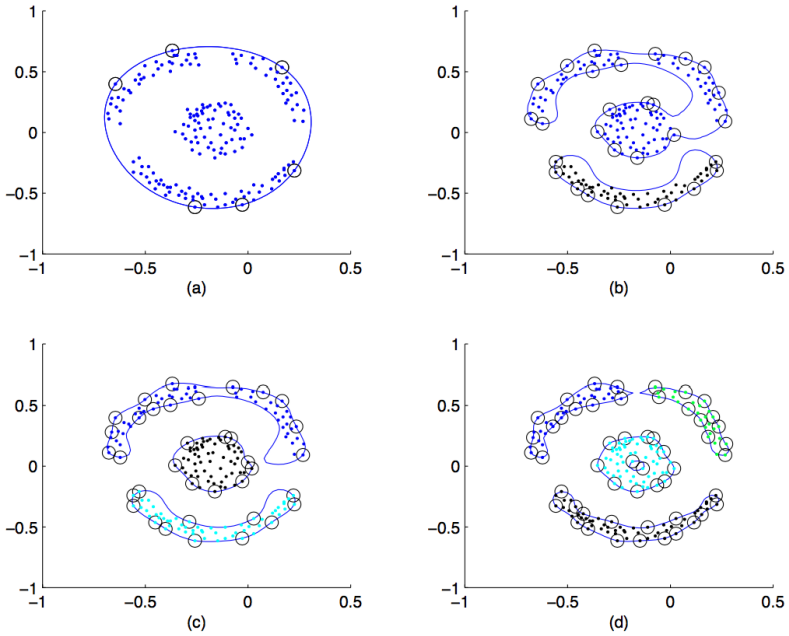


Figure 2.15: The cluster boundaries identified by Support Vector Clustering as the scale parameter, q, of the Gaussian kernel is increased. (a): q = 1 (b): q = 20 (c): q = 24 (d): q = 48. [51]

obtained; an issue typically overcome by developing a heuristic method for choosing a good set of initial centroids. This non-determinism is something we certainly want to avoid in extracting a set of events at a given level of detail.

2.2.2 Support Vector Clustering

Unlike K-means clustering, Support Vector Clustering (SVC) is a non-parametric model that makes no assumptions as to the number of clusters in the underlying data. The technique is based upon the use of Support Vector Machines, using a Gaussian kernel to map items in the input space into a high dimensional feature space [51]. The method then aims to find the minimal enclosing sphere of the feature space, which when mapped back into the original input space defines a number of contours around the input data. More intuitively, the method essentially aims to draw boundaries in the areas of the input space where there is little data [51]. As a result, the areas enclosed by each of these boundaries become our clusters [51].

The results produced can be adjusted through the alteration of two parameters [51]:

- The width of the Gaussian kernel determines the tightness of boundaries to the data points. Decreasing this value will lead to an increasing number of clusters.
- A soft margin constraint allows for outliers, leading to smoother boundaries and reducing over-fitting.

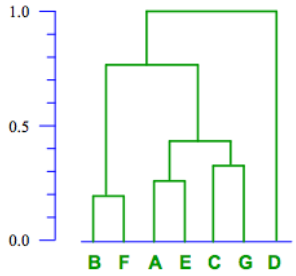


Figure 2.16: Dendrogram illustrating the results of hierarchical clustering. In this example the y-axis represents the "distance" (inverse of similarity) between items, thus items with a smaller distance between them are linked before those with a greater distance [52].

Figure 2.15 shows the effect of decreasing the width of the Gaussian kernel on the boundaries identified [51]. These properties make SVC an extremely promising option for event clustering, providing us with a means of adjusting cluster granularity without the need for specifying the number of clusters expected. Additionally, the resulting quadratic problem can be solved with an off-the-shelf solver relatively efficiently and yields a single global solution [51]. Thus, addressing a number of the issues identified with K-means.

2.2.3 Hierarchical Clustering

Hierarchical clustering is also a non-parametric model, similarly not requiring the user to specify the number of clusters expected in advance. In this case, we have a recursive process that progressively builds up a set of clusters. Beginning with an individual cluster for each item, at each step we merge the two most similar clusters according to some pre-defined similarity metric, continuing this process until our stopping criteria is met, typically because either there is no reason to cluster any longer or all items are contained within a single cluster. More precisely, this describes the basic process of *agglomerative* hierarchical clustering. *Divisive* hierarchical clustering, on the other hand, works in the reverse manner to gradually break down a single large starting cluster into a set of smaller clusters.

As a result we obtain a hierarchical structure (as the name suggests) that can be illustrated as a dendrogram of the form shown in Figure 2.16. Such a dendrogram highlights the order in which the clustering took place reflecting which items are most similar, and also shows the cluster sets we'd obtain if we took a horizontal cut at any particular step. For example, taking a cut at a score of 0.5 in the example of Figure 2.16 would yield 3 clusters: {B, F}, {A, E, C, G}, and {D} [52]. A structured output like this has proven extremely useful in highlighting gene lineage in Bioinformatics, and this technique has also been used in previous works on document clustering. In the case of [53], not only does document clustering provide a means to efficiently search through a number of documents by collecting only those of relevance to your search, but by employing hierarchical clustering the user can also adjust the clusters of documents produced to allow them to "browse through the results at their desired level of specificity" [53].

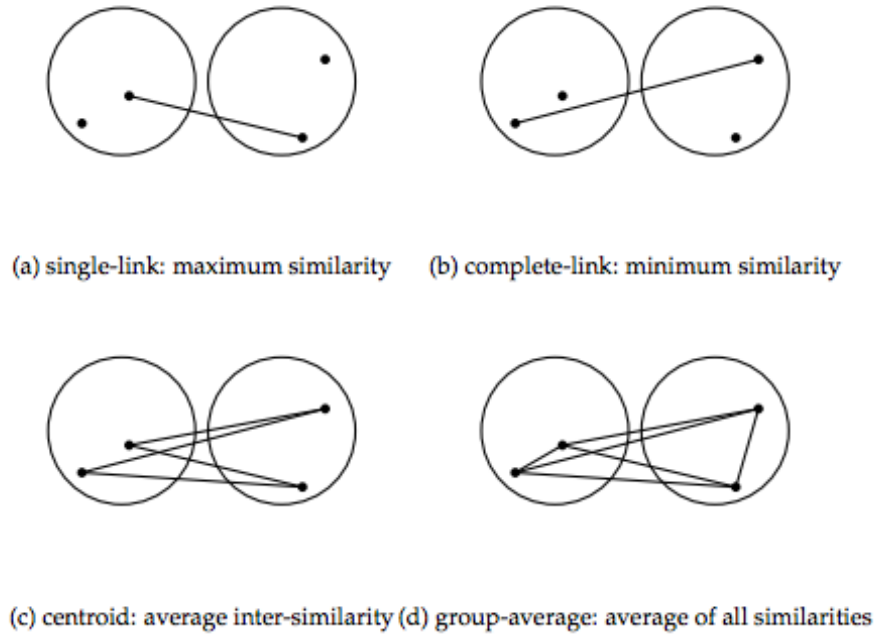


Figure 2.17: Graphical representation of the different linking strategies between neighbouring clusters in Hierarchical clustering [54].

While the fundamental approach to hierarchical clustering is relatively fixed, the results can be changed significantly through the choice of *linking strategy* employed. Following the merge of two clusters, the similarity score of the new cluster with its neighbouring clusters is updated, however exactly how we do this depends on our choice of linking strategy. The three most common strategies being: **single-link**, **complete-link**[29], and **average-link**, illustrated nicely in Figure 2.17 [54].

Single link clustering updates the scores between any two clusters with the scores of the most similar items within each of the clusters [54]. **Average link** clustering updates the scores between any two clusters with the average of the scores between the items contained in each cluster [54]. And finally, **complete link** clustering updates the scores between any two clusters with the scores between the two most dissimilar items contained in the clusters [54]. Each of these yields different dynamics that may be preferable under different circumstances.

Finally, while the two alternative methods described previously immediately result in a set of clusters, hierarchical clustering leaves us with this hierarchical structure from which we need to extract our clusters. Various *cutting* strategies exist to identify a good place to cut the hierarchy to obtain a desirable set of clusters. One approach is to identify a good score threshold at which we take a horizontal cut, as we did in the brief example of Figure 2.16, or alternatively cutting the tree at the point where the gap between two successive merges is largest is credited with often yielding a natural cutting point [54]. Thus, hierarchical clustering presents a fairly simple yet effective technique for clustering, which yields a structured hierarchy of clusters that naturally fits with my objective of providing an adjustable level of detail of the resulting events extracted.

[29] Also known as furthest-link.

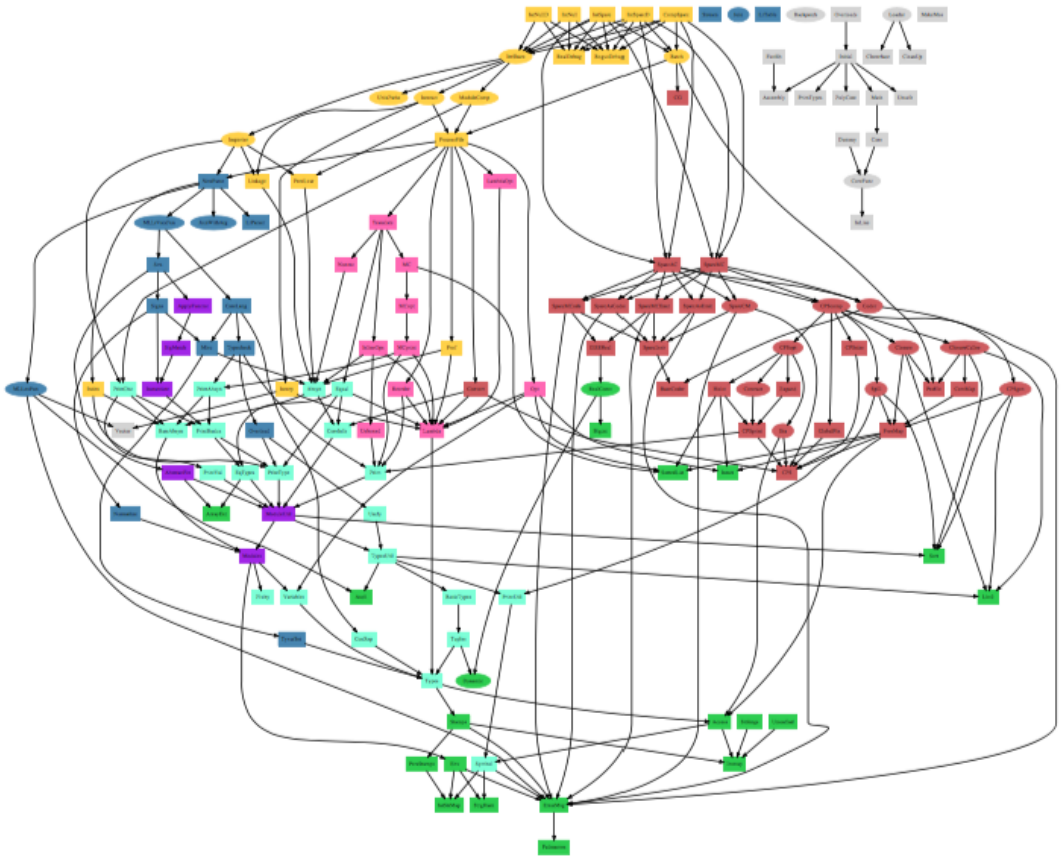


Figure 2.18: An example graph produced by GraphViz's dot tool highlighting its ability to clearly layout even complex relationships between elements in a graph [55].

2.3 Timeline Visualisation

As important as the extraction of relevant events from a given piece of text is, an effective visualisation of the results is also vital in providing valuable insights into the underlying text. Part of the novelty of this project is the emphasis on the development of an *infographic* event timeline; i.e. providing the user with a concise visualisation displaying a variety of useful information extracted from the text, at a glance.

2.3.1 Existing Tools

Many tools already exist for the purpose of constructing a timeline given the structured data, with varying levels of flexibility. One of the most flexible tools being GraphViz[30], a graph drawing tool that employs various layout heuristics in order to construct a clear representation of the resulting graph. The nature of this tool allows the resulting timeline to be drawn in any way we wish, enforcing very few constraints on the resulting visualisation and thus presenting an extremely versatile option for the construction of an event timeline. An example graph produced by GraphViz is shown in Figure 2.18.

[30] Available from http://www.graphviz.org/

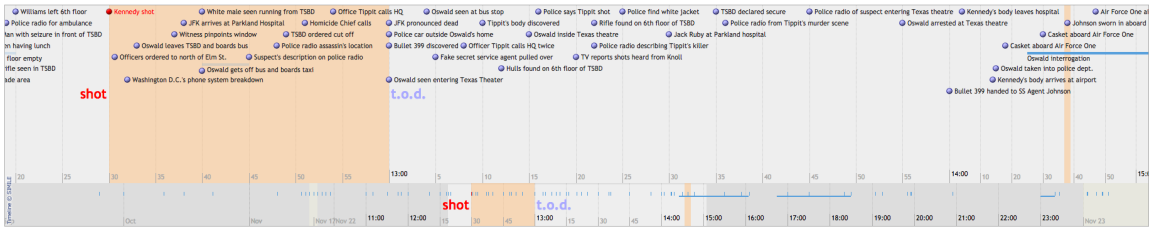


Figure 2.19: An example timeline produced by MIT's SIMILE [56].

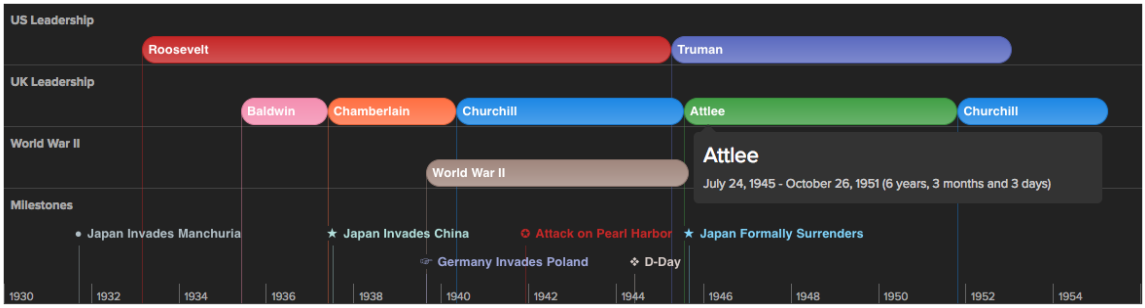


Figure 2.20: An example timeline produced by Preceden [57].

MIT's SIMILE[31] is another option; this is a publicly available timeline visualisation tool for the construction of interactive event timelines that can be embedded directly into a web page. SIMILE allows the automatic construction of event timelines from XML formatted input data, but lacks the flexibility of tools such as GraphViz in terms of the possible visualisations achievable. For example, it would not be possible to represent the distinct paths of multiple actors on the same timeline without embedding multiple SIMILE widgets into the webpage. Thus, the lack of flexibility here makes SIMILE a less attractive option. An example timeline produced by SIMILE is shown in Figure 2.19.

A more modern alternative to SIMILE is Preceden[32], which again constructs interactive event timelines that can be directly embedded into a website. In addition, this tool offers the ability to layer the resulting timeline, providing the potential to have layers for distinct actors, topics, or places mentioned in the text. The software appears to produce a clear, modern looking timeline that can be constructed by supplying a CSV formatted file describing the data to display. Thus, this tool presents a good compromise between the two previously mentioned alternatives. An example timeline produced by Preceden is shown in Figure 2.20.

2.3.2 Force-Directed Graphs

However, fundamentally we have a data visualisation problem: our aim being to construct a clear visualisation to faithfully reflect the information being displayed. Force-directed graphs enable this by encoding the various properties of the data being displayed as physical forces that influence the resulting layout of the graph induced.

[31] http://simile-widgets.org/timeline/
[32] Available from https://www.preceden.com/

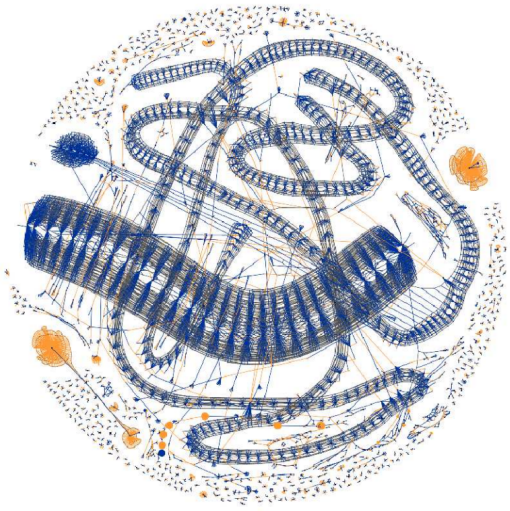


Figure 2.21: Force-directed visualisation of Bitcoin transactions revealing the occurrence of a transaction rate attack on the Bitcoin network [58].

One recent example of the use of force-directed graphs is in the work carried out by researchers at Imperial College London on visualising Bitcoin transaction patterns, which I was fortunate enough to see for myself [58]. One of the key properties of force-directed graphs that make them so appealing is the fact that they provide the means to produce natural, emergent structures without requiring the user to manually specify the placement of nodes and edges [59]. As a result, we allow the data itself to influence the resulting layout, which can result in incredibly interesting and unforeseen insights being drawn from the resulting plot. In the case of the Bitcoin visualisation, the resulting plot enabled the authors to witness a developing transaction rate attack on the Bitcoin network and recognise further unusual activity in unprecedented clarity [58]. This result is illustrated in Figure 2.21 [58].

However, while force-directed graphs provide the means for some elegant visualisations, it is important to be aware of the limitations of scalability. Force-directed graphs only behave well for a relatively small graphs consisting of at most hundreds of nodes [59]. Beyond this, the effect of many local optima in the underlying physical simulations and unpredictability of the initial layout can lead to fairly poor results [59]. In our context, this should not be a constraint that affects us in practice as it is unusual to have a story with over 100 events, or at least it would be extremely difficult for any reader to comprehend such a timeline.

Thus, the use of force-directed graphs provide a novel opportunity to assess their ability in producing a timeline of the form we saw in the introduction in Figure 1.1, reflecting the interactions of characters in the text while maintaining a good level of clarity. They also present the means to explore new ways of visualising the events within text, and the properties they can reflect.

2.4 Choice of Text Corpora

The choice of text corpora for development is another important factor to consider. The aim of this project is to produce an application that is able to construct general event timelines for texts over a large selection of different domains, thus my aim is to keep the project as domain agnostic as possible. Of course, in some cases domain-specific knowledge is required, for example I would not expect my system to perform well in identifying the events of a biological phenomenon, such as the transcription and expression of genes etc., but do hope it is able to recognise events we'd recognise as general knowledge. As a result, the choice of text corpora for development and evaluation of the system will prove vitally important to achieving good results.

Fairy tales are one possible corpus of material, and are used in the development of Cui's automated timeline generation software that we will discuss shortly in Section 2.5 [60]. Fairy tales are short, simple narratives, usually with a clear sequence of events that one would like to extract. However, the use of fictional character and place names typically does not interact well with current NER or coreference tools that often rely on a world-knowledge that does not include fictional information. As a result, names such as *Goldilocks* are unlikely to be recognised, and even less likely to be identified with the correct gender. This could be tackled though the creation of our own labelled training data and retraining the tools to recognise such expressions, however this poses an additional workload in a context that does not appear to be the most impactful application of this software.

News articles are another option, which immediately overcome the issue of non-fictional names and places. The short nature of news articles typically with explicit temporal expressions, make them a good target for many existing NLP tools, which in a lot of cases have been specifically trained on news articles from the TimeBank corpus of annotated material. Notable examples include text summarisation, named entity recognition, and temporal information retrieval tools. So, news articles remain a useful source of material for the purpose of judging the best results we could perhaps hope to achieve. Additionally, news articles are not always written in chronological order, thus putting an emphasis on the need to re-order the events described before constructing the resulting timeline: an additional challenge that we may consider depending upon our time constraints.

Historical texts are another appealing domain, as the insights gained from a timeline visualisation of a historical text can prove invaluable in a variety of different contexts, from education to current historical research. Such texts typically span over a reasonably long period of time relative to other texts, and again consist of non-fictional entities and events that can often be resolved by current solutions to named entity recognition and temporal information retrieval. However, the challenge we may face with historical texts is that of *scalability*. That is, history text books are typically hundreds of pages long, which combined with the amount of processing time required by the different stages in an NLP pipeline could present additional challenges that we'd rather postpone to a later stage. Our focus is on incremental improvement, developing a strategy that proves effective on short, simple texts before tackling the challenges of scalability.

Thus, my initial focus shall be on short, simple English prose: texts consisting of non-fictional character and place names, and a mix of relative, explicit and implicit temporal expressions. One method of generating such texts could be to replace all fictional entities in fairy tales and short novels with non-fictional equivalents. This corpora of text encapsulates the broadest domain of possible input texts with the nice property that such texts are typically written in chronological order, removing the need to introduce temporal information extraction for re-ordering the events identified. The focus on non-fictional entities allows us to explore the maximum potential of existing NLP tools by testing them under conditions most similar to their development conditions.

2.5 Related Work

The topic of automated construction of event timelines from text has become a prominent area of study in recent years, with multiple research groups and authors citing this as a potential application of NLP technologies. Now, due to the advancement of the relevant NLP tools reaching a point of such sophistication, the development of a solution to this problem is becoming an increasingly viable prospect.

One project similar in nature to mine was that of Lu Cui, fellow Imperial College student, who worked towards the development of a web application for the automatic extraction of actor-centric event timelines from text [60]. Cui's work focused on combining machine learning techniques with NLP tools to be able to train a model to recognise the actors of choice within a given text, regardless of the name being fictional or non-fictional, using the Stanford CoreNLP toolkit to achieve this. Experimentation with the use of regexes versus named entity recognition for this purpose revealed named entity recognition to be the superior technique in this case for identifying the entities in text, a technique that I intend to employ. For the purpose of the project, Cui defines an *event* as an "action and [the] characters involved", focusing primarily on the fairy tale of *the three little pigs*: a narrative text that has the advantage that the order of events typically follow the order of occurrence in the text [60]. Figure 2.22 shows the output of the resulting web application on three little pigs narrative.

Despite a good performance on this example, Cui also identifies some of the limitations of the system, most notably the fact that the application does not account for any event re-ordering that may be required when events are not expressed in chronological order. Secondly, Cui notes the issue of resolving collective co-reference in text, such as the difficulty in identifying that "the three pigs" refers to all three of the little pigs: one of the key challenges of coreference resolution [60]. Cui also presents and evaluates a collection of potential timeline visualisations for the story of three little pigs, but did not have the time to further explore the creation of such timelines.

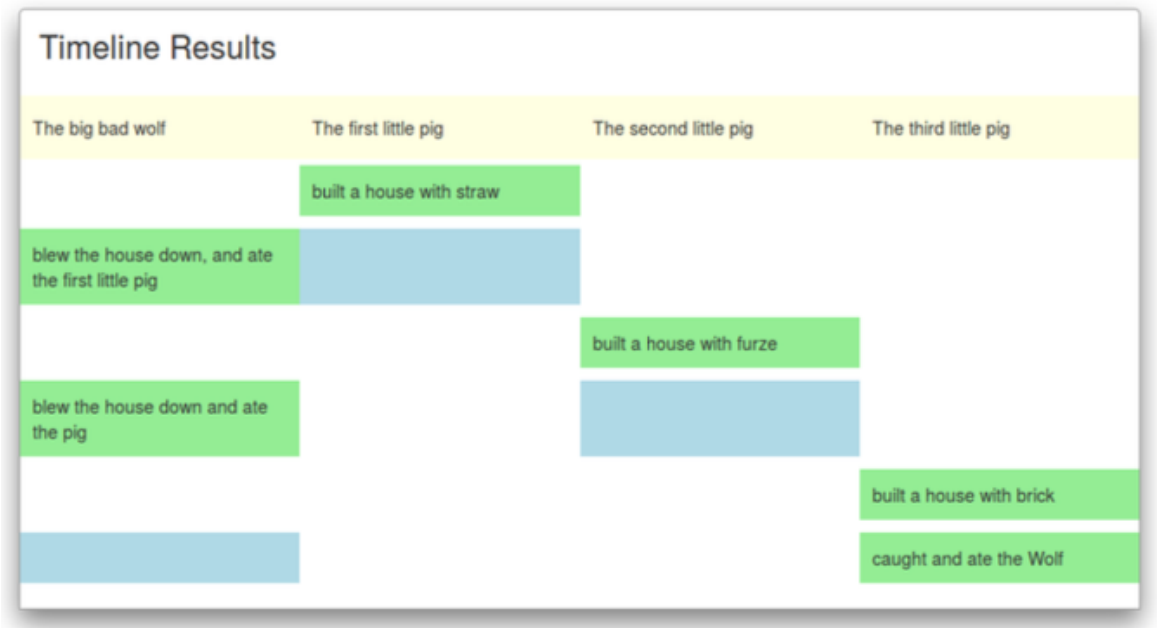


Figure 2.22: Lu Cui's event timeline for the three little pigs story [60].

Another, earlier project pursuing a similar goal describes a tool focused on extracting only the most salient events from Wikipedia articles, in particular searching for events associated with an explicit date that can then be displayed on the resulting timeline [61]. The authors split the task into 2 distinct subtasks:

- Identifying which events are the most important in the text
- Ordering and displaying these on a timeline using the MIT SIMILE software discussed in Section 2.3.

The importance of an event is evaluated using a classifier trained using event and sentence level features, such as whether or not a sentence contains any numerical digits, or whether it contains any named-entities and the frequency with which these entities are mentioned in the document [61]. In order to recognise temporal expressions in text, the authors choose to employ an extensive collection of regexes, seemingly outperforming GUTime, a predecessor to Stanford's SUTime temporal information retrieval tool that we discussed back in Section 2.1.4.

Evaluating their results against a selection of manually annotated Wikipedia articles, the system achieves an F_1-score of 66.8%. Although, there are certainly some discrepancies as to what is considered an *important* event. A weakness of this system is its inability to effectively order events which do not mention an explicit time, where in such cases the authors make the assumption that historical texts are typically written in chronological order. An example of the resulting visualisation, a combination of MIT's SIMILE timeline widget and Google Maps to display the location of extracted events, is shown in Figure 2.23.

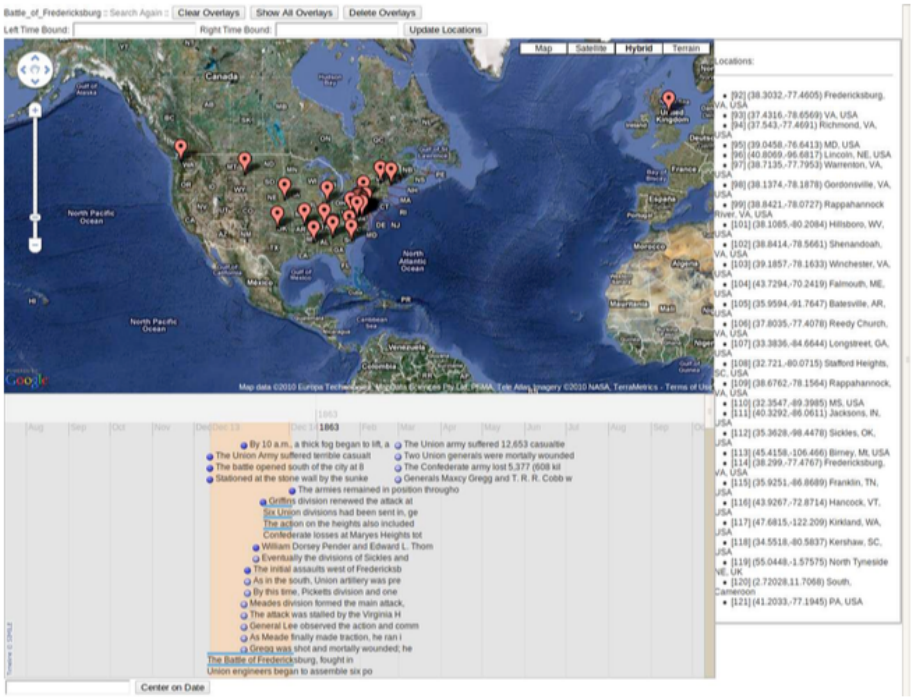


Figure 2.23: An example visualisation of a historical Wikipedia article [61].

Linea is another tool focused on the generation of event timelines from Wikipedia articles, using a three stage process consisting of *event extraction*, *event ranking*, and *event matrix construction* to automatically construct a timeline of the 10 *most important* events in the article [62]. The *event matrix* is a visual tool used to present a collection of histograms showing the number of sentences that refer to each period in time, as a result allowing the user to quickly see important periods of time in the text. Each histogram can focus on a different period of time, using a user-specified scale to provide insights that may not be visible initially. Through this the user is able to explore and customise the resulting timeline [62]. Figure 2.24 shows an example of such a histogram. This appears to be a useful visualisation that certainly provides additional insight that would not be immediately gathered from the text alone, or perhaps even from a timeline alone, so is certainly an additional visualisation that I may consider including in my infographic event timeline.

The display of only the *most important* events appears to be a common aspect to these applications, but is not a restriction I wish to impose. Indeed, the restriction of the visualisation to solely seemingly important events could be particularly dangerous in the context of witness statements; it is the job of the lawyer and the courts to decipher what is important information in a witness account, and what could prove useful in determining the outcome of a case. In many cases it is a small, seemingly insignificant, piece of information that provides the most useful insight. My aim is to construct an accurate timeline displaying all the events identified in a given text, while the idea of allowing the user to customise the resulting timeline is certainly a useful feature.

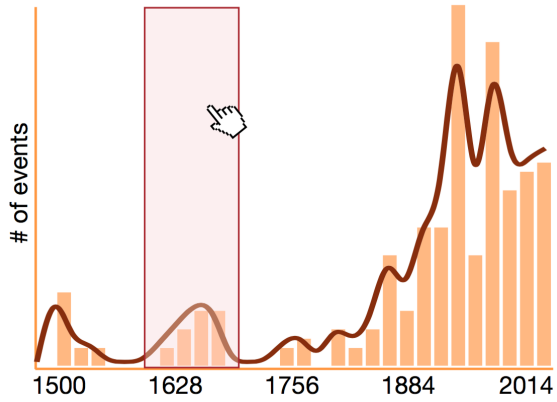


Figure 2.24: Example histogram produced by Linea over a historical Wikipedia article [62].

A severe limitation of the Linea system is that it uses dates as a proxy for important events with the reasoning that if the writer decided that it was worth being specific about exactly when something happened, it is probably important [62]. Despite being an effective strategy in the context of Wikipedia articles, in the broader context of narrative text this is certainly not the case.

Another project following a similar initiative to mine was that of [48], constructing narrative event chains, as discussed in Section 2.1.6 on event extraction. The project by Stanford appears to have achieved extremely accurate results, producing an effective timeline from text without the restriction to solely explicit temporal information. Utilising grammatical aspect, verb tense, and aspectual class, in addition to other linguistic features, the system achieves a good accuracy on identifying the *before* relationship between the events extracted. As a result, the system produces a narrative event chain that could quite easily be displayed in the form of an event timeline with some additional effort.

Lastly, the recent work of [63] pursued the idea of automatically visualising the contents of a children's story in a virtual world. This work focusses primarily on the context of children's stories, and highlights the difficulties of current NLP tools to adapt to other domains. In response, the authors manage to achieve an improvement of between 8 and 20% F_1 across 4 different semantic roles identified by their SRL tool by developing a set of training material reflecting their target input domain [63]. Further to this, they also observe the ignorance of existing coreference resolution tools to the typical constraints of storytelling: failing to correctly resolve nominal coreferences such as "a woman" with "the woman" [63]. As a result, they develop their own tailor-made system yielding an improvement of 4.46% F_1 over their example texts [63]. This again highlights the impact of training data on the performance of the current state-of-the-art NLP tools, and is thus why I shall initially tailor my input slightly in attempt to observe what is maximally achievable from current tools at present, without moving these tools too far from their development domain.

This study highlights some valuable points to consider, and while closely related to my project targets a very different outcome. [63] treats each *sentence* as a possible event to visualise, with the goal of then identifying the key action described in that sentence and its corresponding arguments, such as the actor, location, and any tools being used. The probabilistic graphical model approach used works well in this context, mapping the input features extracted from each sentence to one of a finite number of possible actions that can be displayed by their graphical engine [63]. My project on the other hand focusses on displaying a visual summary of the information extracted, identifying *events* at a higher level, and thus not necessarily requiring the *understanding* of the events described.

Chapter 3

Laying the Foundations

3.1 Aims

In this chapter we outline exactly we consider to be an event, discuss our experimentation with the most promising tools we found during our initial investigation discussed in the previous chapter, and finally present our selected technologies for use in building the underlying event extraction system discussed later in Chapter 6. This experimentation proved vital in deciding which tools provided annotations of most value to both the resulting visualisation and as possible features to inform the event clustering process, and consequently guided our approach throughout the rest of the project.

3.2 Defining an Event

Of central importance to the project is exactly what we define to be an *event*. As we saw in Section 2.1.6, the majority of existing event extraction tools focus on recognising key phrases that act as event triggers in a piece of text and then extracting their surrounding arguments. While such an approach is useful in the context of information retrieval, it is not as relevant in the case of event timeline construction. As mentioned earlier, it is a human's innate ability to generalise information to varying degrees of detail that we wish to emulate. Thus, we define an event as any *distinct* occurrence or set of related actions that take place within a piece of text, allowing this to be generalised to a varying level of detail.

At the finest level of detail, we restrict ourselves to the granularity of treating any individual sentence as an event, while at the coarsest level of detail we may encapsulate the entire story as a single event. The reasoning for this is two-fold: in the context of a narrative text, any single sentence will typically only describe a single event, and secondly this limits the maximum number of events to be displayed in the resulting timeline with a potentially overwhelming result if we were to additionally decompose sentences into their constituent events.

3.3 Experimentation

3.3.1 Coreference Resolution

Coreference resolution lies at the heart of this project. Being able to recognise which actors are involved in each of the events is paramount to being able to construct a resulting timeline of the form shown previously in Figure 1.1. As noted in Section 2.1.3, while both Illinois and Stanford universities offer readily available coreference resolution systems, it is Stanford that appears to provide the superior performance at present, with three alternative implementations to choose from: a rule-based implementation, a statistical implementation, and a neural implementation. While a naive approach would suggest we simply use the neural implementation, having achieved the top score on the CoNLL 2012 dataset, we're now using these tools under a completely different context, and as such the results on the CoNLL 2012 corpus is unlikely to accurately reflect the results in the domain of *narrative* text. Additionally, we explore the effect of adjusting the parameters of these tools on the results produced; the statistical implementation provides a *pairwise score threshold* that can be reduced to increase the likelihood of merging two clusters into a single coreference chain [28]. Similarly, the neural system defines a *greediness* parameter that can be increased to increase the chance of merging two coreference clusters [28]. For this comparison we take the two example texts of *Little Red Riding Hood* and *Goldilocks and the Three Bears*; our exact versions are included in Appendix A.

We begin with the text of Little Red Riding Hood, setting the *score threshold* parameter of the statistical system to the recommended default of 0.35, and the neural with a *greediness* parameter of 0.45, slightly below the suggested default. In the majority of cases all three of the tools perform similarly, however Table 3.3 highlights some of the key differences between the tools on this first text. It is clear that each of the tools have strengths and weaknesses, yet the overall result here is fairy inconclusive. One facet that is clear: all of the systems struggle with nominal mentions. Particularly in the case of sentence 6 where all 3 of the tools fail to recognise the mention of *the wolf*. This was an issue highlighted by our background research, and it is clear that this may be a significant problem here: particularly in the context of fairy tales where characters often lack names. Table 3.1 summarises the overall performance of each of these implementations on this example, and reveals the statistical implementation to yield the marginally superior results. These metrics were calculated only considering the ability of the implementations to correctly recognise coreferent *actor* mentions. For this purpose, we ignore the performance on identifying coreference mentions of other entities, such as *the woods* for example, that is also repeatedly mentioned but not directly relevant to our investigation here.

Increasing the greediness of the neural implementation had no effect until we reached a value of 0.55, leading to a slight improvement on this example. In this case correctly recognising the mention of *the wolf* in sentence 6, as discussed previously. These additional results are shown in Appendix B.1. However, while an increased greediness yielded mildly better results in the Little Red Riding Hood example, in the case of Goldilocks this resulted in all 4 of *Goldilocks*, *the Papa bear*, *the Mama bear*, and *the Baby bear* being resolved into the same chain, i.e. being treated as a single entity. This highlights the negative impact of being overly greedy in constructing coreference chains.

	Rule-based	Statistical	Neural
Recall	77.2%	83.1%	75.6%
Precision	93.8%	95.5%	95.2%
F1-Score	84.7%	88.9%	84.3%

Table 3.1: Performance summary of each of the Stanford Coreference Resolution implementations over the example of Little Red Riding Hood. These scores only consider the matches of *characters* in the text, ignoring any other entities that are mentioned multiple times in the text that may or may not also be identified by the systems. Links to the raw annotation data can be found in Appendix C.1.

Incorrectly merging two large chains can lead to a severe degradation in the quality of results: incorrectly tagging effectively every event with all 4 characters would require the user to correct the mentions in *every* event, yielding no real benefit to the user. For our purposes, we'd rather have a greater number of distinct chains, such as the distinct chains for *Little Red Riding Hood* and *a little girl who lived in a village near the forest*, than incorrectly merging two coreference chains that are actually distinct entities. i.e. Favouring *precision* over *recall*. In the former case the user need only correct the system to consider two characters as the same, while the latter may require the user to manually update the character tags for every single event.

Another factor to consider is the effect of the text consisting of predominantly *nominal* mentions as opposed to *named* mentions, bearing in mind that the majority of existing training data for this task consists primarily of news articles and other non-fictional texts. The use of unfamiliar language may be part of the reason for all 4 of the characters in the Goldilocks text were merged into a single cluster. To explore this, we additionally repeat the same analysis as above with an adapted version of the Little Red Riding Hood text with the following changes:

- *Little Red Riding Hood* is now referred to as *Sophie*.
- *Her mommy* is instead referred to as *Caroline*.
- *Her Grandma* is now referred to as *Alice*.
- *The wolf* is instead named *Alan*.
- *The woodsman* is instead named *Andrew*.

	Rule-based	Statistical	Neural
Recall	90.7%	78.4%	88%
Precision	97.1%	86.6%	97.1%
F1-Score	93.8%	82.3%	92.3%

Table 3.2: Performance summary of each of the Stanford Coreference Resolution implementations over the example of Little Red Riding Hood with characters mentions replaced by *named* mentions. These scores only consider the matches of *characters* in the text, ignoring any other entities that are mentioned multiple times in the text that may or may not also be identified by the systems. Links to the raw annotation data can be found in Appendix C.1.

Thus, we replace the large set of nominal mentions we had previously with named mentions that are more typical of the training data used by these systems. The results validate this hypothesis, with the statistical and neural systems showing the most significant improvement, particularly during the dialogue between Little Red Riding Hood and her Grandmother, where the rule-based system struggles. Table 3.4 again highlights some of the key differences between the systems, where the sentence numbers included are for ease of reference in order to compare these results to those of Table 3.3. In this case, the neural system seems to outperform both of the other implementations, with the rule-based system struggling particularly during the dialogue between Little Red Riding Hood and her Grandmother, and the statistical system incorrectly resolving *Alice* (the grandmother) as the same entity as *the little girl*: we first see this in sentence 1 of Table 3.4, and unfortunately this error subsequently propagates throughout the rest of the text.

Table 3.2 summarises the overall performance of these implementations over the altered text, quantifying the performance improvement of both the rule-based and neural implementations. We also see the reduced overall performance of the statistical implementation on this example due to the early error. These results clearly reveal the impact of *named* mentions on the performance of coreference resolution systems, and thus the potential benefit to be gained from preprocessing an input text to ensure the characters mentioned are predominantly *named*. Additionally, Table 3.2 also appears to reveal the impact of using these tools perhaps out of their intended domain. The nature of the training material used for the neural implementation means that when tested under the domain of *narrative* texts it performs no better than the rule-based implementation, despite being far superior in the context of the CoNLL 2012 data set.

As a result, we choose initially to employ the Stanford neural coreference implementation with a greediness of 0.45. This yields a fairly consistent set of results whether we have named mentions or nominal mentions, and avoids incorrectly merging clusters that should remain distinct. Increasing the *pairwise score threshold* of the statistical implementation improved its performance in the preprocessed example, but not sufficiently to outperform the neural implementation.

No.	Sentence	Mentions (Rule-based)	Mentions (Statistical) - 0.35	Mentions (Neural) - 0.45	Comments
1	Remember, go straight to Grandma's house, her mommy cautioned.	her mommy, Grandma	her mommy, not her Grandma, a little girl who lived in a village near the forest	her mommy, her Grandma	Statistical best as identifies all 3 characters
3	I'm on my way to see my Grandma who lives through the forest, near the brook, Little Red Riding Hood replied.	Little Red Riding Hood, the way, the poor Grandma, the forest	Little Red Riding Hood, Wolf, not her Grandma, the forest	The wolf, Little Red Riding Hood as she, the forest	Rule-based and statistical best.
4	Poor Grandma did not have time to say another word, before the wolf gobbled her up!	a time, the poor Grandma, The wolf	Wolf, not her Grandma, a little girl who lived in a village near the forest	The wolf, Poor Grandma	Neural and rule-based best; Statistical incorrectly includes little girl
5	It's me, Little Red Riding Hood.	a cackly voice, The wolf	Little Red Riding Hood, me, it	The wolf, Little Red Riding Hood as she	Statistical best, then neural
6	Oh, I just have touch of a cold, squeaked the wolf adding a cough at the end to prove the point.	Your	she		All miss the wolf
7	But Grandma! What big eyes you have, said Little Red Riding Hood.	Little Red Riding Hood, you	Little Red Riding Hood, But Grandma, you	Little Red Riding Hood as she	Statistical best
8	She ran across the room and through the door, shouting, "Help! Wolf!" as loudly as she could.	the door, Wolf, the poor Grandma	her, the door, Wolf	the door, a little girl who lived in a village near the forest	Statistical and neural best

Table 3.3: Comparison of Stanford coreference resolution implementations on the Little Red Riding Hood text.

No.	Sentence	Mentions (Rule-based)	Mentions (Statistical) - 0.35	Mentions (Neural) - 0.45	Comments
-	Once upon a time, there was a little girl who lived in a village near the forest.	a time, a village near the forest, the forest, a little girl who lived in a village near the forest	a village near the forest, the poor Alice the forest	a village near the forest, the forest, a little girl who lived in a village near the forest	The statistical system incorrectly resolves Alice to be the little girl
3	I'm on my way to see Alice who lives through the forest, near the brook, Sophie replied.	Sophie, the way, the poor Alice, the forest	Sophie, the poor Alice, the forest	Sophie, Alan, the forest	Rule-based and statistical best
5	It's me, Sophie.	a cackly voice, Sophie, Alan	Sophie, me, it	Sophie	Neural system certainly more confident here, no longer incorrectly including the wolf.
6	Oh, I just have touch of a cold, squeaked Alan adding a cough at the end to prove the point.	Your	Sophie, a hungry Alan		The statistical system now recognises Alan.
7	But Alice! What big eyes you have, said Sophie.	big ears you have, Sophie, the poor Alice	Sophie, the poor Alice, you	Sophie, Poor Alice	All systems now correctly recognise both Sophie and Alice.
8	She ran across the room and through the door, shouting, "Help! Alan!" as loudly as she could.	Sophie, the door, Alan	Sophie, the door, a hungry Alan	Sophie, the door, Alan	The neural system now correctly recognises both Sophie and Alan (the wolf).

Table 3.4: Comparison of Stanford coreference resolution implementations on the Little Red Riding Hood text with character names replaced with real names. We display each sentence along with the mentions identified within that sentence by each of the systems. Each mention presented in the table is the *representative mention* identified by the system as the most representative of the entities mentioned in the current sentence and the sentence numbers correspond to those of Table 3.3 for comparison.

NLP Task	Execution Time (Seconds)
POS, Lemma, Shallow Parse	10
NER	112
Stanford Parse	4
SRL	33

Table 3.5: Profiling results of each NLP task in the Illinois NLP pipeline when executed over a short example news article consisting of just *6* sentences.

3.3.2 Named Entity Recognition

Named entity recognition was another area of potential value: providing information that could be useful in identifying the set of actors to include in the resulting visualisation, as well as a range of other information that could prove useful in enhancing the resulting infographic produced. However, our initial experimentation with the Stanford NER online demo[1] suggests less promising results in our domain of text. As we can see in Figure 3.1, the system does not handle fictional names very well, failing to recognise *Goldilocks* as a person, and also missing *the forest* as a location we'd potentially like to pick out. As suspected, the fictional nature of our chosen domain does not lend itself well to largely fact-based tasks.

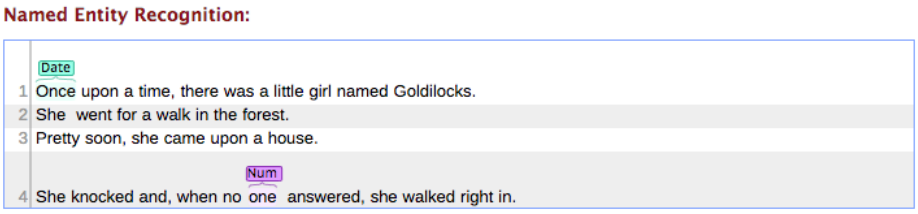


Figure 3.1: Named entities recognised by the Stanford CoreNLP NER tool.

In addition, our basic profiling results of the performance of each component of the Illinois NLP pipeline shown in Table 3.5 revealed NER to be the most time-intensive process of the pipeline. Thus, this seemingly resource-hungry process does not yield sufficient value to warrant inclusion in our resulting pipeline at present. A simpler method to begin with is to simply require the user to specify any actor names in the text on input, with the automation of this process left as a later task.

[1] Available at nlp.stanford.edu:8080/ner/

```
<TEXT>
John was a farmer, with a pet dog Rusky, and his wife Caroline. He was <EVENT class="OCCURRENCE" eid="e1">plowing</EVENT>
the fields when Rusky suddenly <EVENT class="ASPECTUAL" eid="e2">started</EVENT> <EVENT class="OCCURRENCE" eid="e3">barking<
/EVENT>. "Quick! <EVENT class="OCCURRENCE" eid="e4">Come</EVENT> over here!" <EVENT class="OCCURRENCE" eid="e5">cried</EVENT
> Caroline. John <EVENT class="OCCURRENCE" eid="e6">ran</EVENT> over to the barn to <EVENT class="OCCURRENCE" eid="e7">find<
/EVENT> Rusky had had 8 newborn pups. George, the rival farmer, <EVENT class="OCCURRENCE" eid="e8">waved</EVENT> at John.
"Why don't you <EVENT class="OCCURRENCE" eid="e9">come</EVENT> over here?" <EVENT class="I_ACTION" eid="e10">suggested</
EVENT> George.
</TEXT>
```

Figure 3.2: The resulting *event* annotations produced by TIPSem on a short example narrative.

3.3.3 Event Extraction

We finally explored the potential of TIPSem: the winning entry of the Temp-Eval-2 shared task on event extraction and temporal relation recognition. As discussed before, this tool does not really follow our definition of an event but instead focusses on the task of information retrieval, identifying event triggers in the text based upon the TimeML definition of an event as *"something that happens/occurs or a state that holds true, which can be expressed by a verb, a noun, an adjective, as well as a nominalization either from verbs or adjectives"* [6]. The system additionally categorises the event triggers identified into one of 7 *classes* listed in [6].

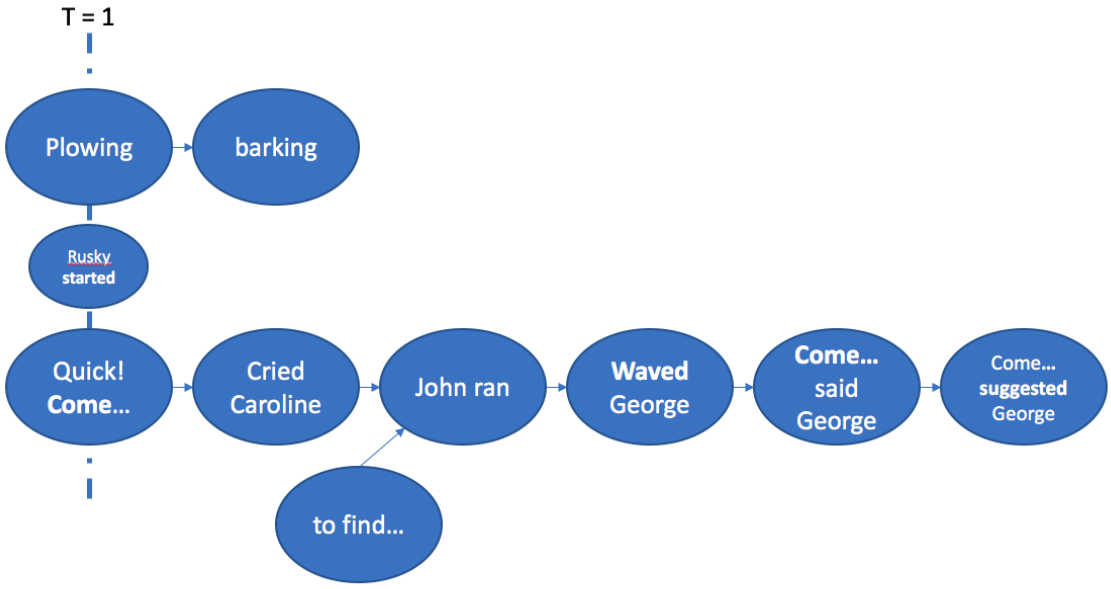
Our brief experimentation showed the tool to perform well, with Figure 3.2 highlighting the results of event extraction on a simple narrative example. This example shows how closely the TimeML definition of an *event* sits to matching the definition of a verb, however the additional classification performed by TIPSem provides information at a higher level that can then be used to further distinguish between different types of occurrence.

Interestingly, TIPSem also performs relatively well at the task of identifying the temporal relationship between events. Figure 3.3a shows the temporal relationships identified between the events found in the example text of Figure 3.2. Figure 3.3b illustrates these relationships graphically, showing the system to have actually performed fairly well. For example, we see the system correctly identifies that John was *plowing* the field when at the same time Rusky *started* barking. We then see the majority of the remaining ordering is correct, essentially following the written order of events. Again, in our domain of narrative texts we can largely assume that events follow their written order, however these results highlight the potential of this tool in later extending our application to other domains where this assumption may not be so true.

Thus, TIPSem shows clear promise in the task of identifying key occurrences within the text, potentially allowing us to link sentences that refer to the same or a similar *event*, and additionally enabling us to discriminate between the different *classes* of event. As we'll see later, repeated mentions of an *occurrence* event, such as a character "running", is likely a more significant piece of information than two sentences both including a *reporting* event such as, "said". Our initial experimentation also suggests the potential use of the tool in identifying the temporal relationship between the events extracted.

```
<TLINK lid="l1" relType="BEFORE" eventInstanceID="ei1" relatedToTime="t0" />
<TLINK lid="l2" relType="INCLUDES" eventInstanceID="ei2" relatedToEventInstance="ei1" />
<TLINK lid="l3" relType="AFTER" eventInstanceID="ei3" relatedToEventInstance="ei1" />
<TLINK lid="l4" relType="BEFORE" eventInstanceID="ei4" relatedToTime="t0" />
<TLINK lid="l5" relType="SIMULTANEOUS" eventInstanceID="ei1" relatedToEventInstance="ei4" />
<TLINK lid="l6" relType="BEFORE" eventInstanceID="ei5" relatedToTime="t0" />
<TLINK lid="l7" relType="BEFORE" eventInstanceID="ei4" relatedToEventInstance="ei5" />
<TLINK lid="l8" relType="BEFORE" eventInstanceID="ei6" relatedToTime="t0" />
<TLINK lid="l9" relType="BEFORE" eventInstanceID="ei5" relatedToEventInstance="ei6" />
<TLINK lid="l10" relType="BEFORE" eventInstanceID="ei7" relatedToEventInstance="ei6" />
<TLINK lid="l11" relType="BEFORE" eventInstanceID="ei8" relatedToTime="t0" />
<TLINK lid="l12" relType="BEFORE" eventInstanceID="ei6" relatedToEventInstance="ei8" />
<TLINK lid="l13" relType="IS_INCLUDED" eventInstanceID="ei9" relatedToTime="t0" />
<TLINK lid="l14" relType="BEFORE" eventInstanceID="ei8" relatedToEventInstance="ei9" />
<TLINK lid="l15" relType="BEFORE" eventInstanceID="ei10" relatedToTime="t0" />
<TLINK lid="l16" relType="BEFORE" eventInstanceID="ei9" relatedToEventInstance="ei10" />
```

(a) The temporal relationships identified by TIPSem between the *events* identified in Figure 3.2.



(b) Graphical illustration of the temporal relationships identified by TIPSem in (a).

Figure 3.3

3.4 Selected tools and technologies

The primary development language for this project is Java. The reason for this is two-fold: firstly, Java is an object-oriented language that provides nice automatic memory management and static typing properties, helping to ensure correctness of the implementation. In addition, a large number of Java libraries exist for the basic utilities to aid development, and both the Stanford CoreNLP and Illinois NLP pipeline tools that I have chosen for the core of this application both provide Java API's, making this the natural fit for the application.

We make use of Apache's Maven[2] as the build system and dependency manager of the project, choosing this tool for a number of reasons. Firstly, the large majority of popular Java libraries are available as Maven dependencies via the Maven repository, including both the Stanford CoreNLP library and the Illinois NLP pipeline. This greatly simplifies our dependency management, making it much easier to install and pick up development of the application on any machine with the Java JDK installed: all we need is our source code which includes the Maven pom.xml configuration file, and this defines all the system dependencies and where to get them. Additionally, when complete the entire application can be automatically packaged into a single JAR, where we also employ the use of the Maven Shade plugin[3] to ensure that all additional Maven dependencies are bundled into a single resulting JAR when packaging the final application. This maximises portability by simply requiring the movement of only a single JAR file to any Java 8 capable machine in order to execute the application on that machine. Of course, any of the remaining few external dependencies outside of Maven must also be installed separately; these are listed in Table 3.6.

For the fundamental text annotations, such as part-of-speech and initial sentence extraction, we chose to employ the Illinois NLP Pipeline. While both the Stanford CoreNLP and Illinois NLP Pipeline provide these basic annotations and perform similarly, we opted for the latter due to its incorporation of the Illinois Semantic Role Labeller should we later wish to introduce this to the application. However, we do also incorporate the Stanford CoreNLP library for the use of the coreference annotators discussed previously. Additionally, we incorporate the TIPSem tool for event extraction: a dependency that is not available via Maven.

Table 3.6 below summarises all our system dependencies, including the additional general purpose libraries for tasks such as testing and logging. Throughout the project we have maintained a test-suite to validate the basic operation of the various components of the system and provide a safety net as we introduced various changes to the system over time. All unit tests are stored in the `tests` directory of our source code, in a package hierarchy that mirrors the `src` itself. However, while unit tests prove useful in verifying the core system behaviour expected, logging during real-world usage has proven invaluable in enabling the quick diagnosis of unexpected problems in practice that were missed under the more controlled testing environment.

[2] Available at https://maven.apache.org/

[3] More information at https://maven.apache.org/plugins/maven-shade-plugin/

Dependency	**Use**	**Available through Maven**
Illinois NLP Pipeline	Basic text annotation such as POS and sentence identification.	Yes
Stanford CoreNLP	Coreference annotation.	Yes
Log4J2	Logging library.	Yes
JUnit	Unit testing.	Yes
Mockito	Mocking object components during testing with JUnit.	Yes
System Rules	JUnit library for defining expected system behaviour. Available from `http://stefanbirkner.github.io/system-rules/`.	Yes
TIPSem	Event trigger recognition and classification. Set-up instructions available from `https://github.com/hllorens/otip`.	No
GraphViz (Dot)	Drawing the basic timeline (Later dropped in favour of a web-based front-end).	No

Table 3.6: Project dependencies, a brief description of their use, and whether or not they're available via Maven.

Chapter 4

Adjustable Event Identification via Hierarchical Clustering

4.1 Intuition

In this chapter we present the development of a modified hierarchical clustering method for the identification of distinct events within narrative text, discussing our investigation and results over a number of example narrative texts. The objective being to develop a method to *automatically* identify an intuitive set of events from a narrative text at an *adjustable* level of detail, following our definition of an event outlined in Chapter 3.

Intuitively, we wish to to group the given input text into a set of disjoint collections of related sentences that each discuss a particular topic, involve a particular item, or concern the same subset of actors. As opposed to taking a bag-of-words approach to relating sentences, our concern lies somewhat more in the *semantics* of what is being discussed. Thus, we instead select a set of *features* to extract from the text that are subsequently used to evaluate the relationship between any two sentences. The use of specific features allows us to tune a weighting scheme that reflects the significance of each feature in determining the relationship between two sentences, and hence whether they both refer to the same event.

In contrast to typical clustering problems, we also respect the constraint of maintaining the textual order of the sentences clustered. That is, we wish to group sentences that are contiguous in the text in order to maintain the temporal nature of the events extracted. For example, in the text of Goldilocks (provided in Appendix A) we do not wish to simply combine *all* the sentences in the text that concern *the porridge*, as we'd sacrifice the temporal relationship between Goldilocks *initially* trying each bowl of *porridge*, and the three bears *later* finding their *porridge* has been eaten.

Our initial analysis, experimentation and method development follows with the final configuration listed in Section 4.5.

4.2 Initial Investigation

We initially focus on the example of Goldilocks. The short nature and repetitive use of language in this text make it quite apparent to see how we'd like such a text to be partitioned into a sequence of events. Figure 4.1 shows our initial analysis of this text, highlighting the potential feature matches between the sentences in this text, and where we'd naturally draw boundaries between the distinct events described.

Figure 4.1: Initial analysis of the Goldilocks story. **Yellow** highlighting reflects common features between nearby sentences, **green** highlighting reflects other features that could potentially be used as features, and **blue** highlighting reflects another possible overlap between sentences. The horizontal lines are the suggested event boundaries based upon this manual analysis.

Sentence/Feature	Goldilocks	ran	eat	sat	chair	porridge	bowl	tired
1	x					x	x	
2	x					x		
3	x					x	x	
4	x					x		
5	x					x	x	
6	x							x
7	x			x	x			
8	x				x			
9	x			x	x			

Table 4.1: Feature vectors from the Goldilocks text analysis in Figure 4.1, highlighting the clear boundaries between distinct events in the text.

It is quite apparent from this example that there are a number of features that could be used to relate the sentences in this text, revealing a fairly distinct set of events. In particular, the features highlighted in yellow provide literal matches in the text, either linking the sentences based upon the actor mention (in this case being *Goldilocks*) or a particular object recurring throughout a sequence of sentences. In this case, we see the strong relationship around *Goldilocks'* interaction with *the porridge* before then moving on to try each of *the chairs*. Additionally, the features highlighted in green reflect other more abstract means of relating each of these sentences to one another. For example, the fact that Goldilocks was *hungry* relates closely to her subsequently *tasting* the first bowl of porridge. This semantic relationship is certainly something to be explored further. We also see other relationships such as the use of antonyms, describing an the porridge as *hot* before subsequently finding another bowl too *cold*. This may or may not be a feature of relating text that generalises to other contexts, and requires further exploration. Lastly, the features highlighted blue represent additional features that could perhaps link two sentences, in this example using the relationship between *eaten* and *ate* which are clearly the same action despite not being syntactically expressed in the same manner. This text continues similarly, proving to be a good example with quite obvious boundaries between distinct occurrences. The remainder of this annotated text is included in Appendix B.1.

Table 4.1 further clarifies how we could employ these features to divide the text into a set of *events*, obviating the boundaries between the events identified above. While the example of Goldilocks may seem trivial, further investigation revealed the repetitive use of language as a common trait of a lot of narrative texts; even in the case of our short extract from Harry Potter and the Philosopher's Stone, shown in Appendix A, we see that objects such as *the map* are repeatedly mentioned during what we'd consider to be a particular event surrounding Mr. Dursley's altercation with a cat reading a map.

An initial Python implementation of hierarchical clustering using the key words we manually identified from the Goldilocks text as input confirmed the potential of such an approach, yielding the results shown in Figure 4.2. This test implementation employed single-link clustering and assigns each word match between sentences an equal weighting of 1. In this case, the sentence features were manually provided with any in insignificant phrases from each of the sentences, such as *the* and *of*, removed. Thus, we continued to explore this promising approach.

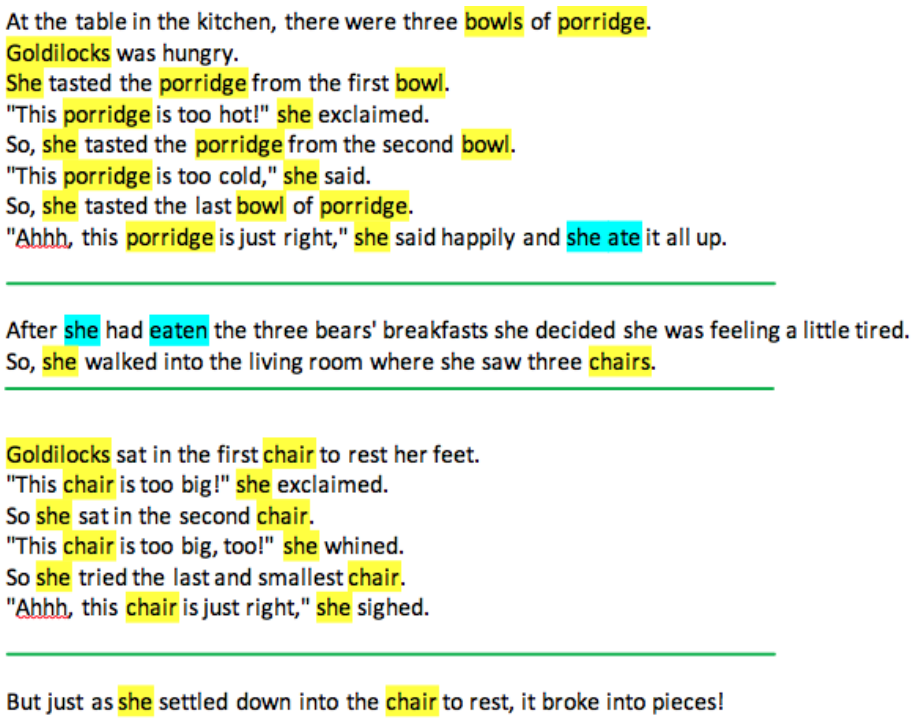


Figure 4.2: The resulting clusters produced using single-link agglomerative hierarchical clustering on the example of Goldilocks, where the horizontal lines highlight the boundaries between distinct clusters. This cluster set was taken at the time step in the clustering process that yielded results most similar to those identified previously in Figure 4.1.

4.3 Feature Selection

As a result of this analysis, we chose to employ the features listed in Table 4.2 in order to determine the strength of relationship between any two sentences, and discuss our motivation for each below.

Coreference resolution can be used to find all **coreferent mentions** of any entities within a text, be it *people* or *objects* that are repeatedly referred to. By nature of the analysis, coreference mentions are phrases that are mentioned more than once, so will certainly contribute to relating two or more sentences together and thus provide a useful feature to include. We distinguish between actor-specific mentions and other mentions to allow for the assignment of different weights for each feature. For example, while actor coreference is important, it is perhaps more significant to group the text based on the topic of discussion; in the particular case of *Goldilocks and the three bears*, we see that Goldilocks is mentioned in almost every sentence however we'd really like to group sentences based upon the repeated mention of common topics such as *the porridge* or *the chairs*.

Feature	Method of extraction
Coreference mentions	Using the Stanford CoreNLP coreference resolution tool.
Actor mentions	Using the Stanford CoreNLP coreference resolution tool along with the user-provided character list.
Nouns	Using the Illinois NLP pipeline POS tagger to extract all noun phrases.
Adjectives	Using the Illinois NLP pipeline POS tagger to extract all adjectives.
Cardinal numbers	Using the Illinois NLP pipeline POS tagger to extract all cardinal numbers.
Speech	Simple syntax analysis.
TimeML EVENTs	Using the TIPSem event extractor.

Table 4.2: Selected sentence features and method of obtaining this information.

Nouns were employed as a feature to recognise any repeated references to objects that are not captured by the coreference tool. The aim being to pick out particular *items* that may be repeatedly referred to in the text. For example, in the case of our example Harry Potter text we have repeated mentions of the *drills*, the *street*, and even in the initial description of both Mr. and Mrs. Dursley who both have quite distinctive *necks*. While coreference resolution performs well on recognising named mentions, and in a large number of cases, nominal mentions, it is not designed to extract repeated mentions of other objects in text. An alternative considered was the use of dependency parsing to identify the *objects* and *subjects* of interaction, providing an additional level of information above that of simply identifying nouns. However, the inaccuracy of dependency parsers makes them not entirely reliable, and even if they were it is common for an item to be mentioned as both the subject and the object across different sentences. For example, in the sentence below we see *the ball* used as both the object and the subject in the two sentences. All we wish to capture is the fact that *the ball* is the item of interest in both sentences: a result that can be achieved through the identification of nouns.

> *The boy picked up the ball and threw it as hard as he could. The ball whizzed through the air until it came to a halt.*

Adjectives were selected based upon the typical role they play in fairy tales; we often see repetitive language in fair tales, with short sequences such as *Goldilocks'* interactions with the porridge where a similar description is employed for her experience with each of the three bowls of porridge: the *first*, *second*, and *last* bowl of porridge. Thus, this feature provides the means to distinguish between each of the individual objects being referred to. Of course, there is also the risk that similar adjectives may well be used to describe quite different objects, so this is a feature of far less significance than those previously discussed.

Similarly to the above, in the case of fairy tales there is often a significance in the number of items described. For example, in the case of Goldilocks there is a reason there are *three* bowls of porridge, *three* chairs in the living room, and *three* beds upstairs. Matches on these phrases may not be so significant initially, but as we reach a far coarser level of detail in the story we may wish to group events that are related by this attribute, potentially resulting in an event consisting of Goldilocks' interactions with each of the porridge, chairs and bed that would otherwise be linked by nothing but *Goldilocks* herself. If this were the case, it would be no more likely for *Goldilocks* interactions with the porridge to be clustered with her initially walking in the forest, however I'd argue her interactions with each of the three items (porridge, chairs, and bed) should be clustered into a single event, before that is merged with her initial walk in the forest. Of course, what we treat as an event is a largely subjective matter, and so this can be argued each way. Ultimately however, our goal is to produce a natural set of events where the user can understand why they have events grouped in such a way. As such, the use of **cardinal number** phrases is a feature that is shared by each of the events in Goldilocks, and thus strengthens the relationship between these three key events discussed.

Speech refers to whether or not a sentence contains speech. This again being a particularly narrative-specific feature. When reading a story, we typically see a mix of description followed by sequences of dialogue between characters. As we increase the coarseness of our summary of the events in the story, we'd typically summarise the dialogue into a single interaction between the characters involved. Thus, to encourage the collection of dialogue into a single event we introduce this as a feature. Two sentences that both include speech are more likely to be involved in a single dialogue than those where only one of them contains speech, in which case other features become more prominent.

Finally, we employ the use of TIPSem to obtain **TimeML EVENT tags**, as we saw in Section 3.3.3. These tags are particularly useful in identifying common actions that are repeated throughout a number of sentences. For example, in Goldilocks we see that she repeatedly *tastes* the bowls of porridge, before then *sitting* in the chairs, and finally *laying* in each of the three beds. An alternative, and perhaps less computationally expensive, approach to extracting this information could have been to extract the *verb* POS tags, as we do to extract nouns. However, while this succeeds in extracting the desired actions from the text, it also extracts a large number of less significant phrases that do not contribute any significant value. For example, the sentence below highlights in **bold** all the verbs that would be extracted by POS tagging:

*After she **had eaten** the three bears' breakfasts she **decided** she **was feeling** a little tired.*

Verbs such as *was* or *had* provide little value in relating two sentences around a particular occurrence or item. As a result, we'd need to filter the large number of possible verbs like this that add little value to relating multiple sentences before using these results. What we're really interested in are the actions and occurrences that take place according to the 7 classes of EVENT identified by the TimeML specification. That is, using the results of TIPSem, we identify the following EVENTs in the same example sentence as above:

*After she had **eaten** the three bears' breakfasts she **decided** she was **feeling** a little tired.*

Thus, we immediately obtain the information we desire and that is of greater significance in relating multiple sentences to one another. Additionally, the TIPSem EVENT tags provide us with some additional semantic information regarding the *class* of EVENT identified. In the example above, all of the identified phrases are categorised as *occurrences*, however phrases such as *said* are instead categorised as *reporting* phrases. This information provides an additional means to distinguish between the different types of action identified in the text, and adjust the significance of their contribution to the clustering process appropriately. Table 4.3 highlights how we rank the 7 classes of EVENT tag based upon their significance in relating multiple sentences.

Rank	**Event Class**	**Examples**
1	Occurrence	die, crash, build, merge, sell
1	State	on board, kidnapped, love, ..
2	Perception	See, hear, watch, feel.
2	Reporting	Say, report, announce
2	Aspectual	begin, finish, stop, continue
2	I_State	Believe, intend, want
2	I_Action	Attempt, try, promise, offer

Table 4.3: Ranking of the significance of each of the 7 TimeML EVENT tag classes in terms of their contribution to relating two sentences with rank 1 being most important. Examples provided as from [5].

This initial feature set is what we take into our hierarchical clustering algorithm to follow, providing a nice mix of both syntactic and semantic information to relate sentences to one another.

4.4 The Algorithm

4.4.1 Overview

Figure 4.3 illustrates the high level flow of the hierarchical clustering algorithm, which shall provide a useful reference during the following discussion of the various parameters of the algorithm that can be adjusted and the additional parameters introduced in our modified approach.

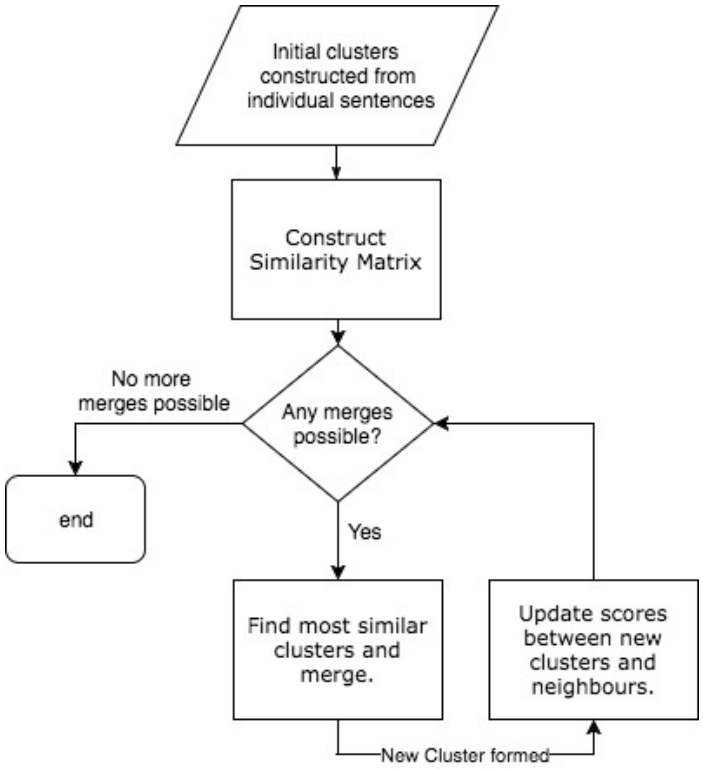


Figure 4.3: Flowchart showing the high level structure of the hierarchical clustering process.

The *similarity matrix* referred to in the figure is an efficient means of storing the similarity score between any 2 clusters, yielding a symmetric matrix where the value stored at index [i, j] represents the similarity score between the i^{th} and j^{th} clusters, and scores along the diagonal are set to 0. Our initial cluster set consists of an individual cluster for each sentence initially extracted, before we then employ agglomerative hierarchical clustering to recursively build up larger clusters.

4.4.2 Parameters and Modifications

Before presenting our modified clustering design and experimentation, we first provide a high-level description of the role that each parameter plays in the hierarchical clustering process. At the core of the algorithm sit the following 3 parameters:

- **Feature Weights**: *a weighting associated with each of the features previously discussed to adjust their contribution to the resulting score.*
- **Scoring Function**: *the function used to judge the similarity of sentences.*

- **Link Strategy**: *the method used to update the scores between clusters following a merge, as discussed in Section 2.2.3.*

Our first modification to the standard hierarchical clustering algorithm is the constraint that we only consider immediate neighbours within a small radius for clustering at each step. As discussed earlier, we wish to maintain the temporal nature of the input text when clustering by ensuring that we only merge contiguous sentences, enabling the resulting events to be plotted sequentially in the timeline. To accompany this modification, we introduce 2 new parameters:

- **Lookahead Distance**: *an integer specifying the maximum distance that a neighbouring cluster can be from the current cluster to be considered for clustering.*
- **Distance Discount Factor**: *a multiplicative ratio used to penalize the score between sentences as the distance between them increases.*

The **lookahead distance** defines an upper limit on the distance between neighbouring clusters that are allowed to be considered for a merge. For example, a value of 2 would mean cluster *3* can only be merged with one of clusters *1, 2, 4, or 5*, assuming clusters are ordered with respect to their textual ordering. Initially, when each cluster represents only a single sentence, this translates to a limit on the the number of *sentences* that can be merged into an event in any single step. Naturally, as the clustering process continues, this radius effectively expands. This parameter ensures that we obtain a *gradual* increase in the size of clusters as the process continues. For example, setting this to a value of 1 ensures that at any particular step we may only merge at most two contiguous clusters. Additionally, to preserve the temporal ordering of our event clusters, should clusters *1* and *3* merge into one, cluster *2* would also be consumed as part of this new cluster.

The **distance discount factor** was later introduced to encode our locality preference: the idea being to encourage perhaps less similar but nearby sentences to merge before more distant sentences merge. This avoids the effect of obtaining unbalanced events, where some highly related yet large clusters form while other lesser related but nearby sentences are left isolated, creating an unnatural imbalance between the generality of the events obtained. This is a multiplicative factor used to reduce the similarity score between neighbouring sentences as the textual distance[1] between them increases, resulting in a far more gradual and balanced progression, shown later in Section 4.6.4.

4.5 System Design

4.5.1 Architecture

Figure 4.4 shows the class structure of the `clustering` package created to encapsulate all the objects responsible for performing the hierarchical clustering of the events initially extracted. The system has been designed with flexibility in mind in order to allow any of the components of the clustering process to be swapped out for an alternative should a new potential approach come to light, or we decide that one approach is preferable to another.

[1] The number of sentences between the first sentence in each cluster.

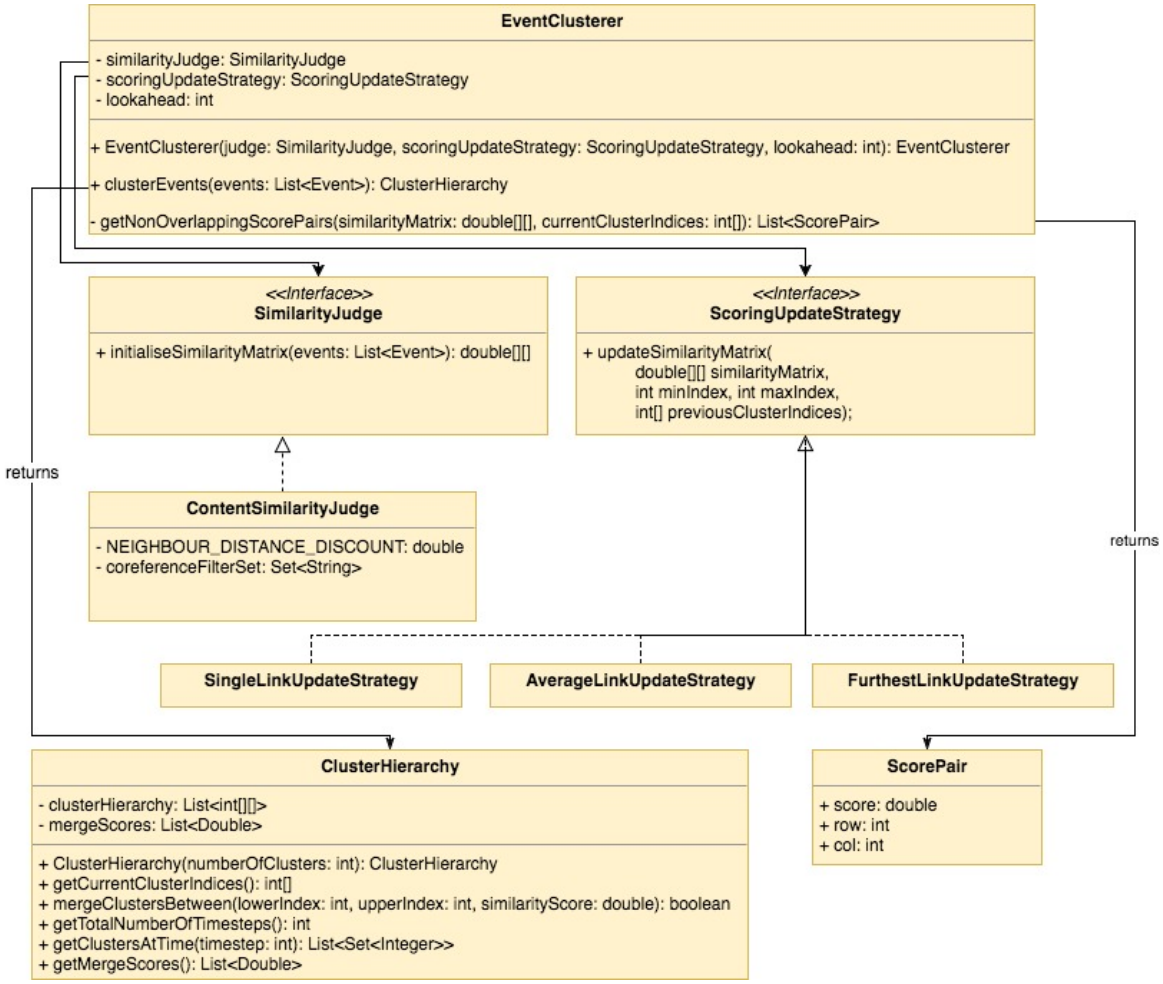


Figure 4.4: UML class diagram of the `clustering` package of our application, encapsulating all the classes involved in the hierarchical clustering process.

To achieve this, we have decomposed the algorithm into a number of distinct components, detailed shortly, making use of object composition to build the resulting system. This system has been designed largely to follow the open-closed design principle to improve maintainability and future extensibility.

Furthermore, the decomposition of each part of the algorithm into subcomponents greatly simplifies testing. We can first unit test each component in isolation before performing larger scale integration testing of all the components working together. Our *test* source directory reflects our *src* directory using the recommended Maven project structure. Thus, all clustering unit tests are also contained within the `clustering` package in the *test* source directory. This is where we employ JUnit for unit testing, and the Mockito library to allow us to construct mock objects for testing. For example, we make use of this tool in order to inject a mock `SimilarityJudge` into the `EventClusterer` during testing.

4.5.2 Components

EventClusterer This is the top-level object responsible for coordinating the hierarchical clustering process outlined in Figure 4.3. Once configured with a `SimilarityJudge`, `ScoringUpdateStrategy`, and `lookahead`, the system iteratively constructs the resulting `ClusterHierarchy` until there are no more non-zero scores in the similarity matrix.

SimilarityJudge This class defines the interface of the scoring function, used by the `EventClusterer` to construct the similarity matrix at the start of execution. In this case, we have only implemented one `SimilarityJudge` which employs the features discussed previously, and it is this object that considers the *distance discount factor* described earlier.

ScoringUpdateStrategy This class defines the interface of the link strategy to be employed by the `EventClusterer`. This abstraction allows us to easily introduce alternative strategies that, as we'll explore shortly, greatly adjust the dynamics of the resulting hierarchy.

ClusterHierarchy This is a high-level representation of the resulting cluster hierarchy produced by the `EventClusterer`. This component plays a hugely important role in maintaining a clear and efficient representation of the resulting hierarchy, with some of the finer implementation details discussed in Section 4.8. The `ClusterHierarchy` holds information regarding the cluster sets at each iteration of the process, the scores leading to each merge, and the indices of the current clusters in the similarity matrix.

ScorePair This is a simple abstraction to represent the position of a single similarity score within the similarity matrix. This is returned to the `EventClusterer` from the private inner method used to search for the next highest similarity score in the matrix, and thus the clusters that should be merged next.

4.6 Experimentation and Results

4.6.1 Scoring Functions Considered

One of the most important components of the algorithm is the scoring function used to calculate the similarity, or more accurately the strength of relationship, between all the sentences initially extracted from the text. In our investigation we consider the following 3 potential scoring functions:

1. An absolute scoring scheme
2. Relative option 1 - an additive combination of the relative feature scores
3. Relative option 2 - calculate the absolute score and divide by the maximum number of possible feature matches.

Our initial approach was to employ the absolute scoring scheme, calculating the number of matching features between any two sentences and taking this as our result. However, a major drawback of this approach immediately became apparent: larger sentences become more likely to be clustered due to the increased number of features contained within the sentence. This unfairly penalises short sentence and fails to reflect the natural approach taken by readers when summarising a piece of text.

We thus consider methods that incorporate a notion of *relative* similarity. Option 1 proposes the following definition:

$$\text{Score} = \Sigma_{i \in features}\left(\frac{weight[i] * numMatches(features[i])}{\text{Total possible features}[i] \text{ matches}}\right)$$

where $features[i]$ represents the i^{th} feature type from those identified previously, $weight[i]$ represents the corresponding feature weight, and *numMatches*() computes the number of feature matches between the two sentences under consideration. Thus, this scheme now considers how many feature matches there are relative to the maximum possible feature overlap, defined as the maximum number of distinct instances of features[i] in either of the two sentences. For example, if between 2 sentences they mention 3 different actors, then the maximum possible overlap would be 3. While this certainly yields an improvement over the absolute method, this scheme still has a problem. As we shall see shortly, the scoring function and the set of feature weights applied are closely related. Features of a higher score should impart a greater influence on the resulting score, however, the examples below illustrate how this function undermines the effect of feature weighting. In this example we assign *nouns* a weight of 1.9, and *adjectives* a weight of 1.

Feature overlap = 1 noun match
Maximum possible overlap: 2 nouns
Result = $1.9/2 = 0.95$

Feature overlap = 1 adjective match
Maximum possible overlap: 2 nouns, 1 adjective
Result = $1/1 + 0/2 = 1$

As a result, we see that while the first example has the more significant match, the result favours the second pairing. While this scheme considers the relative overlap between features, it fails to penalise missed feature matches. Thus, the final proposition fixes this by constructing a fraction with a single denominator, defined similarly as:

$$\text{Score} = \frac{\Sigma_{i \in features} weight[i] * numMatches(features[i])}{\text{Total possible feature matches}}$$

As a result, we now obtain the behaviour we desire with higher weighted features having a greater influence on the clustering results.

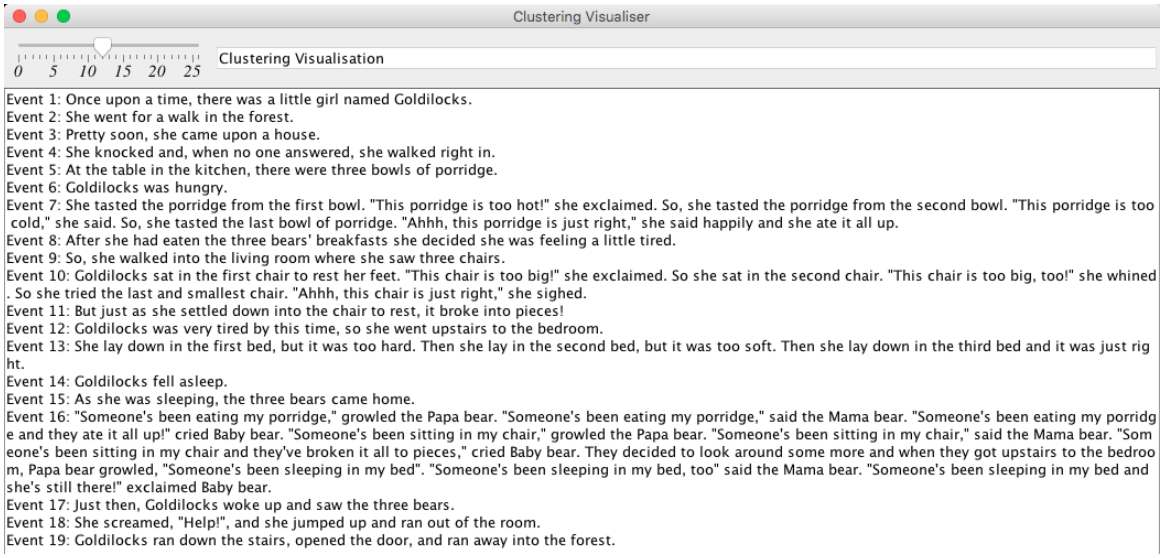


Figure 4.5: The results of hierarchical clustering with feature weights of 1 using the single-link clustering strategy.

4.6.2 Feature weights

The feature weights sit closely to the scoring function employed and serve to influence the ordering of the clustering that takes place. In this case, we wish to identify a set of weights that reflect the precedence of each feature when we naturally think about summarising text. In this case, our highest precedence goes to nouns and coreference mentions in order to group sentences based primarily on the topic of discussion, be it *porridge*, *chairs*, or *beds*. We then reflect on the actors involved and the narrative-specific consideration of dialogue, where we wish to keep a dialogue within a single event. Lastly, we consider any actions being carried out in the text according to our ranking of TimeML EVENT tags defined earlier.

Leaving all features with the equal weight of 1 using a single-link clustering strategy lead to a set of reasonable results, however failed to recognise the significance of the *topic* of discussion within each event. In the example of *Goldilocks*, this leads to the latter series of events involving each of the three bears merging into a single large cluster before we complete the merging of events surrounding the *porridge* and *chairs*. As a result, losing the balanced progression of event granularity we desire as we start to see what we'd deem to be more general events (i.e. event 16) emerge while we still have rough boundaries around smaller, more detailed events (i.e. event 10 and event 11). These results are shown in Figure 4.5.

Experimentation with various feature weightings revealed their limited impact on the results in comparison to the choice of link strategy, however they do make subtle differences on the resulting event boundaries identified. We additionally constrain our resulting parameter values to the range [1.0, 2), encoding the idea that two of any feature should always outweigh one of another. This ensures that while certain features may have more impact than others, ultimately the number of feature matches should be the most significant factor in determining the relationship between sentences. Table 4.4 lists the the feature weights assigned in the order of significance.

Feature	Weight
Coreference mentions	1.9
Nouns	1.9
Actor mentions	1.7
Speech	1.6
TimeML EVENTs (Occurrence and State)	1.3
Adjectives	1
Cardinal numbers	1
TimeML EVENTs (Other classes)	1

Table 4.4: Assigned feature weights.

4.6.3 Link Strategy

The link strategy plays a major part in determining the *dynamics* of the resulting hierarchy produced, and is one place where there is perhaps no obvious choice as to which approach will yield the most natural results. We thus present the results of our experimentation with each of the three common link strategies in the following paragraphs, using the example of *Goldilocks and the Three Bears* to demonstrate our results. Note that for this analysis we have replaced mentions of the *mama bear* and the *baby bear* in the text with the names *Caroline* and *Mary*, respectively. This is to aid the coreference system in distinguishing between each of the three bears and thus allow us to obtain more accurate results. This issue is later addressed more formally in Section 6.3.3.

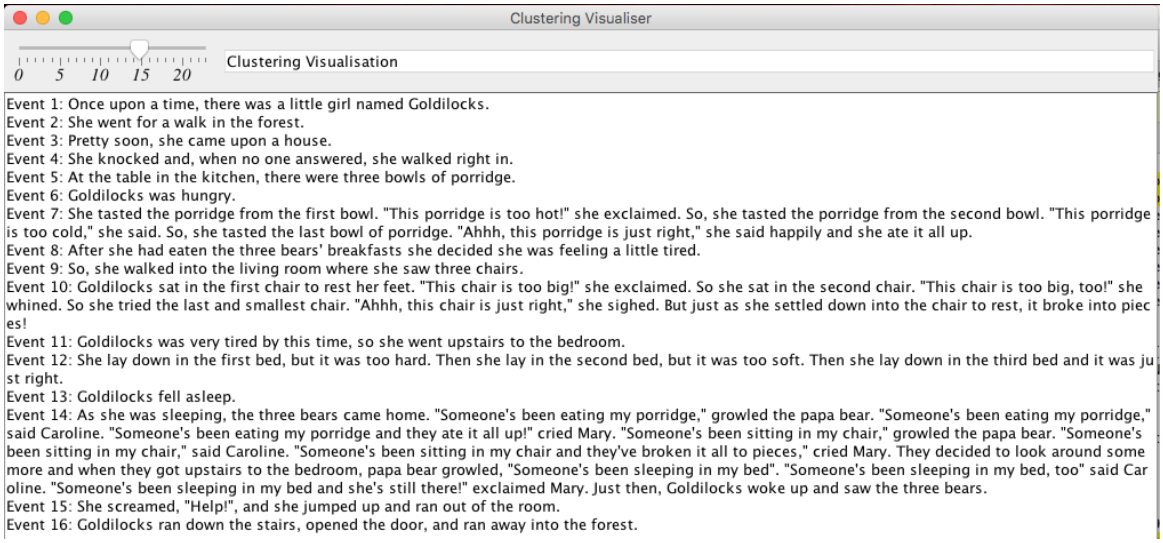
Figure 4.6 shows the results of the **single-link clustering** strategy. This approach yields a nice set of initial clusters with a clear distinction between each of the events, although event 14 forms fairly early, losing the distinction between the bears reactions to each of the *the porridge*, *the chairs*, and *the beds*. The clusters then begin to deteriorate as we approach the latter stages of the process with a black-hole effect starting to emerge. The strong relationship between the events that take place throughout the middle of this story lead to a single large event cluster forming here (event 8 in Figure 4.6b), while the peripheral event clusters remain as singletons. Unfortunately, resulting in an unbalanced and fairly unnatural set of events.

Average-link clustering overcomes this problem by instead updating similarity scores with the average similarity between the sentences in each of the clusters, avoiding the issue of only considering the strongest matches. As a result, we see a far more gradual increase in the size of clusters and a far more balanced set of results throughout the process. However, we still see that the event boundaries identified are not quite as we'd like. Figure 4.7a shows that while we start to see a nice set of

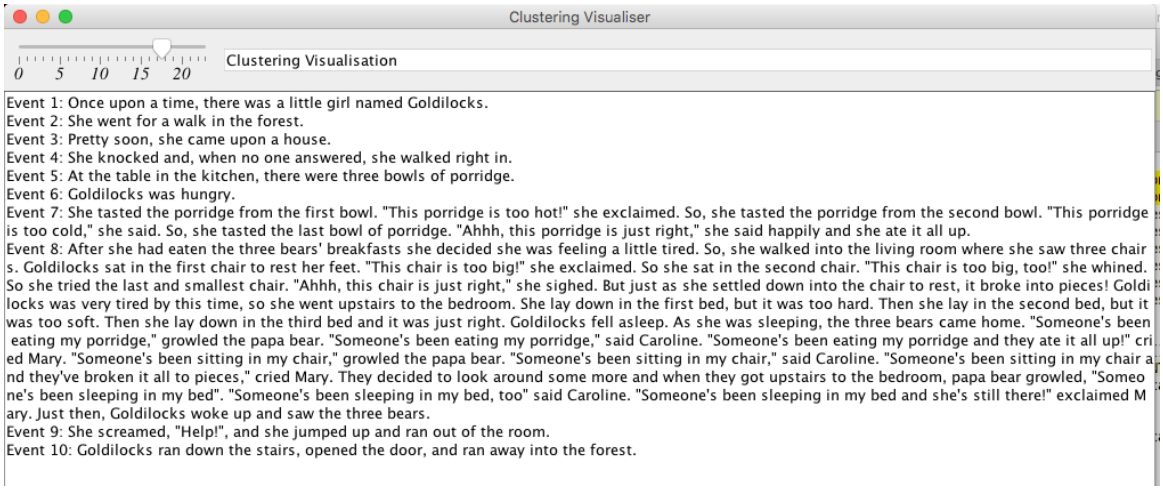
clusters emerge in general, event 18 forms relatively early placing *Mary's* reaction to both *the porridge* and *the chairs* in the same cluster. We can see the strong similarity between these reactions, explaining why this happens, however our desire is to cluster each of the sentences based on the *porridge*, before then later merging all of the bears reactions into a single event as the coarseness of clusters increases. Thus, average link clustering certainly provides an improvement over single-link clustering in terms of the balance of clusters identified but would require some further refinement in order to achieve the results desired.

Finally, we consider **furthest-link clustering**. This approach starts similarly to average-link clustering, developing a fairly balanced set of distinct clusters. Unfortunately, we again see Mary's reactions to both the porridge and the chairs cluster into event 10 in Figure 4.8a so require some further adjustment to refine this boundary. However, the nice facet of furthest-link clustering is that we see smaller clusters emerge before larger ones, maintaining both a balance and distinction between events. We avoid any black-hole effects by always considering the weakest relationship between neighbouring clusters, only merging them when they are the next most similar events. We also see that despite Mary's reactions being grouped relatively early in the process, we maintain a distinction between the other bears reactions to each of the topics even into the late stages of the process.

These results make it apparent that the average-link and furthest-link strategies provide the most promising results, with the balanced and progressive dynamics we desire. While both strategies yield somewhat similar results, the furthest-link approach appears to encourage more distinct boundaries between events, effectively avoiding any bloating of events as time progresses, and hence is our chosen strategy at present.

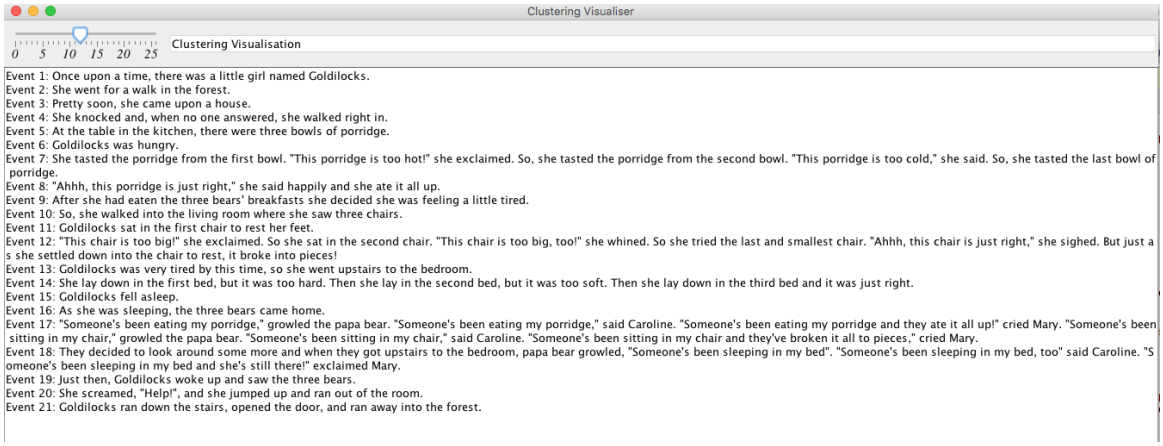


(a)

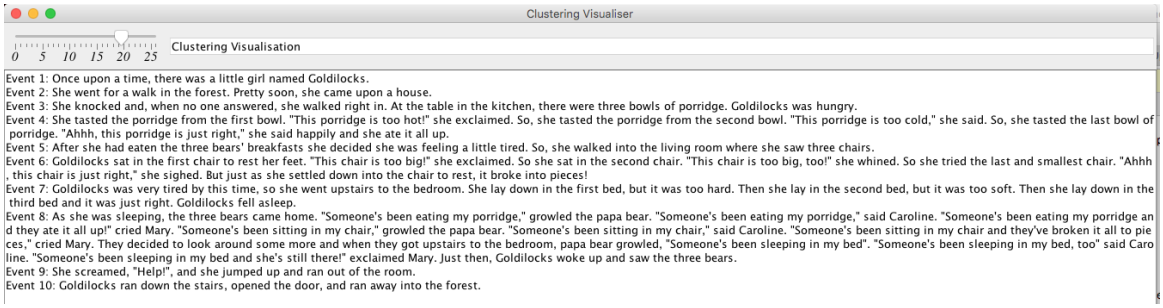


(b)

Figure 4.6: (a) The event clusters formed initially using the *single-link clustering* strategy. (b) The event clusters formed towards the final stages of the clustering process under the single-link clustering.

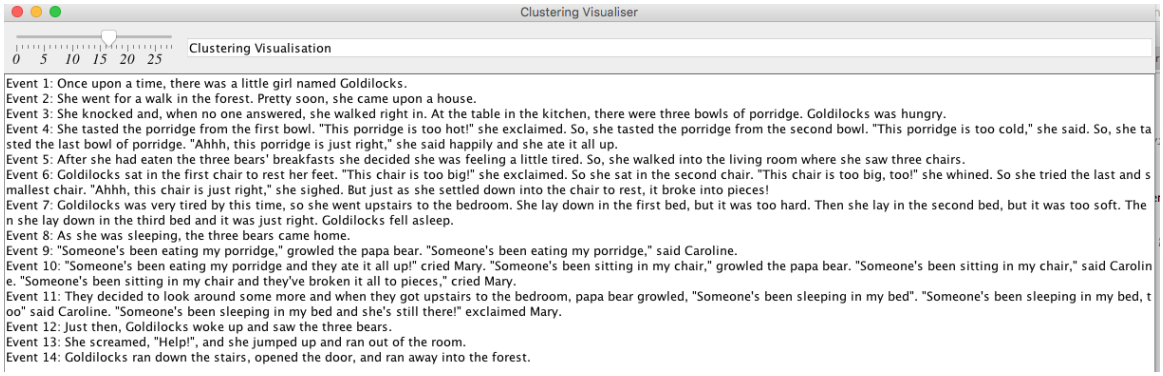


(a)

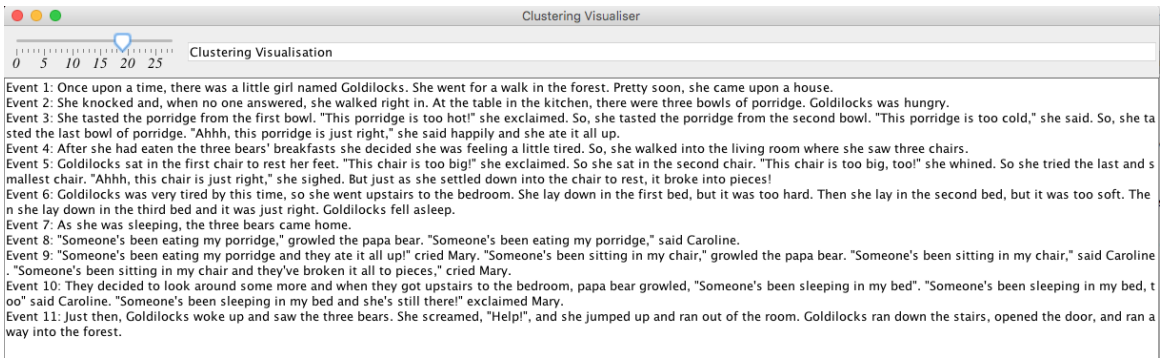


(b)

Figure 4.7: (a) The event clusters formed initially using the *average-link clustering* strategy. (b) The event clusters formed towards the final stages of the clustering process under the average-link clustering.



(a)



(b)

Figure 4.8: (a) The event clusters formed initially using the *furthest-link clustering* strategy. (b) The event clusters formed towards the final stages of the clustering process under the furthest-link clustering.

4.6.4 Distance Discount Factor

To improve the boundaries identified by the furthest-link clustering approach we introduce a **distance discount factor** into our scoring scheme as defined earlier, with the resulting definition being:

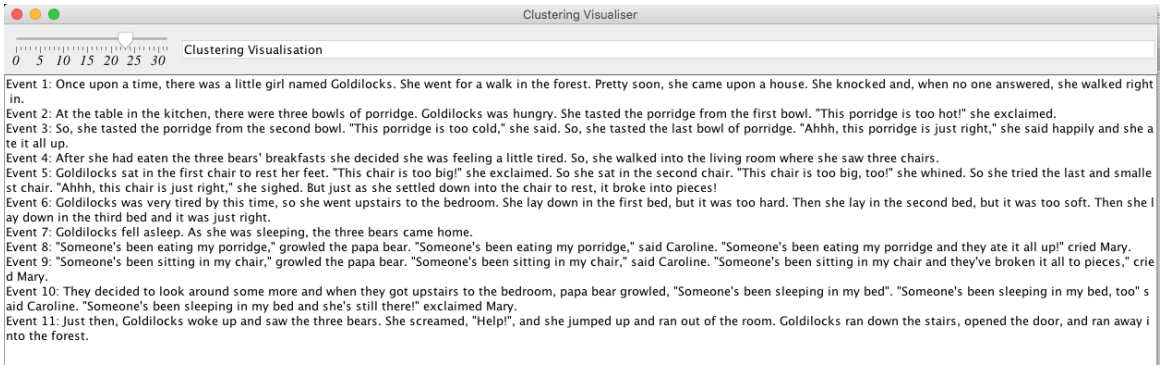
$$\text{Score} = d_f^n \times \frac{\Sigma_{i \in features} weight[i] * numMatches(features[i])}{\text{Total possible feature matches}}$$

where d_f is the discount factor, and n is the number of sentences between the first sentence in each cluster.

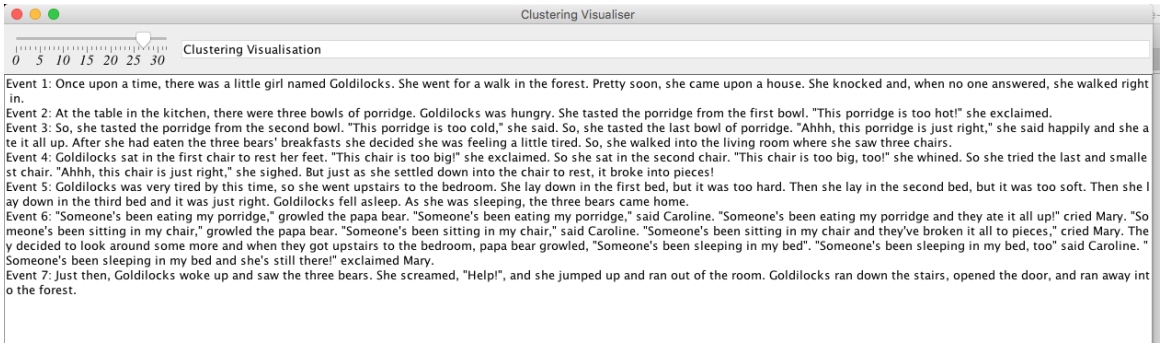
The aim of this parameter is to promote the value of *locality*: sentences that appear nearer to one another are more likely to be related than those further apart. Additionally, the incorporation of this parameter further encourages a gradual progression of event clusters regardless of our *lookahead* parameter defined as a hard limit to the number of clusters that can be merged in any single step.

An initial value of $\frac{2}{3}$ revealed the promise of the new scoring scheme, with Figure 4.9a showing each of the three bears reactions now being grouped into distinct events based upon the topic of discussion. However, as the procedure progresses we see that such a large discount has too much influence over the resulting merges, with events 2 and 3 in Figure 4.9b, both regarding Goldilocks' tasting of *the porridge*, failing to merge until the final stages of the process.

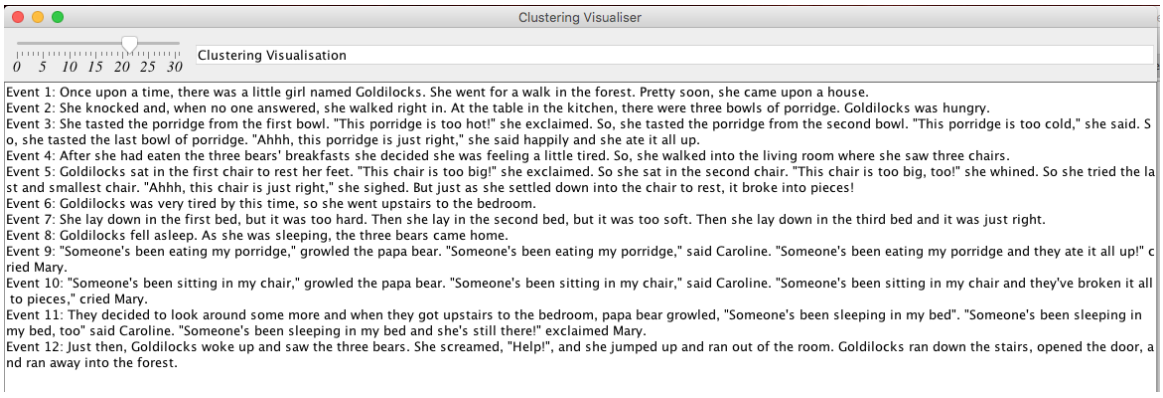
A far shallower discount of $\frac{4}{5}$ yields the final results we desire, shown in Figure 4.9c. We see a nice set of early clusters form with clear and natural event boundaries, while the properties of furthest-link clustering maintain this distinction between clusters even into the later stages of the clustering process, the results shown in Figure 4.9d. This configuration performs similarly well on our other example texts, with further results under this final configuration shown in our later case studies in Chapter 7.



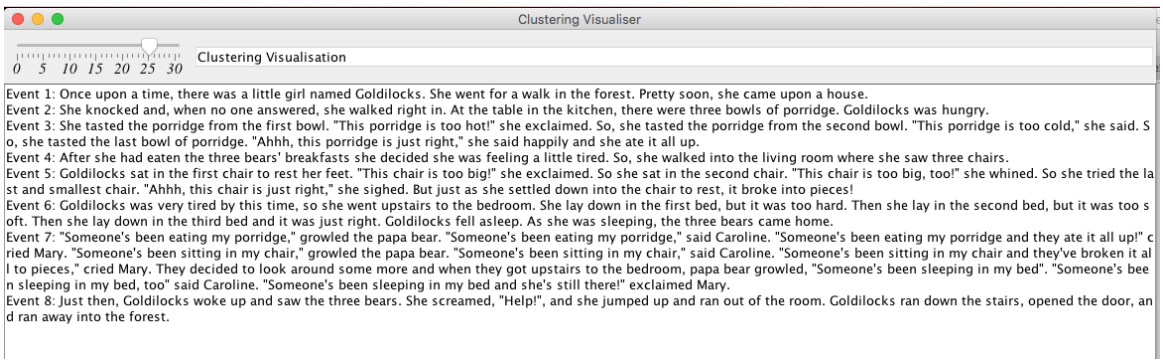
(a)



(b)



(c)



(d)

Figure 4.9: The results of furthest-link clustering with the incorporation of a distance discount factor. (a) and (b) present the results with a discount factor of $\frac{2}{3}$ at two points in the resulting cluster hierarchy. (c) and (d) present the results with a discount factor of $\frac{4}{5}$ at two points in the resulting cluster hierarchy.

4.6.5 Final Configuration

As a result of this experimentation, our final clustering configuration is listed in Table 4.5. Two key values of this table to clarify are the *lookahead* and the chosen *link strategy*. We did not discuss the effect of varying the lookahead, which in practice made little difference beyond a value of 3 due to the locality preference encoded in the distance discount factor. We thus maintained a value of 3 to again ensure a gradual progression of the size of event clusters and to avoid any large increases in cluster size within any single step. Secondly, Table 4.5 states that we employ a *modified* furthest-link clustering strategy. This reflects the small modification we have introduced when updating the scores between neighbouring clusters to ensure that we select the smallest *non-zero* score between neighbours as the new score. This avoids us prematurely ending the process with a distinct set of clusters when there is indeed still *some* relationship between the resulting clusters. Thus, this modification ensures that we will always continue merging clusters until there is absolutely no remaining feature overlap between neighbouring clusters.

Parameter	**Value**
Scoring function	$\text{Score} = {d_f}^n \times \frac{\Sigma_{i \in features} weight[i] \times numMatches(features[i])}{\text{Total possible feature matches}}$
Discount factor	$\frac{4}{5}$
Feature weights	As defined in Table 4.4
Link strategy	Modified furthest-link
Lookahead	3

Table 4.5: The final parameter configuration for our modified hierarchical clustering process.

4.7 Automatic Cluster Selection

As a result of the clustering process we obtain a hierarchical structure that allows the user to pan through each step in the hierarchy in order to find an event set at the level of detail to suit them. But as nice as this is, from a usability perspective having 30 or more options can be a little overwhelming. Thus, the challenge remains to develop a method to automatically identify a number of milestones in the cluster hierarchy where we have a good set of event clusters, each presenting a set of events at a different level of detail. The user need then only search within a radius of this benchmark to identify a clustering they're happy with. This process is more commonly known as *cutting* the hierarchy.

To identify such a method, we looked at a number of features of the clusters at each step in the process to see whether any distinctive pattern emerged that could be used to automatically determine when a good cluster set has been found, regardless of the input text. These features included the similarity score leading to the merge at the next time step, which clustering features contributed to each merge, and the distance between the sentences in the clusters that lead to the merge.

We repeated this manual analysis over the examples of *Goldilocks*, *Little Red Riding Hood*, *The Gingerbread Man*, and the first chapter of *Harry Potter and the Philosopher's Stone*, with the full results of this investigation provided in Appendix B.3.

While no pattern was immediately obvious, the results do highlight the two possible methods of identifying a good clustering, aligning with the two most commonly used *cutting* strategies:

1. Taking the point where we see a large change in the score leading to a merge.
2. Identifying a score threshold at which we make our cut.

Time step	**Score**	**% of initial score**	**Distance between merged clusters**
T = 0	0.48	100	
T = 11	0.138971429	28.95238095	3
T = 12	0.121904762	25.3968254	1
T = 13	0.118857143	24.76190476	2
T = 14	0.108571429	22.61904762	1
T = 15 (optimal fine-grained clustering)	0.076	15.83333333	2
T = 16	0.0608	12.66666667	2
T = 17	0.052869565	11.01449275	2
T = 18	0.035576686	7.411809524	6
T = 19 (optimal coarse-grained clustering)	0.027670756	5.764740741	6
T = 20	0.02125114	4.427320889	8
T = 21	0.005627888	1.172476704	11

Table 4.6: Table of scores contributing to the merging of clusters at each time step, and their magnitude relative to the initial maximum similarity score. This also highlights how the scope of sentences considered for merging increases as the process progresses.

Table 4.6 shows our results for the example of *The Gingerbread Man*, highlighting that both these approaches could yield similar results. For example, while we identify t = 19 as our desired optimal cluster set, we see that the sharp drop in score at t = 21 could also be used to identify a cluster set that is not far off this either. This is an artefact that is also evident in the other examples analysed. However, at present we have opted to take the latter approach. It is clear from the results of analysis that there are fairly similar score thresholds at which we identify our *optimal* cluster sets. These being at 20%, 10%, 5%, and 2.5% of the initial maximum similarity score between any two sentences, where we take the first cluster set with a merge score below this threshold as our cluster set. We thus treat these as our 4 default cluster sets, ranging from a set of highly-detailed, small events to a set of far larger, more coarsely defined events, respectively. Our investigation into the longer example of Harry Potter also revealed these thresholds to remain fairly stable regardless of the length of the input text.

4.8 Implementation Details

Throughout the development of this method, we have also introduced a number of optimisations, both for correctness and performance. We briefly discuss a number of these below.

4.8.1 Feature Filtering

In some cases we see feature duplication resulting in bloated similarity scores. For example, in sentences with the *papa bear*, we see him included as both a coreference tag and as the 2 nouns: *papa* and *bear*. As a result, contributing to the score of matches with other sentences involving *papa bear 3* times. To overcome this problem, we prioritise coreferential mentions of actors over any potential noun form, removing any of the constituent words of the character name from the noun phrases of the sentence.

4.8.2 Similarity Matrix Optimisation

The similarity matrix is the data structure at the core of the algorithm. This is an $n \times n$ matrix holding the scores between all pairs of clusters, where n is the initial number of sentences. Of course, as the clustering process progresses the number of clusters for comparison reduces. However, rather than reconstructing a smaller similarity matrix at each step, we instead re-use the same $n \times n$ matrix throughout. This removes the need for garbage collection of these structures, reducing memory consumption and improving the potential time performance if we can reduce amount of interruption from garbage collection.

When clusters are merged, we consider the new index of the cluster to be the index of the earliest sentence in the cluster. By maintaining an array of these active indices, we allow a far more efficient search of a clusters neighbours when considering the next merge as we need only check the scores of any *active* indices, mimicking the effect of having a much smaller matrix.

4.8.3 Clustering Acceleration

Another small performance optimisation takes place in our implementation of the `getNonOverlappingScorePairs` method of the `EventClusterer`. This method is responsible for identifying the next highest scoring pair of clusters that should be merged during the current iteration. However, rather than simply returning the indices of a single pair of event clusters to merge in the current iteration, this method instead returns the indices of *all* the pairs of clusters that are eligible for merging according to our current *lookahead distance* parameter in the current iteration. For example, if we find that clusters *1* and *2*, and clusters *3* and *4 both* obtain the same maximal similarity scores, then both of these pairs of merges shall be take place in the current iteration, potentially reducing the overall number of iterations in the process.

Additionally, to ensure correctness of the algorithm we are also careful to check that, for example, if clusters *1* and *2*, and clusters *2* and *3* are both to be merged in the current iteration, we perform the necessary reduction to a single transformation merging clusters *1* and *3*, automatically absorbing cluster *2* into the new cluster. This is necessary as we always merge clusters into the *lower* index. As such, if we were to merge cluster *2* into cluster *1*, the subsequent merge of cluster *3* into cluster *2* would fail since cluster *2* is no longer present. This issue could have been similarly overcome by imposing an ordering constraint on the resulting merges, however our alternative optimisation further reduces the number of operations that need take place. Finally, another consideration is to ensure that if we have the case where clusters *2* and *5*, and clusters *3* and *4* both obtain the same highest current similarity score, we make sure to perform the single merge of all clusters between *2* and *5* for similar reasons to the above.

To achieve this, the `getNonOverlappingScorePairs` function takes as input the current similarity matrix, along with the `currentIndices` array specifying the indices in the similarity matrix of the *active* clusters at the current iteration.

4.8.4 Event Cluster Representation

The `ClusterHierarchy` is the abstract representation of the resulting structure created by the clustering process. Internally, each cluster is stored as an array of integers, each representing the index of the event in the initial list of events provided to the `EventClusterer`. This reduces the memory overhead imposed, as there is no need to duplicate the `Event` pointers themselves during this process. This instead provides sufficient information to later construct the resulting events when necessary.

Chapter 5

Timeline Visualisation

5.1 Aims

With a good set of events now extracted from the input text, we shift our focus to constructing a clear infographic timeline representation of these events. In particular, we focus on highlighting two key properties of the text: the temporal ordering of the events identified in the text, and the dynamics of interactions between the characters involved in the text. Additionally, we aim to enhance the investigative capability of the resulting timeline by incorporating a degree of user interaction.

As a result, we hope to enable the user to see at a high level the interactions between characters in the text, identify the distinct storylines in the text, and to highlight the role of characters in the text: whether they make a relatively short-lived appearance, or can be identified as the main protagonist within the text. The incorporation of user interaction is to aid user exploration into the underlying text, as the typical drawback of static timeline visualisations is the inability to delve deeper into the timeline to better understand what is actually happening at any particular moment.

5.2 Design Objectives

To formalise our target result, we list below our key design objectives for the resulting timeline, inspired largely by the original example of Figure 1.1:

- The timeline should clearly reflect the sequential ordering of the events displayed.
- Event nodes should be positioned relative to the actors that are involved in that event.
- There should be a distinct path through the event nodes for each actor involved in the text.
- Paths should avoid overlapping where possible.
- The timeline should enable the discovery of potentially significant events.

5.3 Options Considered

As we discussed in Section 2.3, there are a number of different tools and approaches to timeline construction. Our early prototypes (shown in Appendix C.3) built using the GraphViz tool expressed the information desired, but lacked the clarity and expression of timelines like our original inspiration from the Lord of the Rings back in Figure 1.1.

While tools such as GraphViz offer a great deal of flexibility, they also present a number of difficult challenges that need to be overcome in order to provide the formal specification required to construct the desired layout. These include positioning nodes effectively such that we can avoid crossing arcs as much as possible, something that quickly degrades the clarity of the result, as well as spacing the arcs and nodes to avoid overlap and maximise clarity when tracing the paths of distinct actors through time. In addition to these requirements, we then have the need to maximise the *smoothness* of the resulting arcs to improve the aesthetics of the result and make it easier to follow.

Force-directed graphs, on the other hand, introduce an entirely different approach to graph drawing by employing the use of physical simulation to create emergent structures, and are increasingly being used for a number of data visualisation tasks. Using this approach, we instead encode our constraints as physical forces and then allow the physical simulation to find an optimal solution. This vastly simplifies the level of detail required in specifying the desired layout, moving us to a higher level of specification and providing surprisingly predictable and intuitive results.

Thus, the use of this technology provided the exciting opportunity to explore the potential of this approach in the novel context of timeline drawing. The following sections explore our use of this technique and the resulting timeline produced.

5.4 Using the Force

5.4.1 Selected Software

A number of existing applications and libraries exist for force-directed graph drawing. **Gephi**[1] is a data visualisation and exploration application that provides a number of tools to create graph representations of data and to also obtain a large number of graph metrics from the resulting graph, which may highlight particular properties of the underlying data. These metrics could prove extremely interesting in the context of alternative infographic visualisations of the underlying text, however are of limited applicability in the context of a timeline. In addition, Gephi provides relatively little opportunity to enforce custom constraints on the resulting graph.

We thus looked towards more flexible offerings that provide the opportunity to modify the resulting force-directed graph to a greater extent in order to achieve the end result we desire. For this, we considered two JavaScript libraries: **SigmaJS**[2] and **D3.js**[3]. The use of JavaScript enables us to

[1] Available from https://gephi.org/
[2] Available from http://sigmajs.org/
[3] Available from https://d3js.org/

integrate the resulting visualisation directly into a web application, and provides the means to later incorporate additional user-interaction with the resulting timeline. SigmaJS is a library dedicated to force-directed graph drawing and as such is tuned for this purpose. However, D3.js also performs similarly well, offers a great deal of flexibility, and has significantly more resources and examples available, making it far easier to pick up the intuitive API for this library. For this reason, we employ D3.js to construct our timeline.

5.4.2 Abusing the force

One of the key challenges of taking a force-directed approach is in balancing the emergent structure provided by this method with an underlying basic structure that we wish to impose on the result. We achieve this by taking advantage of the full control of the force-based simulation provided by D3, imposing a fairly rigid structure on the resulting timeline while allowing the simulation to then determine the few remaining details.

Force	**Description**
Link Distance	The target distance between nodes that are connected by an edge. This adjusts the properties of edges to try and match this value.
Link Strength	How "stiff" the edges in the graph are, making them more or less able to achieve the link distance specified and adjusting their influence on node positioning.
Node Charge	A force between nodes in the graph. Negative charges repel; positive charges attract.
Friction	A force acting against the movement of nodes.
Gravity	A force pulling nodes towards the centre of the canvas. (This is a global value)

Table 5.1: Description of each of the forces employed in the D3 Force Layout [64].

Table 5.1 provides a brief description of the forces at our disposal in the D3 library [64]. All of the forces listed here apply on a node-by-node or edge-by-edge basis as appropriate, with the exception of **gravity**. As a result, each node can have its own charge or friction based, for example, upon some property of the input data. Gravity on the other hand is a *global* force with the same influence over all nodes, designed to draw all nodes towards the centre of the canvas.

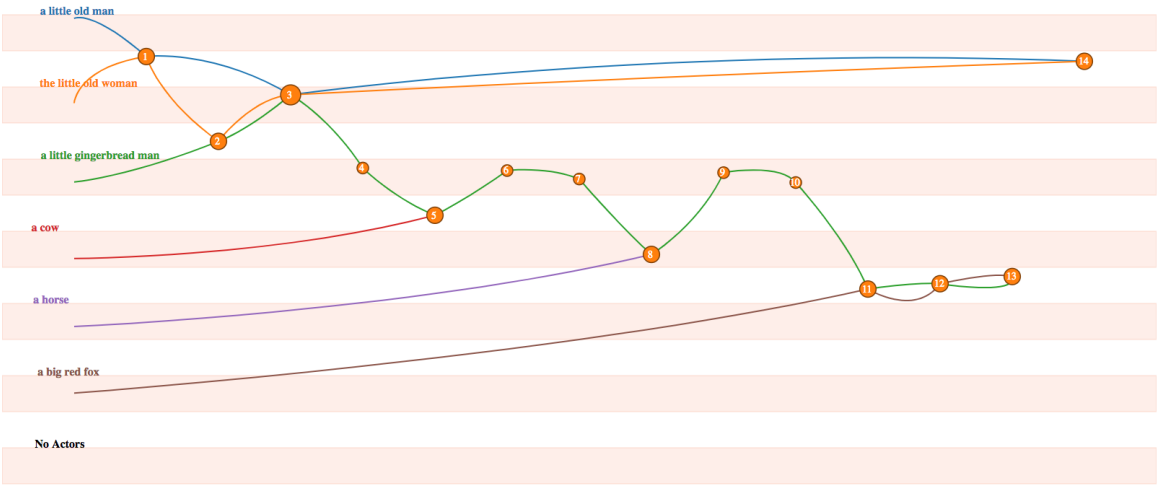


Figure 5.1: Our resulting timeline for the events identified in *The Gingerbread Man* story.

One of the most useful properties of force-directed graphs is that *forces are additive*. This means, for example, that if we had one node equally attracted to each of two surrounding nodes, it will end up sitting in equilibrium between these two nodes. This is an incredibly useful property that we take advantage of in laying out our resulting timeline.

5.5 Result and Features

Figure 5.1 shows the resulting timeline produced for *The Gingerbread Man* story, using a set of fine-grained events identified by our event clustering process. The exact set of events being displayed here are shown in Appendix C.2. From this timeline it is quite easy to see the interactions of the gingerbread man with each of the other prominent characters in the story, and how much of a role they play in the story. For example, it is clear that there is a short interaction between the *gingerbread man* and *the cow* and *the horse*, while *the fox* clearly plays a far more significant role in the sub-plot that unfolds towards the end of the story.

We can also see a nice separation between the paths of each character between events, avoiding any overlap while also producing fairly smooth paths through the timeline. The various properties of the plot are described below along with some of the key features of the plot, while the following section describes some of the implementation details as to how this result is achieved. Further examples are explored in our evaluation case studies in Chapter 7.

5.5.1 Node Properties

Each node in the graph represents a distinct event identified in the text according to our event clustering process; varying the point at which we take a cut of clusters from the cluster hierarchy will alter the resulting timeline, providing insights at a different level of detail. Figure 5.1 is regarded as a relatively fine-grained cluster set. The key properties of the nodes are:

- **Node Size** *reflects the number of characters involved in the event.*
- **Node Number** *represents the event number to aid visual clarity and make it easier to follow the events sequentially.*
- **Node X-Position** *reflects the temporal ordering of the event, with events evenly spaced along the x-axis in text order.*
- **Node Y-Position** *reflects the actors involved the event, sitting between the labels of the actors involved.*

5.5.2 Edge Properties

Edges represent the paths of each character through the story, with an edge present between any two consecutive events that involve the same character. The edge colour reflects the character whose path is being represented and corresponds to the colour of the character label on the left-hand side of the timeline.

5.5.3 Timeline Features

In addition to the basic plot, we have also incorporated a number of other features to aid exploration and understanding of the timeline.

Background Bars The horizontal orange bars in the plot background were introduced as a visual aid to improve the user's ability to recognise which actors are involved in each event based upon its y-position. As mentioned above, a node's y-position reflects the characters involved, and in the particular case that only one actor is involved in an event, the event node will sit along the y-axis of that actor. The orange bar is designed to help the user quickly trace back from an event to the actor(s) involved.

Node Hover Text The first element of user interaction is the ability to hover over an event to read the text and who's involved in that particular event. This aids user investigation and understanding to be able to delve deeper into the underlying text, and understand the details behind particular parts of the story highlighted by the plot. Figure 5.2 shows an example of this.

Path Highlighting The second element of user interaction is the ability to click on character labels to highlight only the paths of characters of interest in the timeline. This again is a feature to enhance the investigative capability of the timeline, making it far clearer *how* and *where* particular characters come into contact. Figure 5.2 shows an example of this, with the paths of *the gingerbread man* and *the fox* highlighted. This is a feature that becomes increasingly useful as the number of events and characters involved increases.

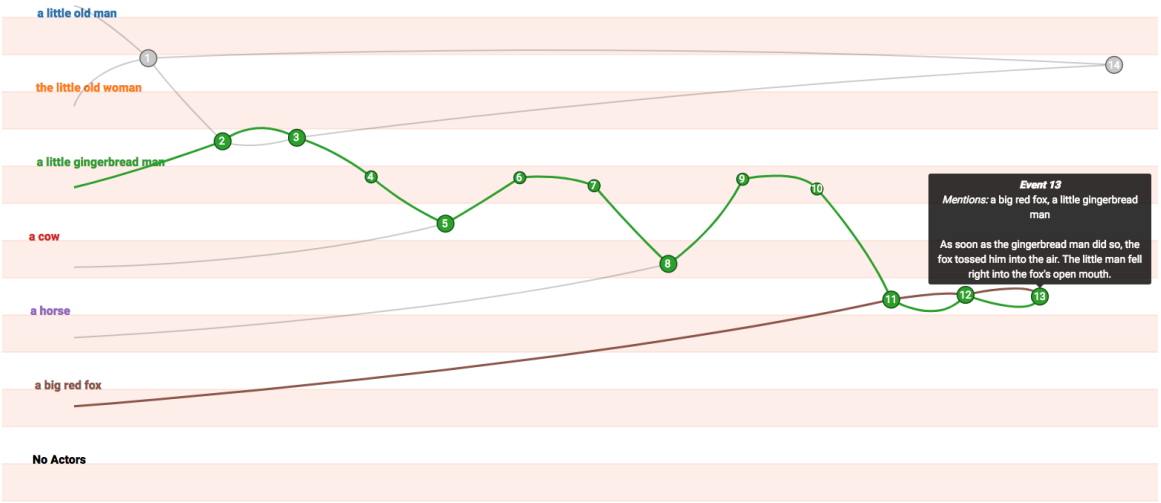


Figure 5.2: Demonstration of the ability to find out exactly what is happening in an event, and to highlight the paths of characters of interest, using the same example as Figure 5.1.

5.6 Implementation Details

5.6.1 Overview

Figure 5.3 illustrates the high level flow of the script that creates the timelines of the form shown above. The resulting timeline is built up on an SVG Canvas in the browser in two major stages.

Firstly, we construct and populate an SVG canvas with the nodes and edges to be displayed. In doing so we use D3's data binding mechanism to bind each SVG node and edge to its underlying JavaScript object that it represents. That is, nodes are bound to their associated *event* object, and edges are bound to their corresponding *edge* representation. The exact interface of these objects is defined shortly.

The second step is to then begin the D3 force simulation, passing the underlying *event* and *edge* data we previously bound to the elements of the SVG canvas as arguments. Having performed this set-up, D3 issues a callback to a *tick* event handler that we register after each step of the force-simulation, giving us the opportunity to update the position of nodes and edges on the canvas in response to changes in position of the underlying data in the simulation. As a result, the simulation eventually settles at equilibrium giving us our final timeline.

In the following sections we discuss some of the details of various aspects of this flow and the resulting timeline.

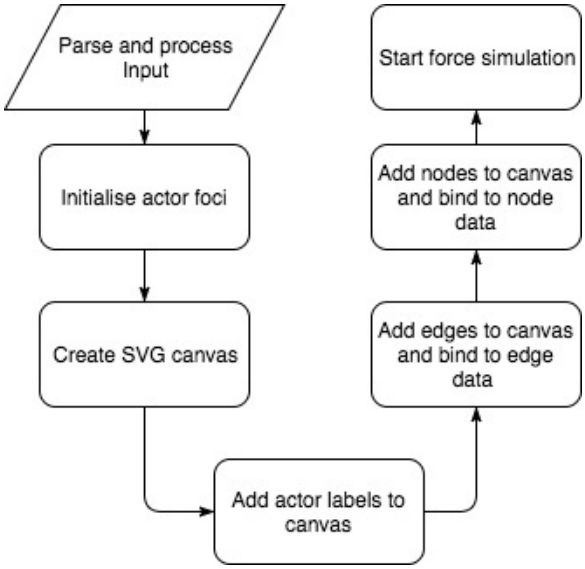


Figure 5.3: High level overview of the flow of the timeline drawing process.

5.6.2 Input Format

The underlying data consists of three components: *actor labels*, *event nodes*, and *edges*. This data provides sufficient information to construct the resulting timeline. Table 5.2 lists the expected object representation of each of these elements.

Component	**Object Representation**	**Description**
Actor Label	{ title: String, type: "label" }	The *title* defines the actor the label represents.
Event Node	{ eventNum: Number, text: String, actors: [String] }	The *eventNum* defines the sequential number of the event in text order, *text* is the event text, and *actors* is the list of actors mentioned in the event.
Edge	{ source: Number, target: Number, actor: String }	The *source* and *target* define the indexes of the source and target event node in the event node list, respectively. The *actor* is the actor whose path this edge represents.

Table 5.2: Object representations of the underlying data modelled by the timeline, and passed as input to the D3 Force Layout.

With the *actor labels*, *event nodes*, and *edges* defined as separate lists of objects, we then initialise the D3 Force Layout object with our list of *event nodes* and our list of *edges*. At this point, D3 then extends our object representations with some additional properties. Each node is given an `x` and `y` property defining its current position, while each edge representation has its `source` and `target` properties replaced with the actual *event node* objects referenced. These properties can then be used in our *tick* handler to update node and edge positions accordingly.

5.6.3 Basic Configuration

For our purposes, we define the following fundamental forces:

Force	Value
Link Distance	1
Link Strength	0
Node Charge	1
Friction	1
Gravity	0

By default, all values are set to 1. We first disable gravity as we do not wish to obtain a circular plot, which is the effect that gravity encourages. Also, as we impose a fairly rigid underlying structure on the timeline plot we do not have the concern of nodes going astray. Our only other initial change is to also reduce the link strength to 0; this is to ensure that we maintain our x-spacing regardless of the effect of links connecting neighbouring nodes. Thus a value of 0 means that links assert no influence over node positioning. By default, we impose a 100 pixel x-spacing between sequential events and a 100 pixel y-spacing between actor labels.

5.6.4 Actor Foci

One of the key challenges of constructing the timeline is in identifying a natural, aesthetically pleasing layout and is one of the key motivations for our use of force-directed graphs. In using this approach, we take advantage of the fact that *forces are additive* to position nodes vertically based upon the characters involved in the event. For example, we see event 1 in Figure 5.1 is positioned between the two labels for *the little man* and *the little woman*. As a result, we get a relatively smooth path between a series of events that involve largely the same set of characters, while getting a nice distinction between events involving disjoint sets of characters.

To achieve this we define a distinct y-axis focus for each character, stored in a map from character name to y-axis focus. During each *tick* update of the simulation, we then influence the y-position of each event based upon the actors involved. The exact update sequence for each node during a *tick* event is as follows in Algorithm 1, where k is a smoothing parameter:

Algorithm 1 Updating the y-position of an event based upon actors mentioned

```
1: for actor ∈ event actors do
2:     node.y ← node.y + (foci[actor] - node.y) * k;
3: end for
```

Thus, we essentially update the y-position based upon its current distance from the desired y-positions of each mentioned character's y-axis focus. The smoothing parameter, k, ensures that we *blend* the influence of all characters on the resulting y coordinate, and is defined as 0.1 times the `alpha` parameter used by D3.js to similarly blend the various forces acting in the simulation, gradually decaying over time until the simulation stops at equilibrium. This approach was inspired by the similar example use of categorical foci at `http://bl.ocks.org/syntagmatic/3991039`.

5.6.5 Leading Edges

As we can see in the example of Figure 5.1, we have a *leading edge* from each character label to the first event in which the character is mentioned. This makes it far easier to identify the first mention of an actor in the text, and is even more important for characters that appear in a single event. Without this edge, there would be no edges for these particular characters. This is a result achieved by augmenting our initial node set with a set of additional nodes for each actor with `eventNum = 0`. This is then used as flag to inform the drawing process to hide these nodes in the canvas, yielding the effect shown.

5.6.6 Curved Edges

Another challenge was in ensuring that multiple edges between any two nodes did not overlap while maintaining a good level of smoothness in the resulting timeline in order to maximise clarity. To achieve this we take an approach inspired by the example at `https://bl.ocks.org/mbostock/4600693`. In this case, we now pre-process our original edge data of the form described in Table 5.2 into a list of `bilinks` describing a path through three nodes: the *source*, *target*, and a new *intermediate* node. Each bi-link is then *bound* to an SVG `path` element in the canvas, which provides us with the flexibility to model a curve that passes through three defined coordinates.

The resulting effect is largely down to the influence of the newly introduced *intermediate* node that sits in the middle of the curve. While all event nodes are subject to the forces defined previously, intermediate nodes are subject to a different set of forces, defined in table 5.3. The link distance is increased to allow for the additional distance travelled by the curved edge, while the link strength is slightly increased to enable the intermediate nodes to ensure that intermediate node is anchored by the influence of the two nodes it's connected to, stopping it from wondering too far astray. Lastly, the most significant change is the increased magnitude of charge. This is responsible for ensuring that intermediate nodes *repel* one another, encouraging edges to curve away from one another and avoid overlap. From our experimentation, this configuration appears to yield a good balance between clarity and smoothness.

Force	Value
Link Distance	10
Link Strength	0.05
Node Charge	-80
Friction	1
Gravity	0

Table 5.3: The altered forces acting on the intermediate nodes along each edge.

5.6.7 Tooltip Text

To create the hover-over event text, we employed the simple yet effective Tipsy jQuery plugin[4]. In doing so, we slightly modified the library to work with the newer jQuery version 3.2.1 that we use instead of a slightly older dependency used originally. This simple library allows us to associate each node in the timeline with a tooltip that is displayed as soon as the user hovers over it.

[4] Available from https://github.com/jaz303/tipsy

Chapter 6

The Application

6.1 Aims

Our final aim surrounds the incorporation of the two components we've discussed in Chapter 4 and 5 into a fully functional and potentially marketable application. The ultimate aim being to construct an application that can provide practical value to its users despite current NLP tools not being 100% accurate at present. Thus, to achieve this it will be paramount to ensure the user has the ability to make the final corrections required to obtain the resulting timeline they desire, and that can then be used in the contexts of education and investigation we identified earlier.

Thus, in creating the final application we target the following objectives:

- Provide the means to obtain the most accurate initial timeline possible.
- Provide the means to edit and correct the resulting timeline.
- Maximise the ease-of-use of the application.
- Enable investigation and exploration of the results.

6.2 Chosen Approach

Having considered a number of different approaches to building the final application, we opted to take a web-based approach. While similar results could be achieved both through a desktop application or a web-based application, the latter enables us to reach the widest possible audience: we reduce the user requirements to simply requiring that they have a compatible browser.

Hosting the application as a web service also shifts the resource needs from the *client* to the *server*, which are particularly significant in this case as the NLP tools we use require a large amount of memory to perform efficiently[1]. Requiring the user to provide these resources would significantly reduce our market size, with the application becoming inaccessible to a large proportion of users.

[1] Specifically, we require 6GB of RAM or more to execute the clustering process.

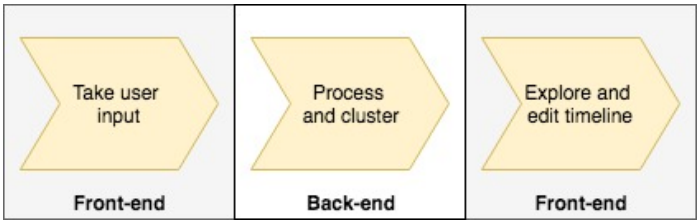


Figure 6.1: High level overview of the application pipeline.

Additionally, a large number of frameworks and libraries exist for simplifying web development, ensuring the typical design paradigms are followed and also helping to provide an idiomatic and familiar user-interface for users. A nice property that directly targets our objective of maximising the ease-of-use of the application.

As a result, we split our application in two: the back-end and the front-end, and consider each component separately in the following sections. Figure 6.1 illustrates the high level application flow.

6.3 Back-End

6.3.1 Responsibilities

The back-end is responsible for the bulk of the NLP processing to produce an annotated set of events via the clustering process described in Chapter 4, as a result providing the data for the front-end to then visualise as a timeline.

6.3.2 Technology Stack

The back-end is implemented in Java 8, using the tools and technologies outlined earlier in Section 3.4, following our initial experimentation.

6.3.3 Pipeline

Figure 6.2 shows the high level processing pipeline that takes place in the back-end, employing a number of the technologies we initially discussed in chapter 2 during our experimentation. A brief description as to the purpose of each part of this pipeline is provided below, numbered to reflect the sequential order of the pipeline shown:

1. Input text and optional actor list The initial input consists of the text to process, along with an optional list of actors to explicitly look out for in the text. This argument is provided to enhance the results produced by the system in two ways: it allows us to explicitly pick out any named mentions of these actors that may be missed by the coreference resolution system, and secondly it informs the following pre-processing stage in order to make some small modifications discussed next.

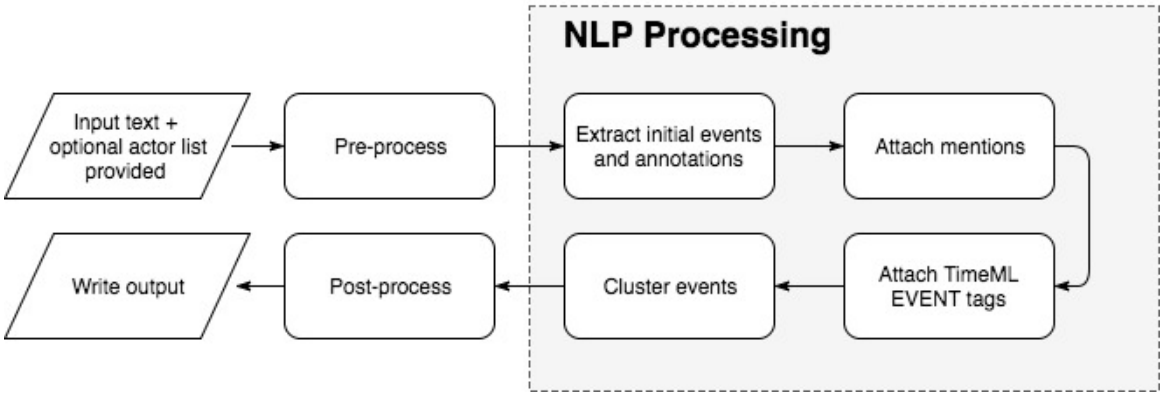


Figure 6.2: The high-level processing pipeline implemented in the back-end.

2. Pre-processing As we saw during our experimentation with coreference resolution tools, one of the major difficulties of the tool was in handling nominal mentions (e.g. the wolf) and fictional names (e.g. Goldilocks). Thus, to overcome this we have introduced a pre-processing phase to replace any explicit mentions of the actors listed in the provided actor list with a more common name of the same gender, improving the performance of the coreference resolution system without the need for us to develop our own training data to improve the coreference resolution system's performance on these types of mention.

3. Initial event extraction Here we identify our initial set of events, treating each sentence as an event. In addition, we also tag each event with the part-of-speech annotations used for the clustering process to follow. A subtlety here is our treatment of speech that occurs within the text. In some cases, we may have multiple sentences contained within a set of quote marks, which our initial sentence extractor splits into multiple distinct sentences. However, we do not wish to split a single speech across multiple events, so we instead piece back together any sentences that are contained within a single quote. To clarify, consider this simple example from Little Red Riding Hood:

> *"Don't dawdle along the way and please don't talk to strangers! The woods are dangerous."*

In such a case, we regard this as a single sentence itself, despite most parsing tools breaking this into two.

4. Attach mentions In this step we perform coreference resolution and tag each of the initial events identified previously with the characters they mention.

5. Attach TimeML EVENT tags We now identify any TimeML EVENT tags identified in each of the initial events and annotate accordingly. Thus, both stage 4 and 5 provide vital information to inform the clustering process to follow.

6. Clustering In this step, we execute our hierarchical clustering process as described in Chapter 4.

7. Post-processing We undo any of the changes made during the pre-processing stage, replacing any changed names with their original names in the input text.

8. Write output We finally output the results in the format expected by the front-end. As part of this process, we identify the optimal cluster sets at each of the thresholds identified in Section 4.7. The exact output format is described in Section 6.3.5.

Further details of each part of this pipeline are discussed in Section 6.3.5 to follow.

6.3.4 System Architecture

The back-end system consists of a large number of components and has been designed with flexibility in mind, employing object composition and inheritance to decouple the various components from one another and make it easy for us to switch between different implementations of particular components during the development process. To control access and develop some structure in the back-end, we define a number of distinct packages to encapsulate each component:

- **default** *Contains the top-level classes responsible for coordinating each of the components involved in the pipeline.*
- **clustering** *Contains the clustering class structure illustrated earlier in Section 4.5.*
- **coreference** *Contains all the components relevant to performing coreference resolution.*
- **eventTagging** *Contains the objects related to tagging the events with any TimeML EVENT tags present within each event.*
- **models** *Contains our abstract "event" data model and any related classes.*
- **utils** *Contains any shared utility classes providing useful functions that are used in multiple places.*

We thus illustrate the various parts of the system architecture individually for clarity.

Figure 6.3 shows the top-level system architecture. The `App` class is the application entry point, which initiates the application pipeline shown earlier. The `InputProcessor` is responsible for pre- and post-processing the input text, and the `JSONFormatter` is responsible for writing the final output to file, the exact format of which we discuss shortly. The `App` also coordinates the two most significant stages of the process: extracting the initial set of sentences from the text along with their associated *feature* annotations for the clustering process, and then executing the clustering process by passing the annotated `Event`s returned from the `EventExtractor` to the `EventClusterer`. We also see that each class maintains its own `logger`; this is the log4j logger that we use for logging any potentially useful debug information. All loggers write to the same output file, specified by a log4j configuration file, and prepend each line of output with the name of the class responsible for the output.

Figure 6.4 shows the event extraction class architecture in more detail. Here, we see the `EventExtractor` now becomes the coordinator, interacting with the `CoreferenceResolver` to perform coreference resolution over the initial input text, and the `EventTagger` to obtain the TimeML EVENT tags present in the input text. We have made use of both object composition and inheritance here in implementing the coreference resolution systems again to maximise flexibility. While experimentation revealed the Stanford *neural* coreference resolution system to outperform the alternatives, we have implemented all three of the Stanford approaches behind a common interface to make it easy to switch between any of these implementations. Both the *neural* and *statistical* implementations are part of the same Stanford CoreNLP package, with the desired approach to use being specified by a simple configuration property during initialisation of the Stanford Coreference Resolution tool. As such, these classes have been implemented via composition to simply initialise a `StanfordConfigurableResolver` with the appropriate properties. The *rule-based* implementation is instead part of a slightly different Stanford CoreNLP library, and thus requires a somewhat different initialisation process. The `EventClass` enumeration represents the 7 possible TimeML EVENT classes discussed earlier, and is used by the clustering process to assign a weighting to each of the possible EVENT classes.

Finally, Figure 6.5 shows the structure of the remaining packages defining the our abstract representation of each event as an `Event` object, and the utils package which currently contains only one utility class: `StringUtils`, which is used both during the initial event extraction and during mention annotation[2]. The details are discussed in Section 6.3.5 to follow.

We refer the reader back to Section 4.5 for the clustering package architecture.

[2]Tagging the actors mentioned in each event.

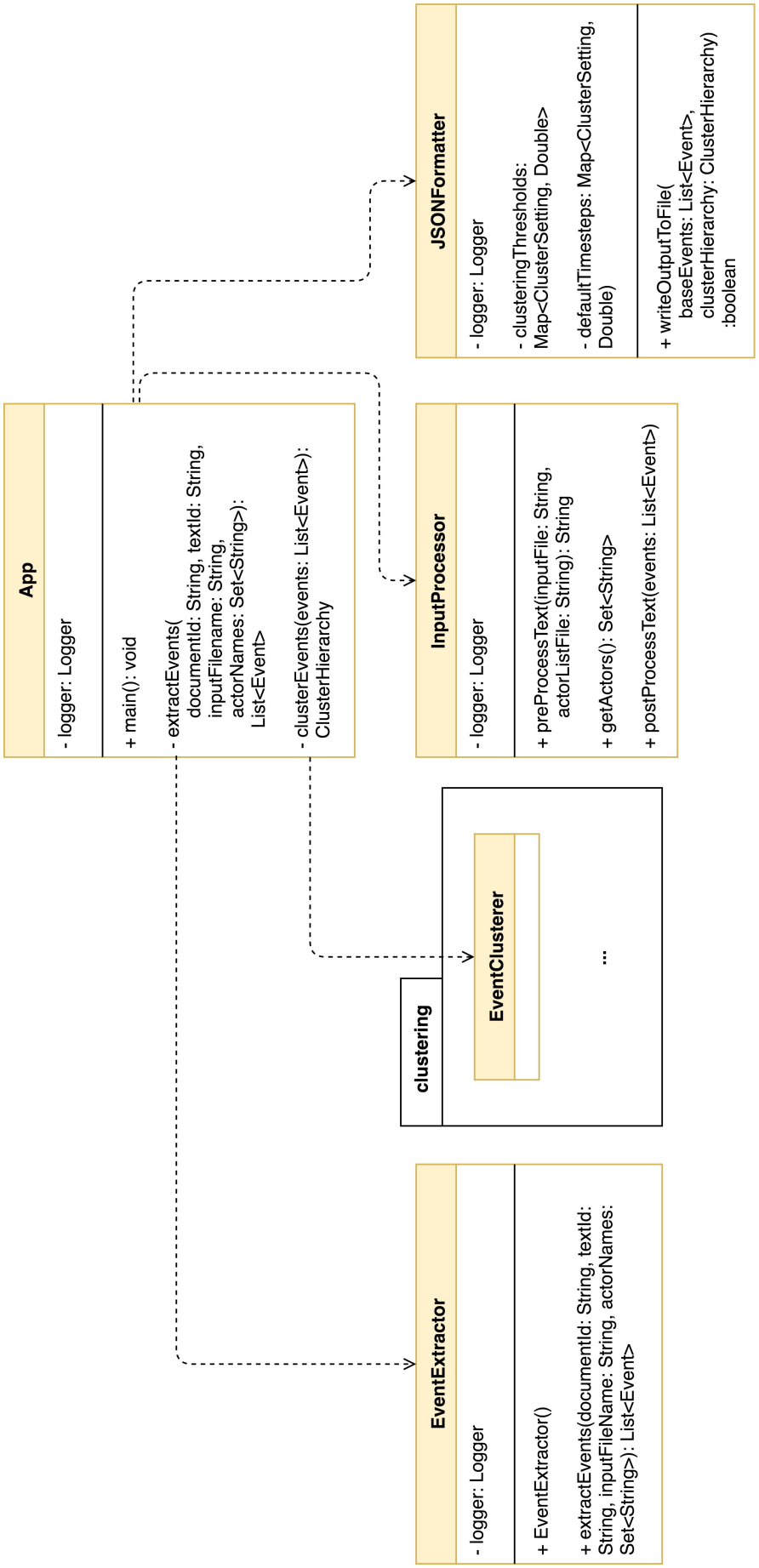


Figure 6.3: The top-level system architecture, where `App` is the application entry point.

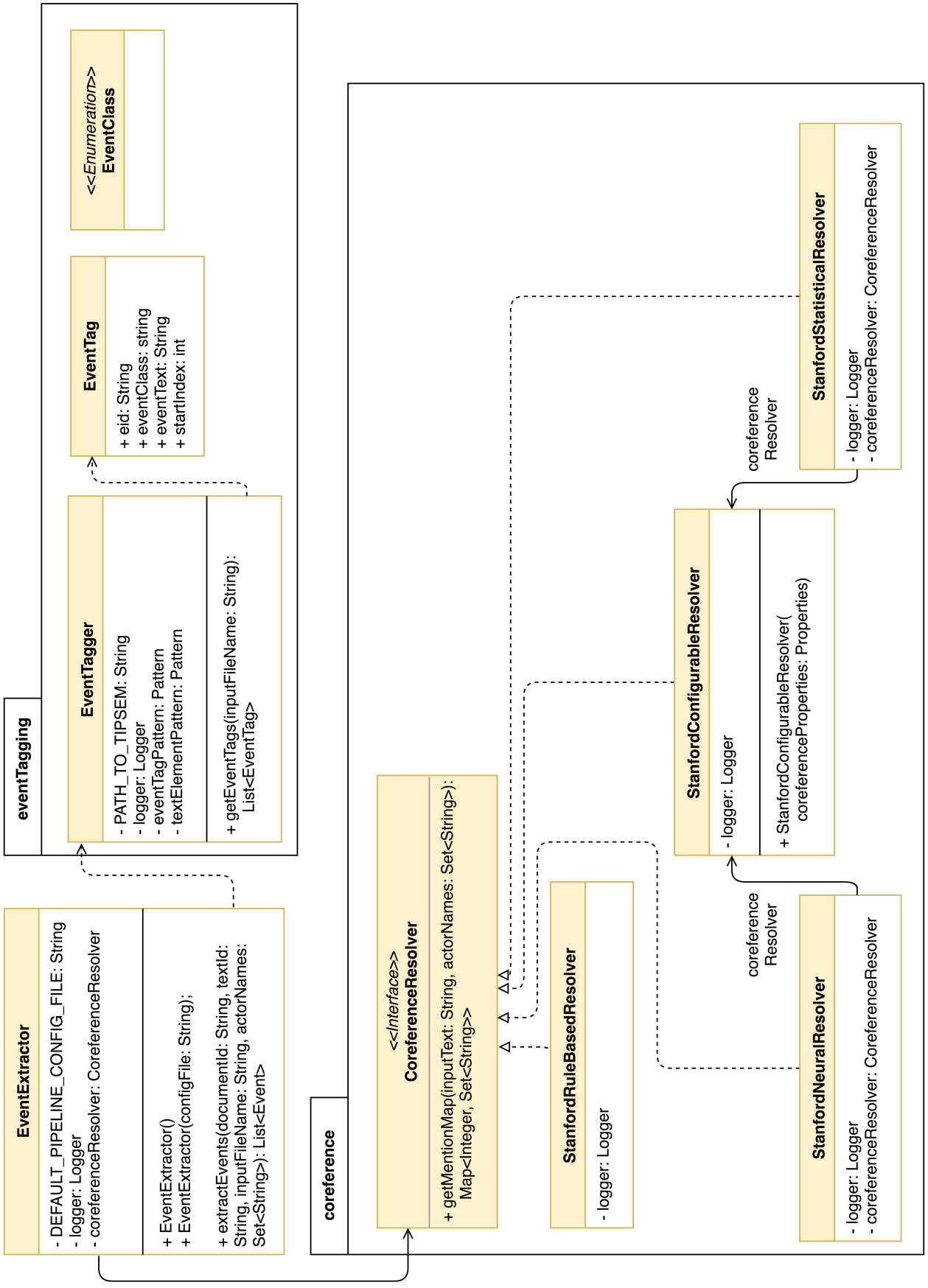


Figure 6.4: The event extraction component architecture, where the `EventExtractor` now acts as the top-level coordinator.

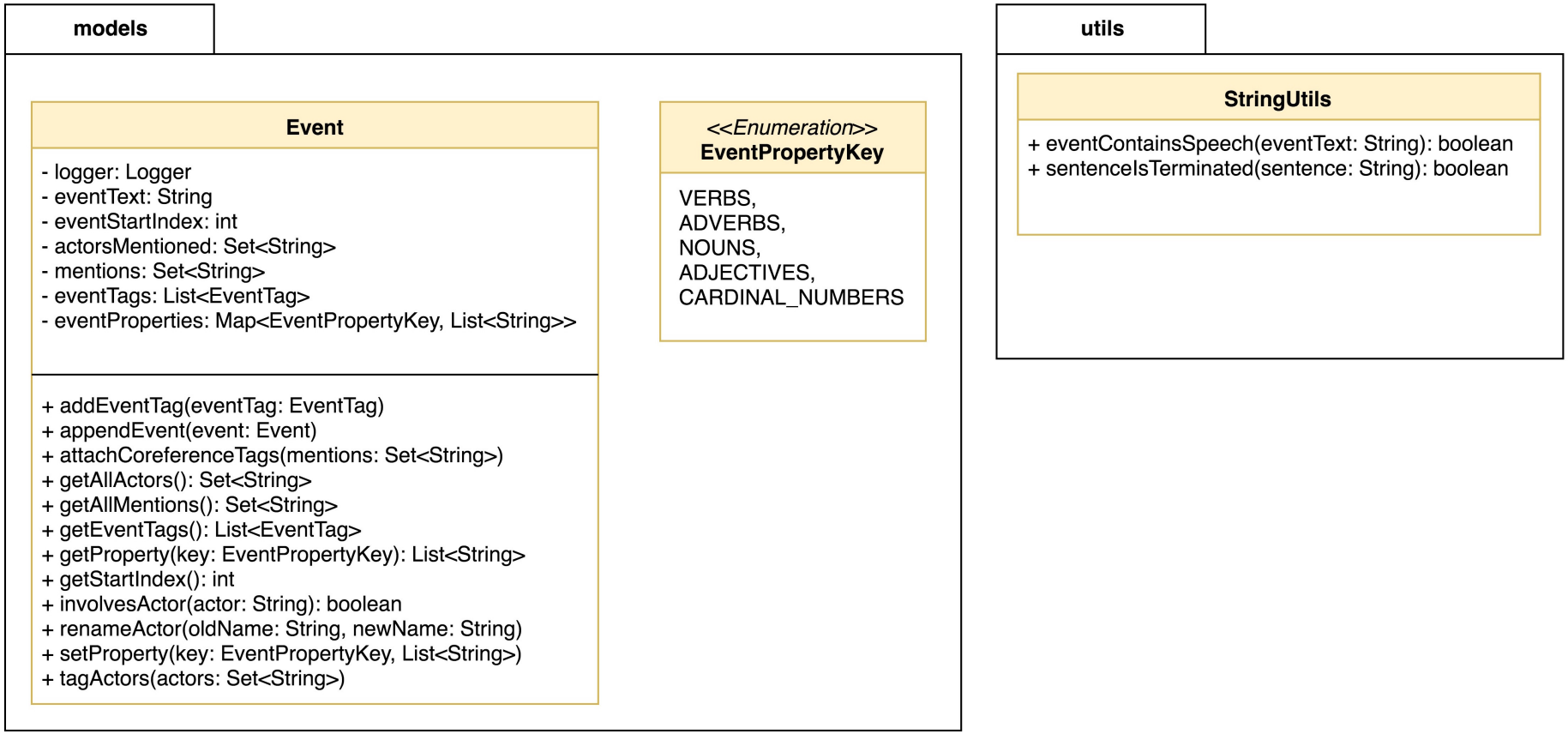


Figure 6.5: The additional components used throughout the application.

6.3.5 Implementation Details

Input Format

The application expects up to 2 input files: the input text to construct the timeline for, and an optional file listing any specific characters the user wants to ensure are recognised in the text. The story text is simply an unstructured .txt file, while the actor list file (if provided) is expected to be a JSON formatted file following the schema illustrated by the example below:

```
[
{ "name": "Goldilocks", "gender": "Female"},
{ "name": "Papa bear", "gender": "Male"},
{ "name": "Mama bear", "gender": "Female"},
{ "name": "Baby bear", "gender": "Female"}
]
```

Figure 6.6: The expected input format for the optional actor list. This is an example for the *Goldilocks* text.

i.e. each line specifies the name and gender of an actor expected to appear in the text, which is then used to inform our pre-processing stage to follow.

Pre-processing

To accommodate the pre-processing performed by the `InputProcessor`, we have defined two lists of common male and female names, stored in separate files of in the `resources/preprocessing` directory of our source. These lists are then used to provide replacement names for each of the actors specified in the user-provided actor list. This process proceeds in three steps:

1. We populate two lists with the replacement names defined in our two replacement name files. In doing this, we also make sure to remove from the list of candidate replacements any names that may already appear in the input text in order to avoid introducing any name clashes.

2. We then construct a mapping from each actor name in the user-specified actor list to its new replacement name, ignoring any names that have an unexpected or unspecified gender as well as any names that are considered to be "good"[3] already.

3. We finally output a new file with each of the actor names replaced with a more common name of the equivalent gender, and return this new file name to the caller. At this point, we are also careful to avoid swapping any phrases that should not be replaced. For example, phrases like "*Mr. and Mrs. Dursley*" should not be replaced with "*Mr. and Alice*" if the user specified *Mrs. Dursley* as an actor. To avoid this, we first process the text to replace any such mentions with a temporary placeholder, before replacing any names in the text. Once complete, we re-insert any of these temporarily removed phrases.

[3] They already agree with our idea of a common name.

Extracting the initial event set

The first three stages of our NLP Processing pipeline shown in Figure 6.2 are executed by the `extractEvents` method of the `EventExtractor`. The first step being the extraction of our initial event set: the set of *sentences* extracted from the text, initialising each as an `Event` object. At this point, we also annotate each `Event` with its associated part-of-speech tags. For both these tasks we employ the Illinois NLP pipeline, which we chose because of the potential to employ the Semantic Role Labelling capability of the pipeline should we choose to later introduce semantic roles as an additional clustering feature or for our visualisation.

The subtlety in this process is the difference in how we define a sentence in comparison to that used by the Illinois NLP pipeline. As we mentioned previously, we wish to ensure that any sentences that appear within a single quote stay together; this is perhaps a narrative-specific requirement, but we do not want speech in a narrative to be split across multiple events. We thus perform some sentence *repair* following the initial sentences extracted by the Illinois NLP pipeline, which ignores whether sentences occur within speech or not. When *stitching* these sentences back together, we must take care to treat the two possible scenarios differently:

1. If we've seen the opening speech mark, but have not yet seen the closing speech mark, we must continue to re-group each sentence we see until we find the last sentence in the speech.
2. On the other hand, if we have found the closing speech mark in a speech but the speech is not terminated, we must continue to re-group the subsequent sentences with this one until we reach a sentence that is terminated.

Let us illustrate the two cases above by example. In the first case, we may have the following two sentences identified by the Illinois NLP pipeline:

1. *"Don't dawdle along the way and please don't talk to strangers!*
2. *The woods are dangerous."*

In this case, we wish to group both of these sentences and treat them as one. The second case applies when we have sentences of the form:

1. *"Someone's been sleeping in my bed and she's still there!"*
2. *exclaimed Baby bear.*

Following this *repair*, we annotate each of these `Events` with the *nouns*, *adjectives*, and *cardinal number* phrases identified by the Illinois NLP pipeline's part-of-speech tagger.

Attaching mentions

Coreference annotation is achieved using the Stanford Coreference Resolution system discussed earlier. During this stage of the pipeline, executed by the `CoreferenceResolver` we build up a mapping from each sentence number (having completed any necessary *sentence stitching*) to the entities mentioned in that sentence. To achieve this, the coreference resolution system first executes over the *entire* original input text and returns a set of *coreference chains*, where each chain contains all the mentions of a particular entity identified in the text.

Each mention annotation holds the sentence number of that mention in the original text, however similarly to the Illinois NLP pipeline we encounter the same sentence identification problem, so we must first convert the sentence numbers of the mentions identified into those according to *our* definition of a sentence using a similar process to the above. Additionally, when populating the resulting map from sentence number to actor mentioned, we make sure to add the most representative mention of the entity represented by the coreference chain. In the case that one of the mentions in the chain matches the name of one of the actors the user has explicitly identified in the *actor list* input file then we add this exact name to the map, while if this is not the case, we take what Stanford's Coreference Resolution system itself identifies as the *representative mention* of the coreference chain: typically the longest mention in the chain.

Having constructed this mapping, we then tag each of the initially extracted `Event` objects from the previous stage with the entities mentioned in that event. During this tagging phase, we tag any actors that were explicitly identified by the user separately to any other mentions to allow for different weightings to each of these features during the clustering process, however we do not make a big distinction between these two types of coreference (whether it is an *actor* or another type of coreference) yet as we can only tell the difference when the user informs us who to regard as an actor as part of the application input: we do not yet have any automated identification as to whether a mention is a person or another object.

Attaching TimeML EVENT tags

The `EventTagger` is responsible for the last phase of feature extraction: executing TIPSem to identify all the *events* that take place in the text according to the TimeML EVENT tag definition. This process requires first executing TIPSem over the original input file to obtain an annotated .tml file (as we saw in Section 3.3.3), before parsing this file to extract any *events* identified in the text. Event tags in the resulting output file are identified by employing the following regular expression:

```
"<EVENT class=\"(?<class>\\w+)\" eid=\"(?<eid>e\\d+)\">(?<eventText>\\w+)</EVENT>"
```

Figure 6.7: The regular expression used to extract each of the *event triggers* recognised by TIPSem.

This expression makes use of *named groups* to allow us to easily extract the three key details captured in each of the TIPSem EVENT annotations: the *event class*, the *event id*, and the actual *event trigger* itself. As a result of executing the `getEventTags` method of the `EventTagger`, the `EventExtractor` is then able to annotate each of our `Event` objects from earlier with the TimeML *events* they contain.

Each `EventTag` returned by the `EventTagger` contains the three key details highlighted above as well as the index of the *event trigger* in the original text. This allows us to then identify the `Event` object whose text contains the current `EventTag`, as each `Event` also holds the index of the first character in its text.

An alternative approach here could have been to re-execute TIPSem over each `Event` object separately, and to immediately tag each event with the TimeML *EVENT* tags identified in that event. However, such an approach would have required repeated executions of the TIPSem process, and the creation of multiple short input files. We instead adopted this approach due to the improved time efficiency despite requiring slightly more complex tagging logic.

Post-processing

At this point, all analysis has been complete and thus we reset any changed names back to their original form. This replacement is performed by the `Event` object's `renameActor` method, which updates the events `eventText`, `actorsMentioned`, and other `mentions` properties.

Output

The final stage of the back-end pipeline is to output the results that feed the front-end, for which the `JSONFormatter` is responsible. The `JSONFormatter` takes the list of the initial events extracted[4] and the final `ClusterHierarchy` produced by the clustering process as input and uses this to construct the resulting `out.json` file of the form shown in Figure 6.8.

This output consists of four components: the *baseEvents*, the *clusters*, the *labels*, and the *defaultClusters*. This representation was chosen as it maximises readability while minimising the amount of data transmitted to the user when they receive their results as the event data is only represented once. The *baseEvents* specify each of the sentence-level events initially extracted. The *clusters* section lists the cluster sets at each time step of the hierarchical clustering process. i.e. in the example of Figure 6.8, we see that the first cluster set consists of each event being contained within its own distinct cluster. By the final time step, we see all three events have been merged into a single cluster.

[4]Each initial event is essentially a *sentence* according to what we define to be a *sentence*, shown earlier.

```
{
  "baseEvents": [
    {"text": "Once upon a time, there was a little girl named Goldilocks.", "actors": ["Goldilocks"]},
    {"text": "She went for a walk in the forest.", "actors": ["Goldilocks"]},
    {"text": "Pretty soon, she came upon a house.", "actors": ["Goldilocks"]}
  ],

  "clusters": [
    [[0], [1], [2]],
    [[0, 1], [2]],
    [[0, 1, 2]]
    ],

  "labels": [
    {"type": "label", "title": "Goldilocks"},
    {"type": "label", "title": "Papa bear"}],

  "defaultClusters": {
    "extra-fine": 0,
    "fine": 0,
    "coarse": 1,
    "extra-coarse": 2}
}
```

Figure 6.8: Example output file from the `JSONFormatter` according to the expected schema.

The labels list the names of the actors we wish to show in the resulting timeline, and while they allow for the separation of the definition of these actor labels from the actors actually tagged in each of the events, in practice we actually initially have a label for each entity mentioned in any event in the text. Finally, the *defaultClusters* section lists the indexes in the *clusters* property of the event clusters at each of our automatically identified "optimal" cluster sets at the 4 levels of granularity identified previously in Section 4.7:

1. **Extra-fine** *20% max score threshold*
2. **Fine** *10% max score threshold*
3. **Coarse** *5% max score threshold*
4. **Extra-coarse** *2.5% max score threshold*

These time steps are identified and populated by the `JSONFormatter` using the information provided by the `ClusterHierarchy` parameter. In the case that a particular score threshold identified previously is never reached, we instead employ a set of default indices based upon the total number of steps in the hierarchical cluster. At present, we split the hierarchy into 6 equally spaced segments, with our default time steps defined as follows:

1. **Extra-fine** *17% through total number of steps*
2. **Fine** *34% through total number of steps*
3. **Coarse** *68% through total number of steps*
4. **Extra-coarse** *85% through total number of steps*

6.4 Front-End

6.4.1 Responsibilities

The front-end aims to provide a clear and intuitive user interface to allow the user to understand, edit and interact with the results produced by the back-end application. The front-end should make it as easy as possible to tweak the resulting output to obtain a final timeline of real value to the user.

In particular, at present we have no means to distinguish between coreferences of *characters* within the text and repeated mentions of other types of entity. As a result, we currently tag events with both these types of mention, which should not necessarily appear in the resulting timeline. To combat this, we must provide the means to easily remove any irrelevant coreference annotations. Additionally, in some cases we see that character mentions are missed entirely by the automated coreference resolution system. As such, we must also provide the means to easily add these missing tags to then update the resulting timeline appropriately.

6.4.2 Technology Stack

The front-end is built using the typical trio of web technologies: HTML, CSS, and JavaScript. The exact frameworks and libraries we employ are listed in Table 6.1 along with a brief description of their use.

In the following you'll find our extensive use of open-source libraries, which has allowed us to rapidly accelerate the development process and benefit from the large amount of development that has already gone into each of these libraries and frameworks. For example, the use of the QUnit gives us far more power in our unit testing, and greatly simplifies the number of concerns we must handle ourselves when unit testing. As an example, QUnit resets the DOM of the test page between every unit test to ensure that each test starts from a clean slate and is thus not affected by the behaviour of our other tests.

Dependency	Purpose
Node.js	A server-side JavaScript runtime to allow us to program our server using JavaScript. Using this also provides access to the Node Package Manager (npm) containing "the largest ecosystem of open source libraries in the world" [65]. This speeds up the development process and allows us to focus on the application development rather than the additional concerns of running a web server.
express	Web framework for Node.js, used to serve the site content.
ejs	Node.js library that allows us to inject JavaScript into our HTML templates. This makes the construction of dynamic web pages far easier.
body-parser	Node.js library used to parse the body of POST requests to the server, and provide direct access to an JavaScript object representation of the body instead.
spinkit	Node.js library used to display a loading animation while the user is waiting for the back-end process to complete.
sortablejs	Node.js library used to create interactive re-orderable lists. We use this to create our sortable list of character labels in the timeline settings window shown in Figure 6.14.
jQuery	JavaScript library used to simplify interaction with the HTML Document Object Model (DOM).
Bootstrap	CSS library used to provide a familiar user-interface and to speed up the development process.
D3.js	The library used to draw the previously described force-directed graph.
Tipsy	The library used to display tooltips over timeline events.
Font Awesome	Used to obtain idiomatic icons that reflect the purpose of various components of the UI.
Google Fonts	Used for the very popular and clear Roboto font.
QUnit	JavaScript unit testing framework, chosen for its graphical test result interface, and structured output on test failure including the potential to produce a native JavaScript stack trace.

Table 6.1: Table of dependencies for the front-end and a brief description of how each is used.

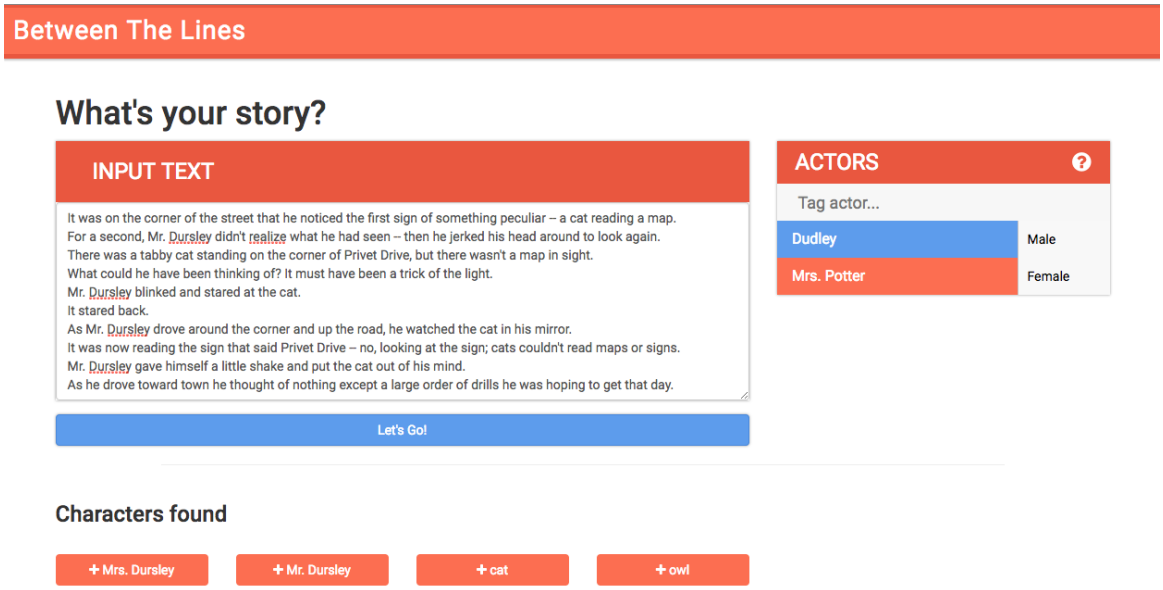


Figure 6.9: The *input page* of the application, showing both the actor list containing two actors, and the additional characters automatically recognised in the input text.

6.4.3 Results and Features

The ultimate idea has been to create an idiomatic and familiar interface to minimise the learning curve with the application. In doing so, we hope to present all the necessary information and provide all the functionality required to construct the final result, without cluttering the interface.

The resulting application consists of two pages: *the input page* and *the insights page*.

The Input Page

Figure 6.9 shows the input page: a relatively minimalistic page where the user provides their initial input text. In addition, this page contains the following two key features:

1) Actor List Alongside the input text box, we also provide the means for the user to explicitly name any actors they would like to ensure are identified by the system and are plotted in the resulting timeline. The elements of this list are then passed as input to the back-end and used to inform the pre-processing phase described earlier. Using this proves particularly useful when the characters mentioned are of the form *Mr. Dursley*, *the wolf*, or are particularly unusual names, as explicitly specifying these enable us to rename such characters during the pre-processing phase to improve the results of coreference resolution and thus improve the accuracy of the initial timeline produced. Once a gender for an actor has been selected, we colour the item appropriately to aid visual clarity when scanning a long list of actors.

2) Automatic Identification of Characters In addition, the *Characters Found* area beneath the input text automatically displays any potential character names that should be added to the actor list described above. Clicking on any of these automatically identified actors adds them to the actor list. The idea of this feature is to reduce the need for the user to explicitly search their text for all the characters mentioned, and to particularly pick out any potentially troublesome mentions that we're able to easily spot, such as "*Mr. x*".

The Insights Page

The *insights page* is responsible for displaying the resulting timeline and enabling the user to make any edits and corrections to the result. This page exhibits a number of key features designed to maximise user insight and usability, which we describe and highlight in the following.

1) Actor Side Panel This refers to the panel on the right-hand side of Figure 6.10, displaying the names of all the entities mentioned in the text. As we can see, this panel currently consists of more than just the characters mentioned in the text, but also any other entities that are mentioned multiple times. This is due to our current inability to distinguish between the two at present. To overcome this, we have added the *quick delete* button on the far right of each mention when hovered over, allowing the user to quickly remove any mentions that are deemed irrelevant.

Additionally, this panel is currently sorted in descending order of mention frequency to highlight any significant characters, with the value in brackets representing the number of events that mention this actor. We also position any "similar" mentions next to one another in this list, a factor more evident in our later example of *the gingerbread man*, when the coreference resolution system identifies two disjoint coreference chains for a single entity. This makes it easier for the user to then recognise that an actor is referred to by two different names and subsequently amend this.

2) Actor Modal Clicking on any of these actors in the side panel brings up the *Edit Actor* screen shown in Figure 6.12: a modal screen that displays all the sentences that are currently tagged as having mentioned this actor. At this point, the user can make any changes to any of these individual sentences, or they can rename the actor in the input box at the top to perform a bulk renaming of this actor. Additionally, the user can also split an actor into multiple mentions. For example, in the case of *the three bears*, we may wish to instead replace any mentions of this entity with each of *Papa bear*, *Mama bear*, and *Baby bear*.

3) Adjustable Event Granularity Scrolling down the page, we find the current set of events illustrated in the timeline at the top of the page, and the *Event Detail* panel, as shown in Figure 6.10. This panel consists of buttons to jump to the automatically identified event sets that should present a relatively good set of events at the granularities discussed before, and we additionally provide a detail slider to allow the user to find the exact set of events that suit them if the automatic milestones aren't quite right.

4) Event Modal Clicking on any event in the event table displays the *Edit Event* screen, enabling the user to make any manual changes to the event text, or to add, edit, or remove any of the actors tagged in a particular sentence within the event. This is illustrated in Figure 6.13.

5) Manually Merge Events If the automatic event clustering algorithm fails to produce a satisfactory set of events, the user can manually create their own events starting from the base set of events at any time step. Entering *Merge Mode* using the button above the event table allows the user to select any contiguous events in the event table that they wish to merge. Subsequently clicking the *Merge Events* button shall then commit any selected event merges. The example of Figure 6.11 shows an example where we've entered *Merge Mode* and selected the top two events. Having completed a merge, the *undo* button is enabled for the current event set, allowing the user to revert any mistaken merges.

6) Re-order Plot Labels In many cases, the exact ordering of the actor labels on the left-hand side of the plot will greatly affect the clarity of the timeline obtained. As a result, clicking on the *Settings* button above the timeline shown in Figure 6.10 opens the *Timeline Settings* window, allowing the user to specify the desired label ordering and subsequently click the *Update Timeline* button beneath the timeline to realise this change, visible in Figure 6.14.

7) Automatically Ordering Labels Despite it not always being immediately obvious how to order the actor labels in the resulting timeline for the best layout, we currently employ the relatively effective heuristic of initially ordering actor labels based upon first-appearance text order. This typically tends to group characters that interact with one another well by positioning actors with any other actors mentioned nearby, however the performance of this approach does degrade with larger texts.

8) Plot Re-scaling We additionally provide the means to zoom in and out of the timeline independently of the rest of the UI, using the two zoom buttons positioned above the timeline plot.

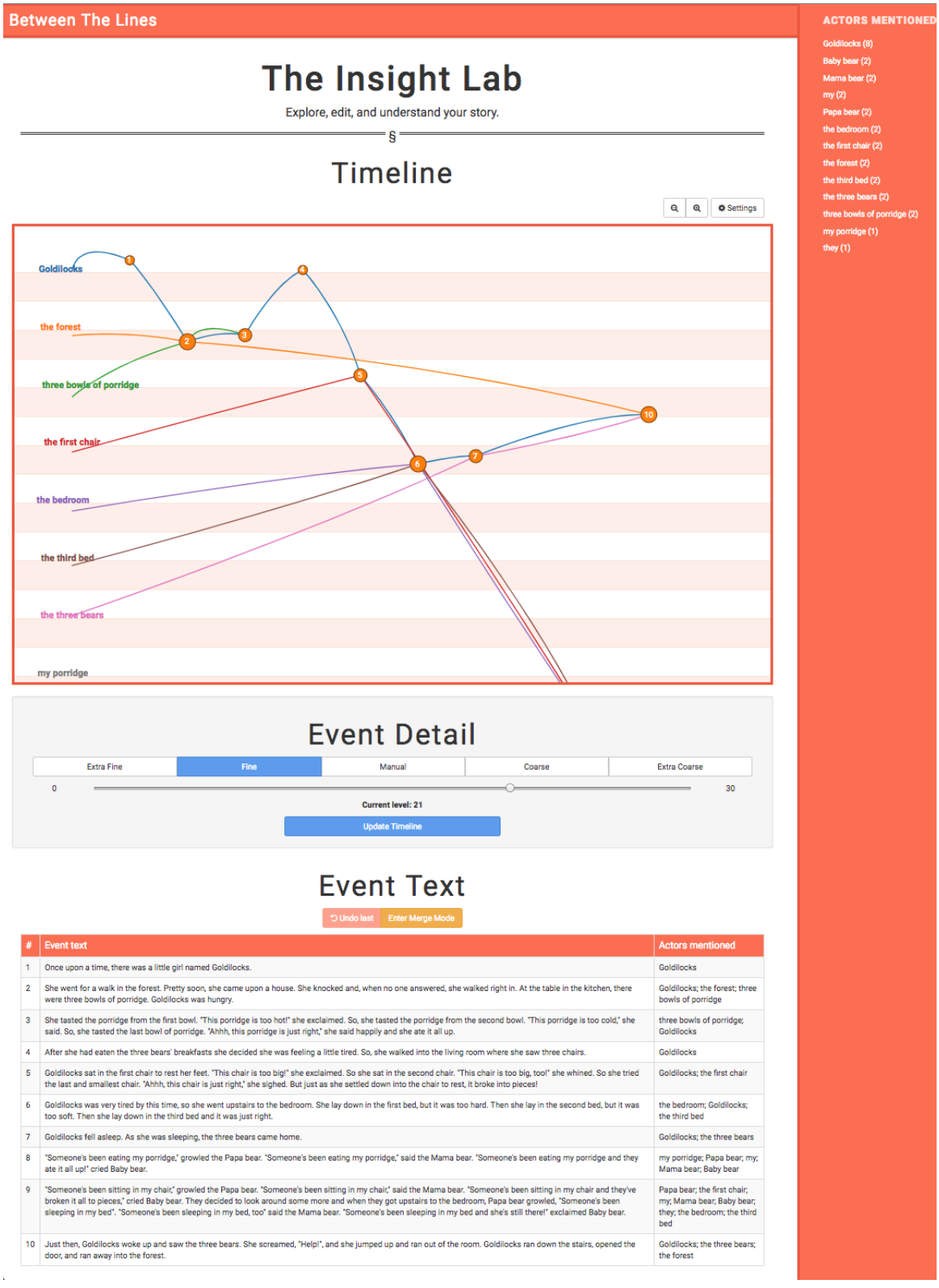


Figure 6.10: Full page overview showing the *timeline view*, *event detail* selection panel which hosts short-cut buttons to our four default levels of event granularity along with a manual slider to more precise granularity control, the *event text* table, and the *actor side panel* on the right-hand side of the screen displaying all the actors mentioned.

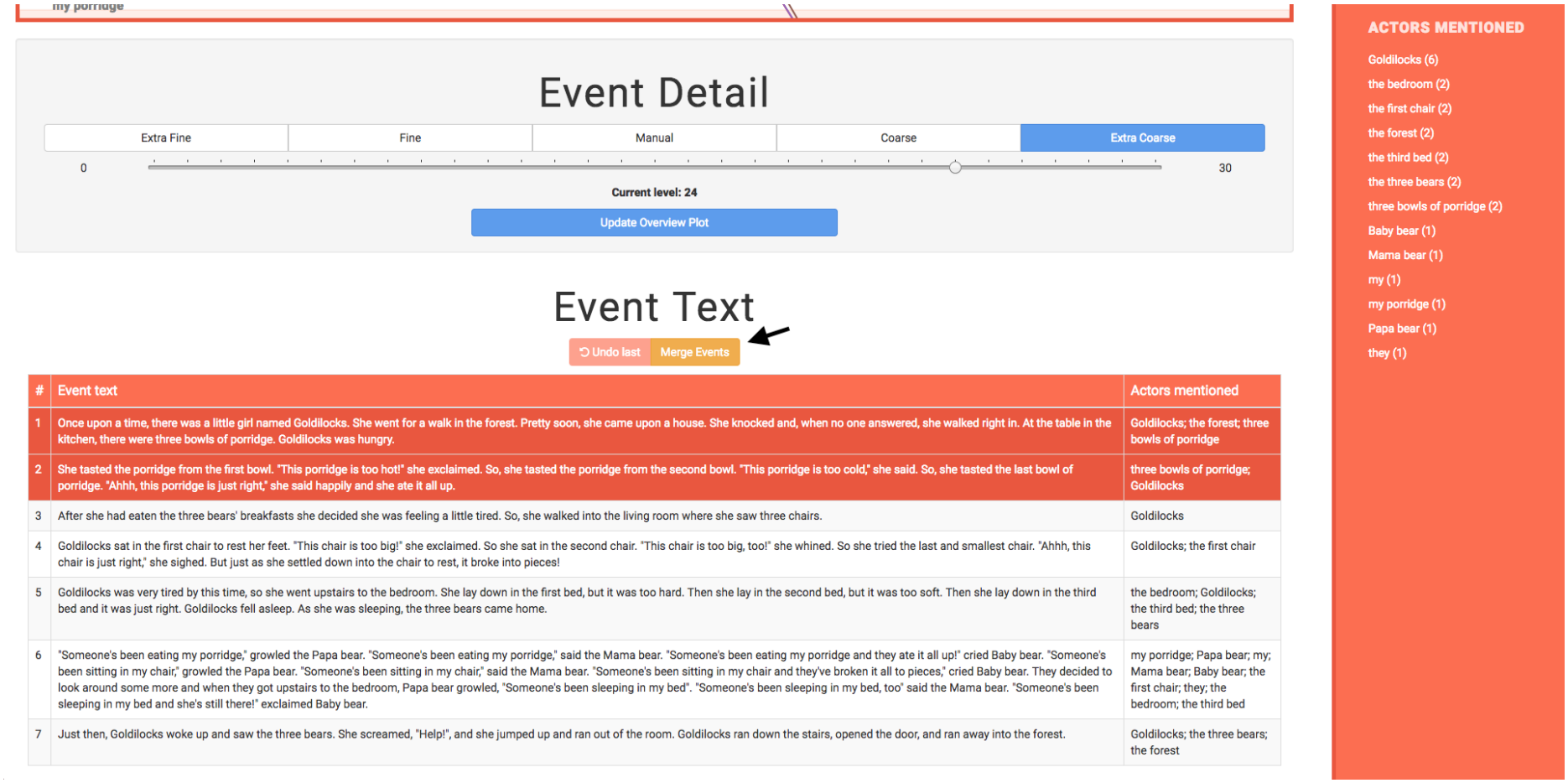


Figure 6.11: Specifying the events to *merge* having entered *merge mode*. The arrow highlights the changed state of the merge button, now containing the text: "Merge Events" in comparison to the previous screen. The two contiguous events highlighted in a deep red are the events that the user has chosen to merge by simply clicking on them after entering *merge mode*.

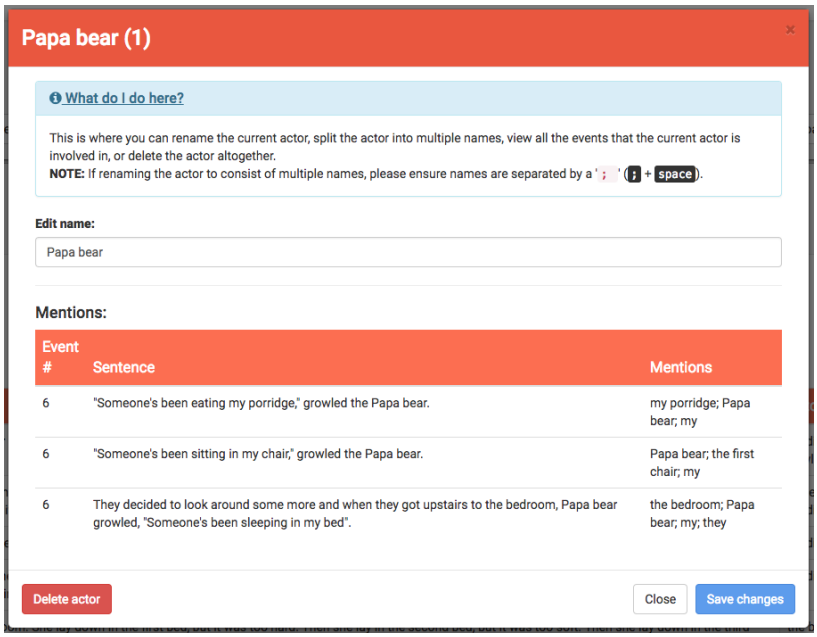


Figure 6.12: The *Edit Actor* modal. The "edit name" input at the top allows the user to rename an actor to one or multiple new names. Beneath this, we see all the sentences that this character is currently mentioned in, along with the event number of these sentences. Additionally, at the top of the panel we see the current character's name and, in brackets, the number of events that mention the current actor at the current level of granularity.

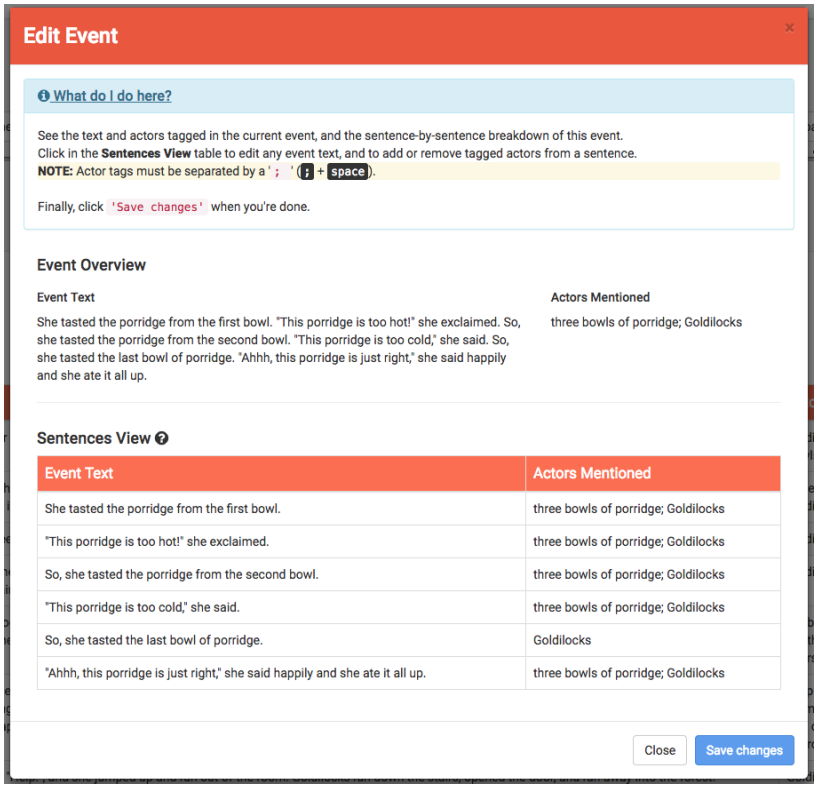


Figure 6.13: The *Edit Event* modal, allowing the edit of an event's details. In this window we see each of the individual sentences that comprise the current event, where the user can then edit any of the text or mentions of each of these individual sentences.

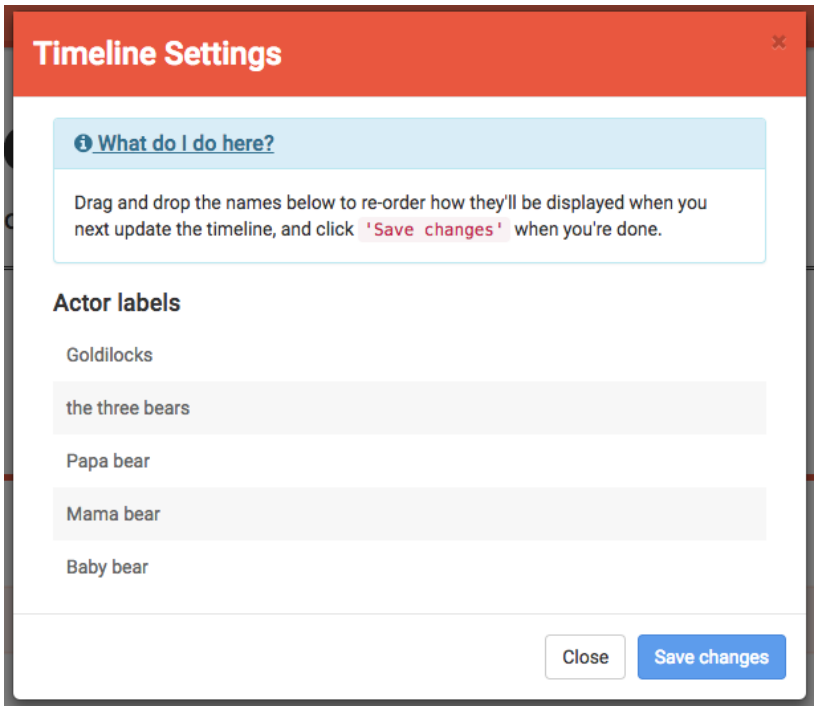


Figure 6.14: The *Timeline Settings* modal. This panel allows the user to drag and drop character names to re-order them. The ordering expressed here is that reflected by the resulting timeline.

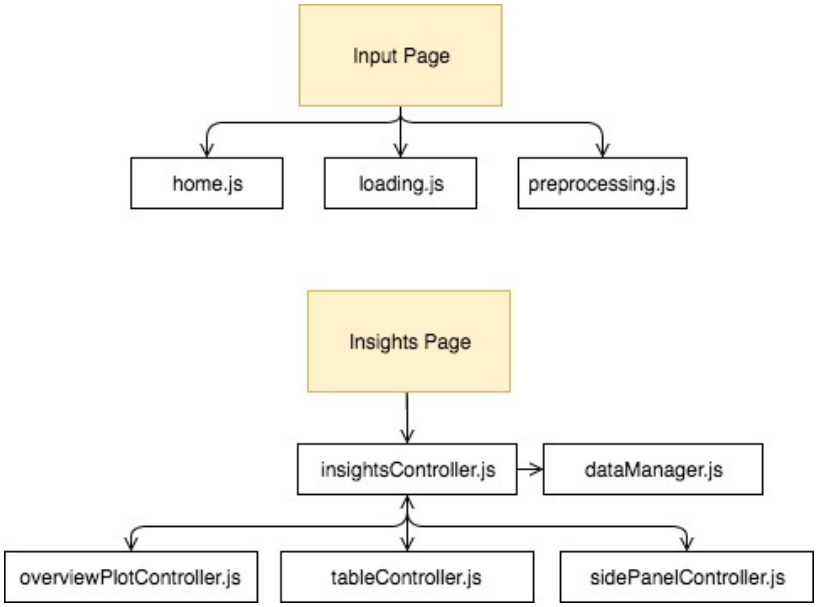


Figure 6.15: High level script architecture of each page of the front-end application.

6.4.4 System Architecture

The front-end has been designed largely following the Model-View-Controller paradigm, yielding a nice separation of concerns between the different components in the system. Figure 6.15 illustrates the high level script structure of the two pages, with arrows reflecting the communication channels between components. Once the *insights page* is displayed, all further interaction and processing is performed on the client-side, possible due to the relatively simple nature of the computation that takes place in the front-end following the processing done by the back-end. This means that the application actually requires relatively little bandwidth, as once the JSON data has been retrieved from the back-end no further server requests are required to perform any of the actions possible on the *insights page*.

The Input Page

The *input page* is a relatively simple page responsible for taking the user input and directing that to the back-end for processing. As a result, we have a fairly simple architecture for this page consisting of three scripts:

- **home.js** *Script responsible for coordinating all the actions on the input page, such as recognising user clicks, adding items to the actor list, and sending the data to the server for processing.*
- **loading.js** *Script defining the loadingPanelController responsible for displaying a loading window while the server processes the input data to ensure the user is aware that the system is still processing, as this can take a while on larger texts. This is triggered by the homeController in home.js.*
- **preprocessing.js** *This script registers a listener to the input text area, triggering the automatic character recognition process whenever the user input changes.*

The Insights Page

The *Insights Page* is somewhat more complex, providing a far greater amount of functionality. Figure 6.15 reflects the hierarchical architecture adopted for this page. Each script encompasses an object of the same name, exposing only the public methods we wish to expose to the outer environment. The `insightsController` acts as a top-level coordination object, and has been enforced as the only route of communication between the three other controllers, each responsible for their own particular portion of the page. The `dataManager` encapsulates our data model, providing access to the event data. This design was developed to nicely encapsulate the behaviour of each component of the page within its own object boundaries, avoiding direct interaction between the different components on the page. This makes it relatively easy to introduce additional components to the page, or replace existing ones so long as the new objects adhere to the same API that is currently exposed.

Under this architecture, it is predominantly the `insightsController` that interacts with the `dataManager`. This ensures that only the `insightsController` need know about *all* the other components on the page, and can keep them in sync with one another whenever there is a change. An alternative considered was to allow direct communication between each of the controllers and the `dataManager` itself, and adopt the *observer* pattern to allow each controller to register themselves as an observer on the `dataManager`. As a result, whenever an update was made to any underlying data, the `dataManager` could notify any observers of this change. However, the downside of such an approach would be that whenever a change is made, all observers would be immediately notified unless we adopted a more sophisticated approach. For example, renaming an actor to one or more new names may lead to a large number of notifications triggering each component to repeatedly re-draw itself. Under our current approach we can instead first perform all data updates, and finally inform the top-level `insightsController` that a change has been made that may require the other views to update. The `insightsController` then triggers the redrawing process.

6.4.5 Implementation Details

In this section we highlight a few of the key implementation details of the features outlined above.

Automatic Character Recognition

At present, we search for characters of the following form:

- Titled names *e.g. Mrs. Dursley, Dr. No*
- Animals
- Common names *e.g. James, Nicole*

Pre-processing names like these typically makes a fairly significant impact on the performance of the application by improving the results of coreference resolution. This process is triggered by the *oninput* HTML event whenever the user changes their current input. This was chosen over listening

for the *onchange* HTML event as the latter requires the user to take the input text area out of focus (i.e. by clicking somewhere else) before the processing would be triggered. However, in practice it is quite common for the user to input their text and hit the start button before allowing the *onchange* handler to fire. The *oninput* event instead fires as soon as there is a change of input.

Searching for animals and common names is performed using a dictionary approach, simply checking for any matches of our expected phrases. Searching for titled names, on the other hand, is performed through the use of the following regular expression:

```
/\b(M(r|rs|s)|Dr)\.  (\w|-)+/gi
```

This searches for *all* matches of this pattern, ignoring case. We subsequently filter the results to remove any phrases of the form: *Mr. and Mrs. x*, to ensure that such characters are not pre-processed. Joint entities such as this are not gendered, typically co-referred to as *they*, so we at present do not perform any replacement of such entities.

Side Panel Ordering

The actor ordering in the side panel is achieved by a three stage process:

1. Score and order mention pairs based upon word similarity.
2. Cluster mentions that are deemed to be similar enough to perhaps be the same entity.
3. Sort the mention clusters both internally and externally in descending order of mention frequency.

Stage 1 calculates a similarity score between every pair of mentions[5]. For our measure of similarity we take the simple measure of *word overlap*. That is, we split the first mention into its constituent words and search for any case-insensitive matches in the second mention. Additionally, at this point we also ignore any pluralisation or possessives. We then take as the resulting score the number of word matches relative to the number of words in the shortest mention. Thus, considering the two example mentions below:

> *a little gingerbread man*
> *the gingerbread man called over his shoulder, "Run, run, as fast as you can*

In this example, we see that there is an overlap of 2 out of the 4 words in the first mention yielding a score of 0.5. Additionally, we ignore any common phrases that provide little value in determining mention similarity such as *a, the, etc.*

The second stage then *clusters* the mentions based upon similarity, moving two phrases into the same cluster if they have a similarity score above 0.25, an empirically found threshold that appears to perform well in practice.

[5]Note, we're careful to only score every pair once as the order of the pair does not matter in this case.

Subsequently, we then order each cluster based upon the frequency of the mentions within the cluster, and lastly order clusters relative to one another based upon the maximum mention frequency of any mention within that cluster, yielding our resulting order for the side panel.

This strategy is a relatively quick and simple technique that yields effective results. Other methods such as employing word vectors and using cosine similarity were also considered, yet yielded similar or worse results, especially if the consideration of plurals and possessives are not encoded into the word vector features.

Manual Event Merging and Undo

The `dataManager` is responsible for providing access to the underlying event data and performing any data manipulation, having parsed the initial JSON output produced by the back-end system. As a result, the `dataManager` maintains three pieces of state:

1. **baseEvents** *The list of basic events at the granularity of individual sentences.*
2. **clusters** *The list of event clusters at each time step in the hierarchical clustering process, as shown in the example back-end output file.*
3. **labels** *The current actor labels to display in the resulting timeline.*

Merging any two events at a particular time step is achieved by merging the two sets of cluster indices in the *clusters* list of the `dataManager`. For example, if *event 1* consists of the cluster indices [1, 2, 3] and *event 2* is represented separately as [4, 5], then following a merge these two separate lists will instead by represented as the single list: [1, 2, 3, 4, 5], representing the indices of the *baseEvents* that are clustered into this single event.

To allow the reverting of any merges, we maintain an *undo stack*. Thus, before performing a merge we first *push* a copy of the current cluster indices to the *undo stack*. Following this, any subsequent undo operation on the events at the current time step will replace the current cluster indices with those that are next *popped* from the stack. The current implementation may be slightly inefficient in terms of memory utilisation as we store the entire set of cluster indices following every merge, however we opted for this simple approach to ensure *correctness* of the implementation before we may then look to optimise the application.

6.5 Resource Requirements

Overall, the back-end processing imposes the most significant demand on resources, requiring approximately 6GB of RAM to execute efficiently. This is split as follows:

- 2GB - Part-of-speech annotation and sentence extraction as part of the Illinois NLP pipeline [66].
- 4GB - Stanford coreference resolution [67].

However, increases in the size of input text can result in increases in coreference memory consumption of up to 5GB or 6GB. This is a result of the number of annotations required by the coreference resolution system prior to execution, first performing POS tagging, dependency parsing and named-entity recognition. Thus, there is certainly some potential for optimisation here by replacing the Illinois NLP Pipeline with the Stanford system for our initial POS tagging, if we're sure we're not going to introduce the Illinois SRL analysis. Alternatively, we could also explore translating the annotations produced by the Illinois system into those utilised by the Stanford system to avoid this duplication.

Allowing the Java runtime heap to cater to this demand requires the user to explicitly increase the maximum heap space allocated to the JVM. Thus, for safety we execute the application with a slightly generous 8GB of possible heap space to allow for even relatively large texts to be processed in one go. This is specified by passing the runtime option, `-Xmx8g`, to the JVM. Scaling the system to handle full length novels would certainly require some additional tailoring to process the text in smaller segments.

Thus, we see the rather extensive resource requirements of this application would make this relatively inaccessible to a large number of users, although the number of modern day computers with 8GB or more of RAM is growing. By taking a web-based approach, separating the client from these high resource requirements, we can reach a wider audience, although server hosting is likely to be far more expensive so further optimisation would need to be explored.

Chapter 7

Results and Evaluation

7.1 Case Studies

In the following sections we present the end-to-end usage of our application over three example texts to highlight the quality of the initial events produced, the ease-of-use of the application, and the potential insights to be drawn from the resulting timeline.

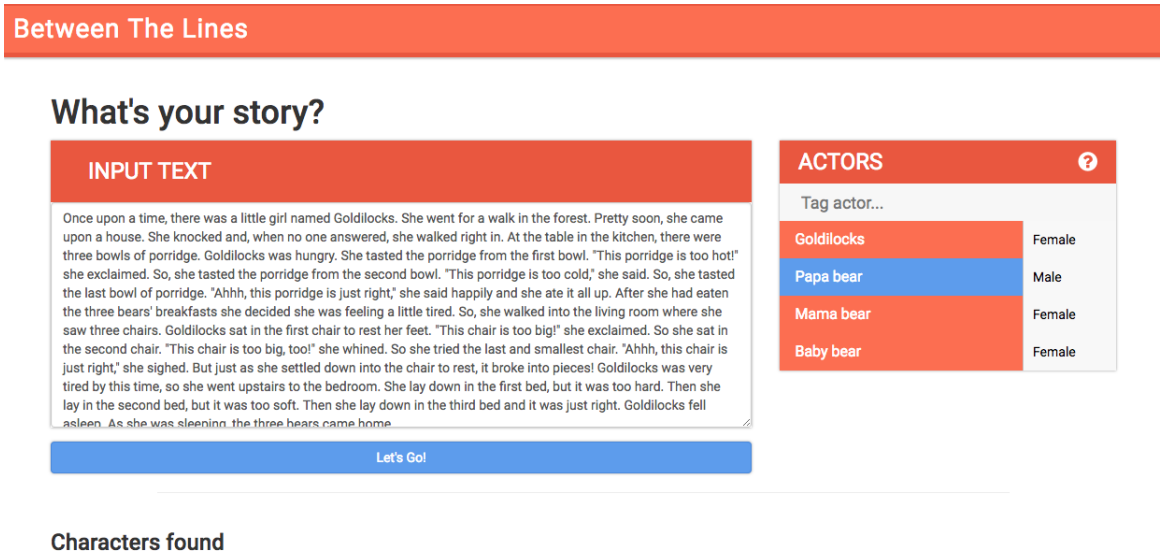
7.1.1 Goldilocks and the Three Bears

We begin in Figure 7.1a by entering our story into the text box and listing the four characters of interest to us here. This makes a particular difference in this example as Goldilocks is not a typical female name and without pre-processing each of the three bears, the coreference resolution system fails to make the distinction between the three of them.

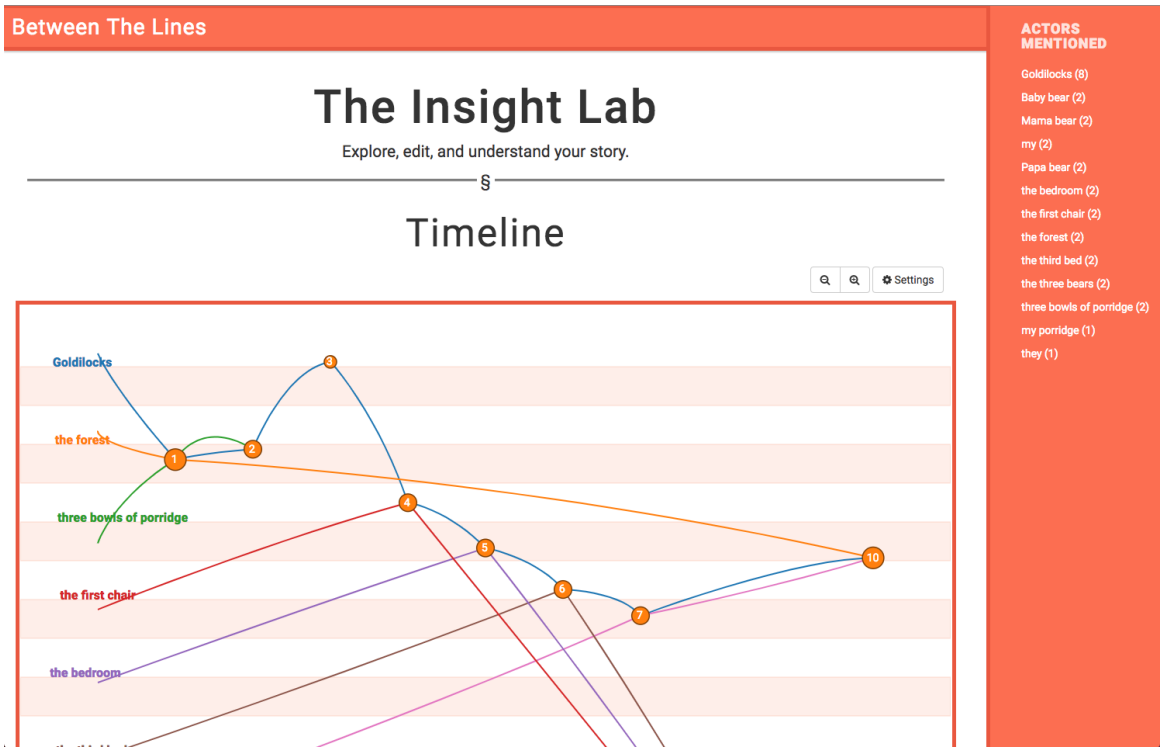
After approximately 1 minute we obtain our results, hence we also incorporated an animated loading screen to ensure the user is aware that the application is actually processing and isn't just hanging. As we can see from Figure 7.1b, the timeline is initially a little messy, containing paths for items such as *the porridge* in addition to our characters of interest. The actor side panel highlights the entities identified, revealing *Goldilocks* to quite clearly make the most frequent appearance throughout the text. Figure 7.2 shows the resulting events produced by the application at three distinct levels of detail. We can see clear distinctions between the events identified in all three cases, yielding what appear to be a fairly natural set of events.

We take the example events of Figure 7.2c as our event set to construct our final timeline. By quickly removing any irrelevant mentions using the *quick-delete* button next to each mention in the side panel, we quickly obtain the timeline shown in Figure 7.3a. From this we can immediately see the interactions between *Goldilocks* and *the three bears*, however, with a little intuition and reading the text it is clear that *the three bears* is actually a collective reference to each of the three bears in the text. Making this quick renaming change, we obtain the final result of Figure 7.3b. Additionally, note that in this example no re-ordering of the actor labels has been necessary to obtain this result.

This was an extremely quick process on this short example. From the resulting timeline we can clearly see the paths of each of the characters in the story and where they interact, and I could easily see such a visualisation as being an extremely useful revision tool, quickly jogging our memory as to the high-level flow of the story. Perhaps one thing that is missing is some additional semantic information to succinctly reflect exactly *what* each event represents to further aid a quick recollection of the events in the underlying text.



(a) Our initial input for the Goldilocks case study, manually specifying the expected characters.



(b) The initial results on *the insights page* for the Goldilocks fairy tale.

Figure 7.1

#	Event text	Actors mentioned
1	Once upon a time, there was a little girl named Goldilocks.	Goldilocks
2	She went for a walk in the forest.	Goldilocks; the forest
3	Pretty soon, she came upon a house.	Goldilocks
4	She knocked and, when no one answered, she walked right in.	Goldilocks
5	At the table in the kitchen, there were three bowls of porridge.	three bowls of porridge
6	Goldilocks was hungry.	Goldilocks
7	She tasted the porridge from the first bowl. "This porridge is too hot!" she exclaimed. So, she tasted the porridge from the second bowl. "This porridge is too cold," she said. So, she tasted the last bowl of porridge.	three bowls of porridge; Goldilocks
8	"Ahhh, this porridge is just right," she said happily and she ate it all up.	three bowls of porridge; Goldilocks
9	After she had eaten the three bears' breakfasts she decided she was feeling a little tired. So, she walked into the living room where she saw three chairs.	Goldilocks
10	Goldilocks sat in the first chair to rest her feet. "This chair is too big!" she exclaimed. So she sat in the second chair. "This chair is too big, too!" she whined.	Goldilocks; the first chair
11	So she tried the last and smallest chair. "Ahhh, this chair is just right," she sighed. But just as she settled down into the chair to rest, it broke into pieces!	Goldilocks; the first chair
12	Goldilocks was very tired by this time, so she went upstairs to the bedroom.	the bedroom; Goldilocks
13	She lay down in the first bed, but it was too hard. Then she lay in the second bed, but it was too soft. Then she lay down in the third bed and it was just right.	Goldilocks; the third bed
14	Goldilocks fell asleep. As she was sleeping, the three bears came home.	Goldilocks; the three bears
15	"Someone's been eating my porridge," growled the Papa bear. "Someone's been eating my porridge," said the Mama bear. "Someone's been eating my porridge and they ate it all up!" cried Baby bear.	my porridge; Papa bear; my; Mama bear; Baby bear
16	"Someone's been sitting in my chair," growled the Papa bear. "Someone's been sitting in my chair," said the Mama bear. "Someone's been sitting in my chair and they've broken it all to pieces," cried Baby bear.	Papa bear; the first chair; my; Mama bear; Baby bear; they
17	They decided to look around some more and when they got upstairs to the bedroom, Papa bear growled, "Someone's been sleeping in my bed". "Someone's been sleeping in my bed, too" said the Mama bear. "Someone's been sleeping in my bed and she's still there!" exclaimed Baby bear.	the bedroom; Papa bear; my; they; Mama bear; the third bed; Baby bear
18	Just then, Goldilocks woke up and saw the three bears.	Goldilocks; the three bears
19	She screamed, "Help!", and she jumped up and ran out of the room. Goldilocks ran down the stairs, opened the door, and ran away into the forest.	Goldilocks; the forest

(a)

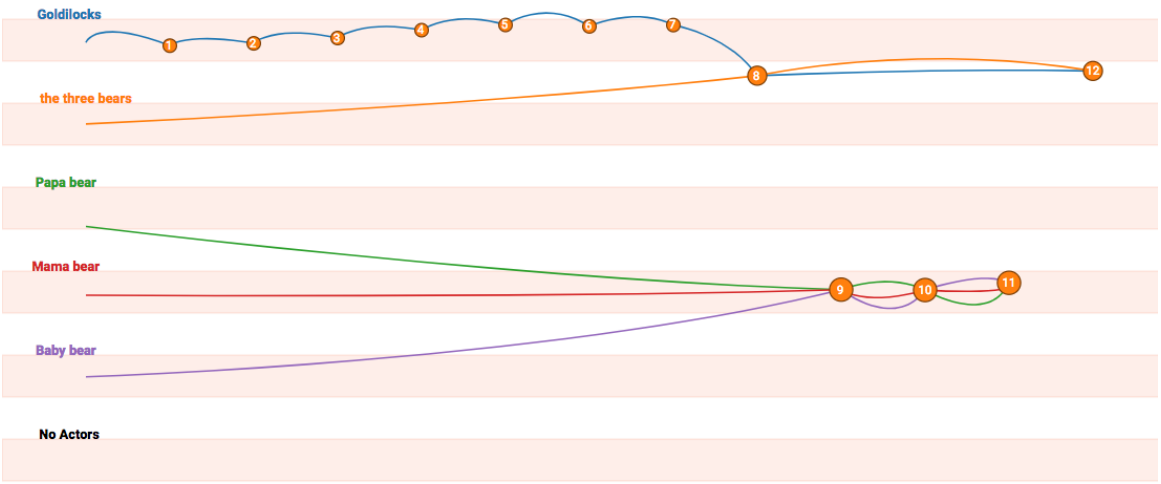
#	Event text	Actors mentioned
1	Once upon a time, there was a little girl named Goldilocks. She went for a walk in the forest. Pretty soon, she came upon a house. She knocked and, when no one answered, she walked right in. At the table in the kitchen, there were three bowls of porridge. Goldilocks was hungry.	Goldilocks; the forest; three bowls of porridge
2	She tasted the porridge from the first bowl. "This porridge is too hot!" she exclaimed. So, she tasted the porridge from the second bowl. "This porridge is too cold," she said. So, she tasted the last bowl of porridge. "Ahhh, this porridge is just right," she said happily and she ate it all up.	three bowls of porridge; Goldilocks
3	After she had eaten the three bears' breakfasts she decided she was feeling a little tired. So, she walked into the living room where she saw three chairs.	Goldilocks
4	Goldilocks sat in the first chair to rest her feet. "This chair is too big!" she exclaimed. So she sat in the second chair. "This chair is too big, too!" she whined. So she tried the last and smallest chair. "Ahhh, this chair is just right," she sighed. But just as she settled down into the chair to rest, it broke into pieces!	Goldilocks; the first chair
5	Goldilocks was very tired by this time, so she went upstairs to the bedroom. She lay down in the first bed, but it was too hard. Then she lay in the second bed, but it was too soft. Then she lay down in the third bed and it was just right. Goldilocks fell asleep. As she was sleeping, the three bears came home.	the bedroom; Goldilocks; the third bed; the three bears
6	"Someone's been eating my porridge," growled the Papa bear. "Someone's been eating my porridge," said the Mama bear. "Someone's been eating my porridge and they ate it all up!" cried Baby bear. "Someone's been sitting in my chair," growled the Papa bear. "Someone's been sitting in my chair," said the Mama bear. "Someone's been sitting in my chair and they've broken it all to pieces," cried Baby bear. They decided to look around some more and when they got upstairs to the bedroom, Papa bear growled, "Someone's been sleeping in my bed". "Someone's been sleeping in my bed, too" said the Mama bear. "Someone's been sleeping in my bed and she's still there!" exclaimed Baby bear.	my porridge; Papa bear; my; Mama bear; Baby bear; the first chair; they; the bedroom; the third bed
7	Just then, Goldilocks woke up and saw the three bears. She screamed, "Help!", and she jumped up and ran out of the room. Goldilocks ran down the stairs, opened the door, and ran away into the forest.	Goldilocks; the three bears; the forest

(b)

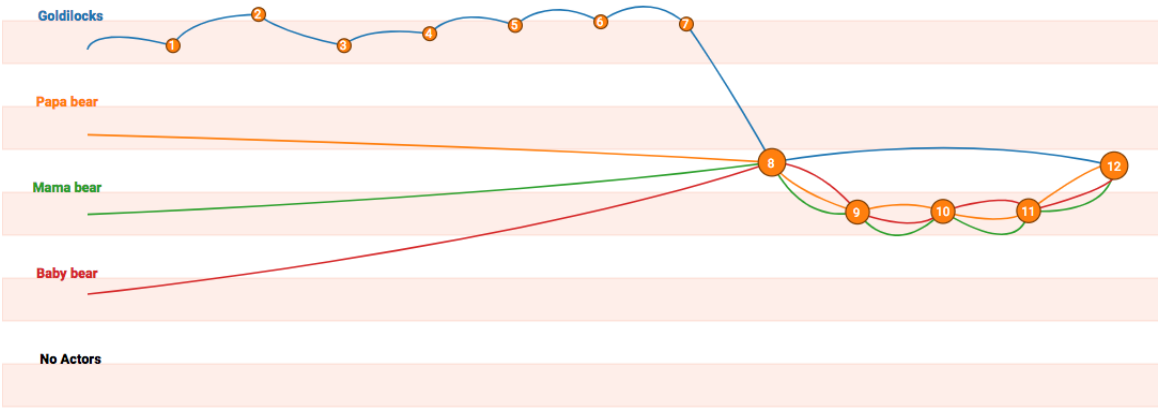
#	Event text	Actors mentioned
1	Once upon a time, there was a little girl named Goldilocks.	Goldilocks
2	She went for a walk in the forest. Pretty soon, she came upon a house. She knocked and, when no one answered, she walked right in. At the table in the kitchen, there were three bowls of porridge. Goldilocks was hungry.	Goldilocks
3	She tasted the porridge from the first bowl. "This porridge is too hot!" she exclaimed. So, she tasted the porridge from the second bowl. "This porridge is too cold," she said. So, she tasted the last bowl of porridge. "Ahhh, this porridge is just right," she said happily and she ate it all up.	Goldilocks
4	After she had eaten the three bears' breakfasts she decided she was feeling a little tired. So, she walked into the living room where she saw three chairs.	Goldilocks
5	Goldilocks sat in the first chair to rest her feet. "This chair is too big!" she exclaimed. So she sat in the second chair. "This chair is too big, too!" she whined. So she tried the last and smallest chair. "Ahhh, this chair is just right," she sighed. But just as she settled down into the chair to rest, it broke into pieces!	Goldilocks
6	Goldilocks was very tired by this time, so she went upstairs to the bedroom.	Goldilocks
7	She lay down in the first bed, but it was too hard. Then she lay in the second bed, but it was too soft. Then she lay down in the third bed and it was just right.	Goldilocks
8	Goldilocks fell asleep. As she was sleeping, the three bears came home.	Goldilocks; the three bears
9	"Someone's been eating my porridge," growled the Papa bear. "Someone's been eating my porridge," said the Mama bear. "Someone's been eating my porridge and they ate it all up!" cried Baby bear.	Papa bear; Mama bear; Baby bear
10	"Someone's been sitting in my chair," growled the Papa bear. "Someone's been sitting in my chair," said the Mama bear. "Someone's been sitting in my chair and they've broken it all to pieces," cried Baby bear.	Papa bear; Mama bear; Baby bear
11	They decided to look around some more and when they got upstairs to the bedroom, Papa bear growled, "Someone's been sleeping in my bed". "Someone's been sleeping in my bed, too" said the Mama bear. "Someone's been sleeping in my bed and she's still there!" exclaimed Baby bear.	Papa bear; Mama bear; Baby bear
12	Just then, Goldilocks woke up and saw the three bears. She screamed, "Help!", and she jumped up and ran out of the room. Goldilocks ran down the stairs, opened the door, and ran away into the forest.	Goldilocks; the three bears

(c)

Figure 7.2: The events identified at three levels of granularity in the Goldilocks fairy tale. (a) The event set at the *extra-fine* level of detail. (b) The event set at the *Coarse* level of granularity. (c) The event set we take to create our resulting timeline, found slightly before the automatically identified *fine-grained* event set, and yielding a nice balance between the two other event sets.



(a)



(b)

Figure 7.3: The resulting timelines produced after a couple of manual edits. (a) is produced by simply deleting any irrelevant mentions. (b) is produced by additionally renaming *the three bears* appropriately.

7.1.2 Little Red Riding Hood

For this example we omit the specification of the actors expected in the text to demonstrate the results without any pre-processing taking place. Additionally, in texts like this where we see characters referred to by multiple different names, for example in this text we see *Little Red Riding Hood* also referred to as *the little girl*, renaming these characters can adversely affect the ability of the coreference resolution system to recognise these entities as the same. Thus, Figure 7.4a shows our initial input.

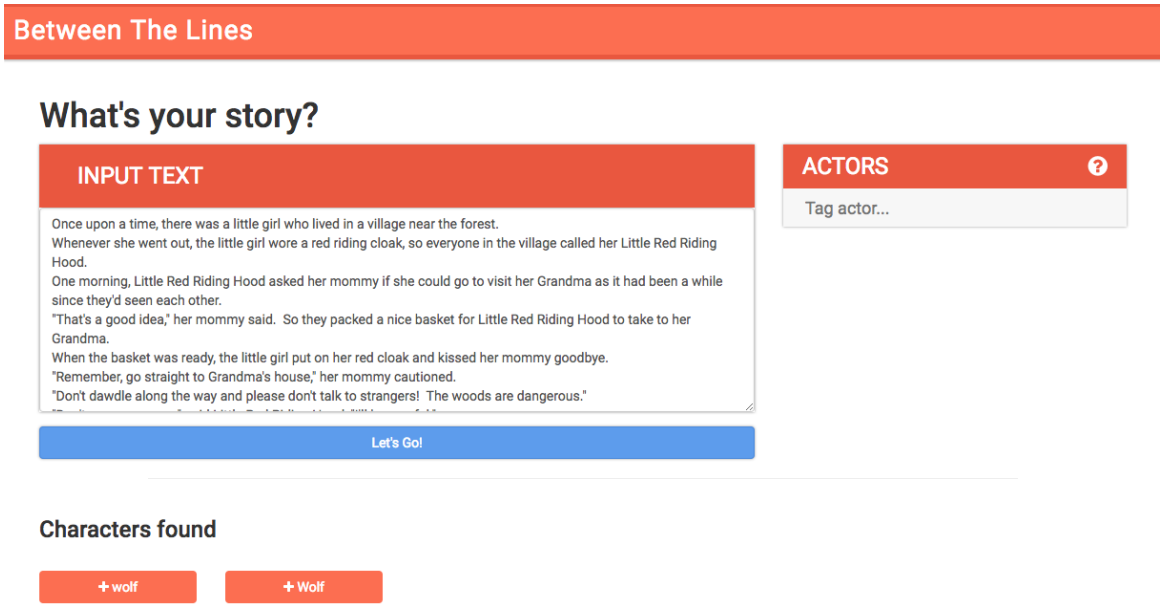
We again end up with a rather messy initial timeline, shown in Figure 7.4b, where the actor side panel reveals that we indeed see multiple distinct mentions that actually refer to the same entity. This occurs in the case of *Little Red Riding Hood* and *a little girl who lived in a village near the forest*, and also in the case of *her Grandma* and *Poor Grandma*. Additionally, we again see a number of other mentions included in the initial timeline that can be immediately removed by the user. Despite this, Figure 7.5 shows that we obtain a relatively nice set of events at both the *fine* and *coarse* levels of granularity, with a clear distinction between disjoint events. The coarse grained events can be summarised quite nicely as follows:

1. *Little Red Riding Hood suggests to her mother that she goes visit her Grandma.*
2. *Her mum helped her pack-up and sent her off to Grandma's with a warning.*
3. *Little Red Riding Hood soon forgets her promise to her mother.*
4. *The wolf finds out Little Red Riding Hood is on her way to her Grandma's and rushes to get there first.*
5. *Poor Grandma is eaten by the wolf.*
6. *Little Red Riding Hood arrives at her Grandma's house.*
7. *Little Red Riding Hood begins to notice something odd about her Grandma.*
8. *Little Red Riding Hood realises that she's actually talking to the wolf, and runs out of the cottage.*
9. *The woodsman saves Little Red Riding Hood and her Grandma from the wolf.*

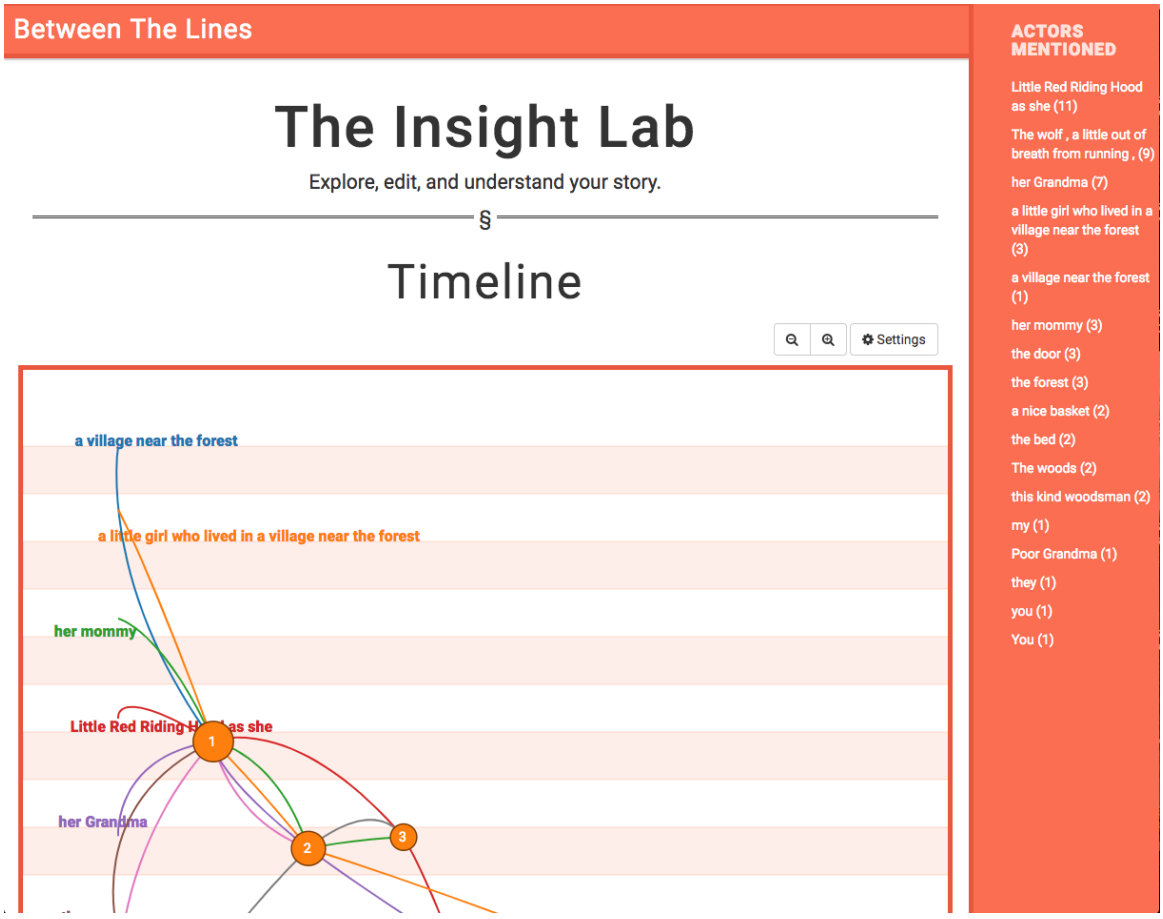
This example presents a real test to the underlying coreference resolution system with no named characters and instead a large number of *nominal* mentions. As a result, we see the results of clustering are affected slightly, drawing event boundaries perhaps in slightly different places to a reader naturally would, as we explore in more detail shortly in our evaluation of the clustering performance. However, a natural artefact of increasing the coarseness of the events used is that it reduces the effect of any missed coreferences; despite some sentences missing some mention annotations, the events as a whole tend to include all the correct mentions. As a result, relatively little manual adjustment of the mention annotations of each event is required in order to construct an accurate timeline from a set of coarse-grained events. We thus take the coarse-grained event set to produce our timeline.

Renaming the distinct mentions of *Little Red Riding Hood* as a single entity and doing the same for the two distinct mentions of her *Grandma*, we immediately get the timeline of Figure 7.6a. However, this example also shows the significance of label ordering in the resulting timeline, where by moving the label of *the woodsman* to the bottom of the timeline we obtain a far clearer result, shown in Figure 7.6b. This timeline again quite clearly shows the role played by each of the characters in the text, and serves as a quick reminder of the overall plot. For example, we see the *mum's* relatively insignificant role at the beginning of the story where she sends *Red Riding Hood* off to her Grandma's, and we can additionally see that the *woodsman* plays a short role at the end of the story in two events that clearly involve all four of *Red Riding Hood*, *Grandma*, *the Wolf*, and the *woodsman*.

Additionally, in this slightly more complex example we can also see the investigative capabilities of our interactive timeline. By highlighting the path of *Little Red Riding Hood* only, as we do in Figure 7.7, our attention is immediately drawn to the *one* event she is not involved in. Looking into this event in more detail we find that this is exactly the point at which *the wolf* eats her *Grandma*. Thus, not only does this serve as a particularly useful revision tool for recollecting the series of events in the text at a particular level of detail, but we can also see the potential that interactive exploration of such a representation may have in the context of witness statements or other analytical contexts.



(a) Our initial input for the Little Red Riding Hood case study.



(b) The initial results on *the insights page* for the Little Red Riding Hood fairy tale.

Figure 7.4

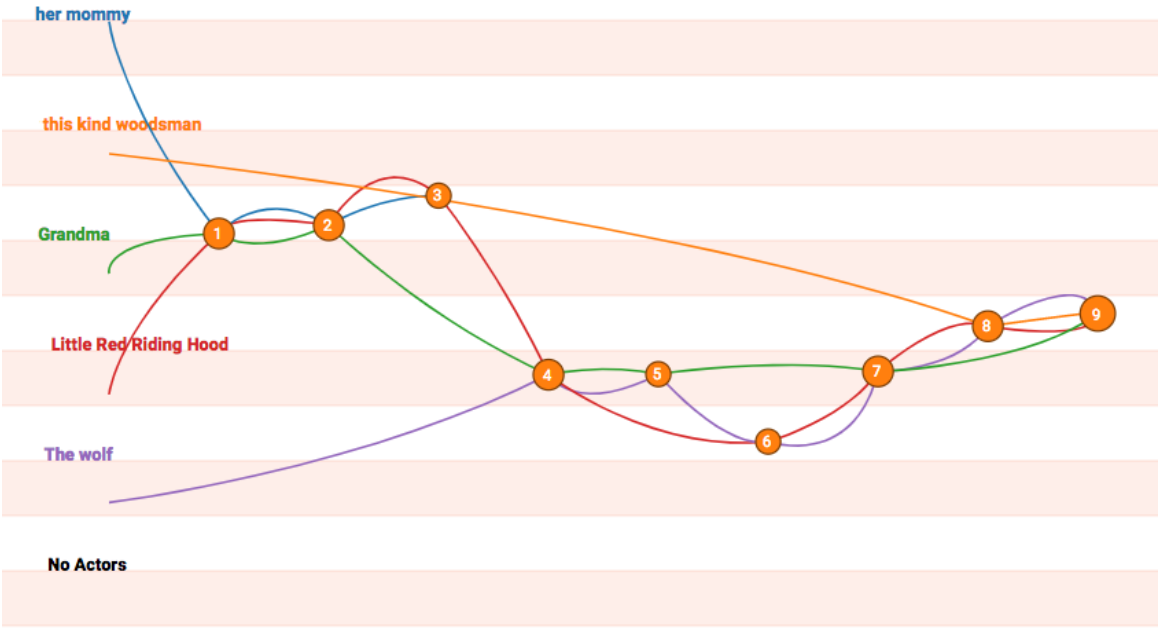
#	Event text	Actors mentioned
1	Once upon a time, there was a little girl who lived in a village near the forest. Whenever she went out, the little girl wore a red riding cloak, so everyone in the village called her Little Red Riding Hood. One morning, Little Red Riding Hood asked her mommy if she could go to visit her Grandma as it had been a while since they'd seen each other. "That's a good idea," her mommy said. So they packed a nice basket for Little Red Riding Hood to take to her Grandma.	a village near the forest; a little girl who lived in a village near the forest; her mommy; Little Red Riding Hood as she; her Grandma; they; a nice basket
2	When the basket was ready, the little girl put on her red cloak and kissed her mommy goodbye. "Remember, go straight to Grandma's house," her mommy cautioned. "Don't dawdle along the way and please don't talk to strangers! The woods are dangerous."	a nice basket; a little girl who lived in a village near the forest; her mommy; her Grandma; The woods
3	"Don't worry, mommy," said Little Red Riding Hood, "I'll be careful." But when Little Red Riding Hood noticed some lovely flowers in the woods, she forgot her promise to her mommy. She picked a few, watched the butterflies flit about for a while, listened to the frogs croaking and then picked a few more.	Little Red Riding Hood as she; her mommy; The woods
4	Little Red Riding Hood was enjoying the warm summer day so much, that she didn't notice a dark shadow approaching out of the forest behind her. Suddenly, the wolf appeared beside her. "What are you doing out here, little girl?" the wolf asked in a voice as friendly as he could muster.	Little Red Riding Hood as she; the forest; The wolf , a little out of breath from running ,
5	"I'm on my way to see my Grandma who lives through the forest, near the brook," Little Red Riding Hood replied. Then she realized how late she was and quickly excused herself, rushing down the path to her Grandma's house.	The wolf , a little out of breath from running ,; Little Red Riding Hood as she; the forest; her Grandma
6	The wolf, in the meantime, took a shortcut. The wolf, a little out of breath from running, arrived at Grandma's and knocked lightly at the door. "Oh thank goodness dear! Come in, come in! I was worried sick that something had happened to you in the forest," said Grandma thinking that the knock was her granddaughter. The wolf let himself in.	The wolf , a little out of breath from running ,; the door; her Grandma; Little Red Riding Hood as she; the forest
7	Poor Grandma did not have time to say another word, before the wolf gobbled her up! The wolf let out a satisfied burp, and then poked through Grandma's wardrobe to find a nightgown that he liked. He added a frilly sleeping cap, and for good measure, dabbed some of Grandma's perfume behind his pointy ears.	The wolf , a little out of breath from running ,; Poor Grandma; her Grandma
8	A few minutes later, Red Riding Hood knocked on the door.	the door; Little Red Riding Hood as she
9	The wolf jumped into bed and pulled the covers over his nose. "Who is it?" he called in a cackly voice. "It's me, Little Red Riding Hood." "Oh how lovely! Do come in, my dear," croaked the wolf.	The wolf , a little out of breath from running ,; Little Red Riding Hood as she
10	When Little Red Riding Hood entered the little cottage, she could scarcely recognize her Grandma. "Grandma! Your voice sounds so odd. Is something the matter?" she asked. "Oh, I just have touch of a cold," squeaked the wolf adding a cough at the end to prove the point. "But Grandma! What big ears you have," said Little Red Riding Hood as she edged closer to the bed.	Little Red Riding Hood as she; her Grandma; the bed
11	"The better to hear you with, my dear," replied the wolf. "But Grandma! What big eyes you have," said Little Red Riding Hood. "The better to see you with, my dear," replied the wolf.	The wolf , a little out of breath from running ,; my; you; Little Red Riding Hood as she
12	"But Grandma! What big teeth you have," said Little Red Riding Hood her voice quivering slightly. "The better to eat you with, my dear," roared the wolf and he leapt out of the bed and began to chase the little girl. Almost too late, Little Red Riding Hood realized that the person in the bed was not her Grandma, but a hungry wolf. She ran across the room and through the door, shouting, "Help! Wolf!" as loudly as she could. A woodsman who was chopping logs nearby heard her cry and ran towards the cottage as fast as he could.	Little Red Riding Hood as she; The wolf , a little out of breath from running ,; the bed; a little girl who lived in a village near the forest; the door; this kind woodsman
13	He grabbed the wolf and made him spit out the poor Grandma who was a bit frazzled by the whole experience, but still in one piece.	The wolf , a little out of breath from running ,
14	"Oh Grandma, I was so scared!" sobbed Little Red Riding Hood, "I'll never speak to strangers or dawdle in the forest again." "There, there, child. You've learned an important lesson. Thank goodness you shouted loud enough for this kind woodsman to hear you!" The woodsman knocked out the wolf and carried him deep into the forest where he wouldn't bother people any longer. Little Red Riding Hood and her Grandma had a nice lunch and a long chat.	Little Red Riding Hood as she; this kind woodsman; The wolf , a little out of breath from running ,; You; her Grandma

(a) The automatically identified set of *fine-grained* events for Little Red Riding Hood.

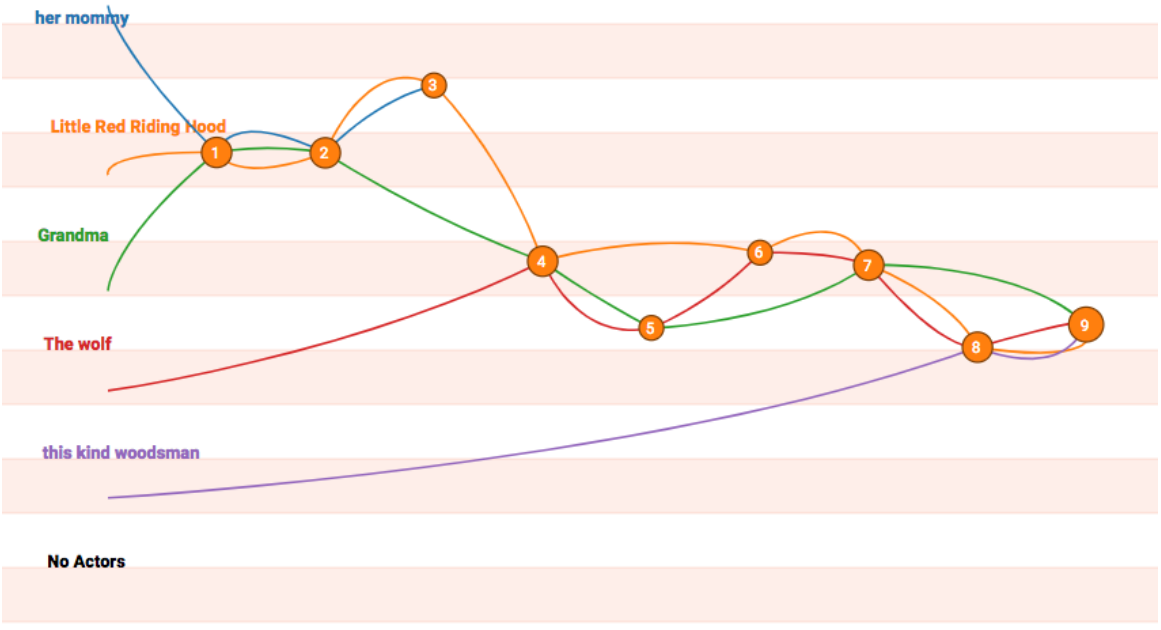
#	Event text	Actors mentioned
1	Once upon a time, there was a little girl who lived in a village near the forest. Whenever she went out, the little girl wore a red riding cloak, so everyone in the village called her Little Red Riding Hood. One morning, Little Red Riding Hood asked her mommy if she could go to visit her Grandma as it had been a while since they'd seen each other. "That's a good idea," her mommy said. So they packed a nice basket for Little Red Riding Hood to take to her Grandma.	Little Red Riding Hood; her mommy; Grandma
2	When the basket was ready, the little girl put on her red cloak and kissed her mommy goodbye. "Remember, go straight to Grandma's house," her mommy cautioned. "Don't dawdle along the way and please don't talk to strangers! The woods are dangerous."	Little Red Riding Hood; her mommy; Grandma
3	"Don't worry, mommy," said Little Red Riding Hood, "I'll be careful." But when Little Red Riding Hood noticed some lovely flowers in the woods, she forgot her promise to her mommy. She picked a few, watched the butterflies flit about for a while, listened to the frogs croaking and then picked a few more.	Little Red Riding Hood; her mommy
4	Little Red Riding Hood was enjoying the warm summer day so much, that she didn't notice a dark shadow approaching out of the forest behind her. Suddenly, the wolf appeared beside her. "What are you doing out here, little girl?" the wolf asked in a voice as friendly as he could muster. "I'm on my way to see my Grandma who lives through the forest, near the brook," Little Red Riding Hood replied. Then she realized how late she was and quickly excused herself, rushing down the path to her Grandma's house. The wolf, in the meantime, took a shortcut. The wolf, a little out of breath from running, arrived at Grandma's and knocked lightly at the door. "Oh thank goodness dear! Come in, come in! I was worried sick that something had happened to you in the forest," said Grandma thinking that the knock was her granddaughter. The wolf let himself in.	Little Red Riding Hood; The wolf; Grandma
5	Poor Grandma did not have time to say another word, before the wolf gobbled her up! The wolf let out a satisfied burp, and then poked through Grandma's wardrobe to find a nightgown that he liked. He added a frilly sleeping cap, and for good measure, dabbed some of Grandma's perfume behind his pointy ears.	Grandma; The wolf
6	A few minutes later, Red Riding Hood knocked on the door. The wolf jumped into bed and pulled the covers over his nose. "Who is it?" he called in a cackly voice. "It's me, Little Red Riding Hood." "Oh how lovely! Do come in, my dear," croaked the wolf.	Little Red Riding Hood; The wolf
7	When Little Red Riding Hood entered the little cottage, she could scarcely recognize her Grandma. "Grandma! Your voice sounds so odd. Is something the matter?" she asked. "Oh, I just have touch of a cold," squeaked the wolf adding a cough at the end to prove the point. "But Grandma! What big ears you have," said Little Red Riding Hood as she edged closer to the bed. "The better to hear you with, my dear," replied the wolf. "But Grandma! What big eyes you have," said Little Red Riding Hood. "The better to see you with, my dear," replied the wolf.	Grandma; Little Red Riding Hood; The wolf
8	"But Grandma! What big teeth you have," said Little Red Riding Hood her voice quivering slightly. "The better to eat you with, my dear," roared the wolf and he leapt out of the bed and began to chase the little girl. Almost too late, Little Red Riding Hood realized that the person in the bed was not her Grandma, but a hungry wolf. She ran across the room and through the door, shouting, "Help! Wolf!" as loudly as she could. A woodsman who was chopping logs nearby heard her cry and ran towards the cottage as fast as he could.	Little Red Riding Hood; The wolf; this kind woodsman
9	He grabbed the wolf and made him spit out the poor Grandma who was a bit frazzled by the whole experience, but still in one piece. "Oh Grandma, I was so scared!" sobbed Little Red Riding Hood, "I'll never speak to strangers or dawdle in the forest again." "There, there, child. You've learned an important lesson. Thank goodness you shouted loud enough for this kind woodsman to hear you!" The woodsman knocked out the wolf and carried him deep into the forest where he wouldn't bother people any longer. Little Red Riding Hood and her Grandma had a nice lunch and a long chat.	The wolf; Little Red Riding Hood; this kind woodsman; Grandma

(b) The automatically identified set of *coarse-grained* events for Little Red Riding Hood.

Figure 7.5



(a) The initial timeline produced having removed any irrelevant mentions from the actor side panel and having renamed the duplicate mentions of *Little Red Riding Hood* and *Grandma*.



(b) The timeline produced after re-ordering the actor labels in (a).

Figure 7.6

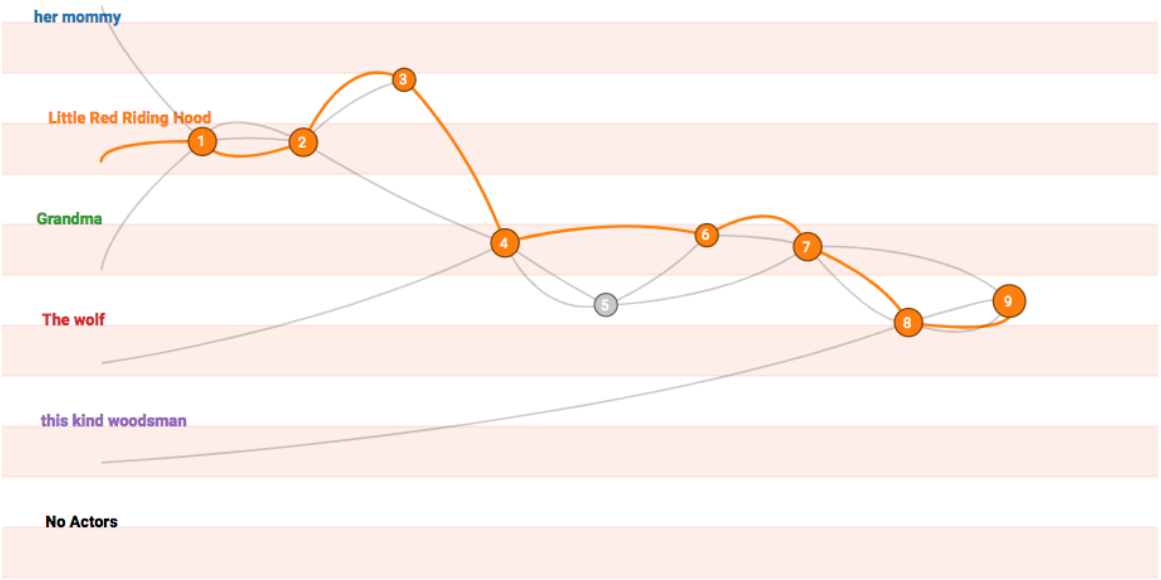


Figure 7.7: The resulting timeline having chosen to see only the path of *Little Red Riding Hood* through the story, highlighting the investigative capabilities of the interactive timeline. Event 5 in this case being the point where *the wolf* eats *Grandma*.

7.1.3 Harry Potter and the Philosopher's Stone

Finally, we take a short extract from the first chapter of *Harry Potter and the Philosopher's Stone* [68]. This is used to explore how well our approach generalises to other narratives beyond fairy tales. Pre-processing is particularly important in this case to help with recognising coreferences to characters of the form *Mr. x* and *Mrs. x*. Our automatic character recognition stage also proves helpful in this example, immediately recognising a number of characters from the text on our behalf, as shown in Figure 7.8a.

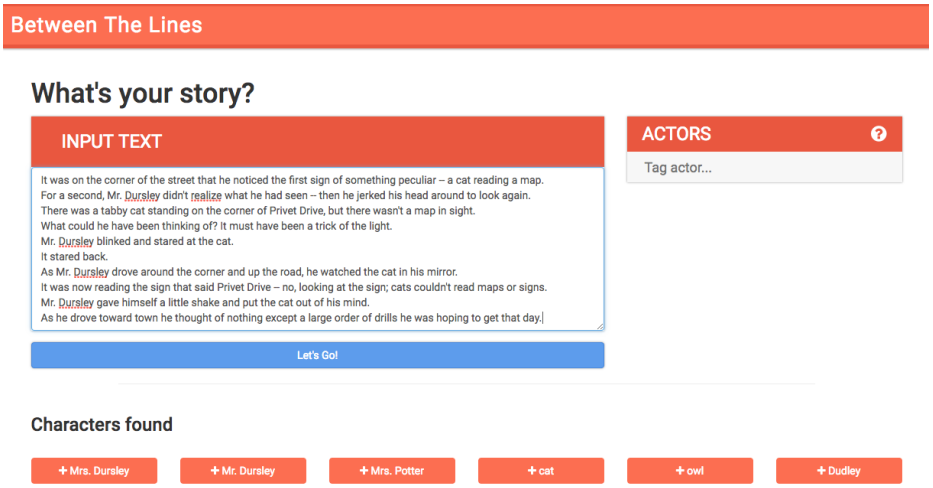
Having selected all of these characters to tag explicitly, we obtain the initial results shown in Figure 7.8b. This again shows a number of non-actor mentions present in the actor side panel that can quickly be removed. We also see there are a number of related mentions listed:

- *Mr. and Mrs. Dursley*
- *The Dursleys*
- *Mr. Dursley*
- *Mrs. Dursley*
- *Dudley*

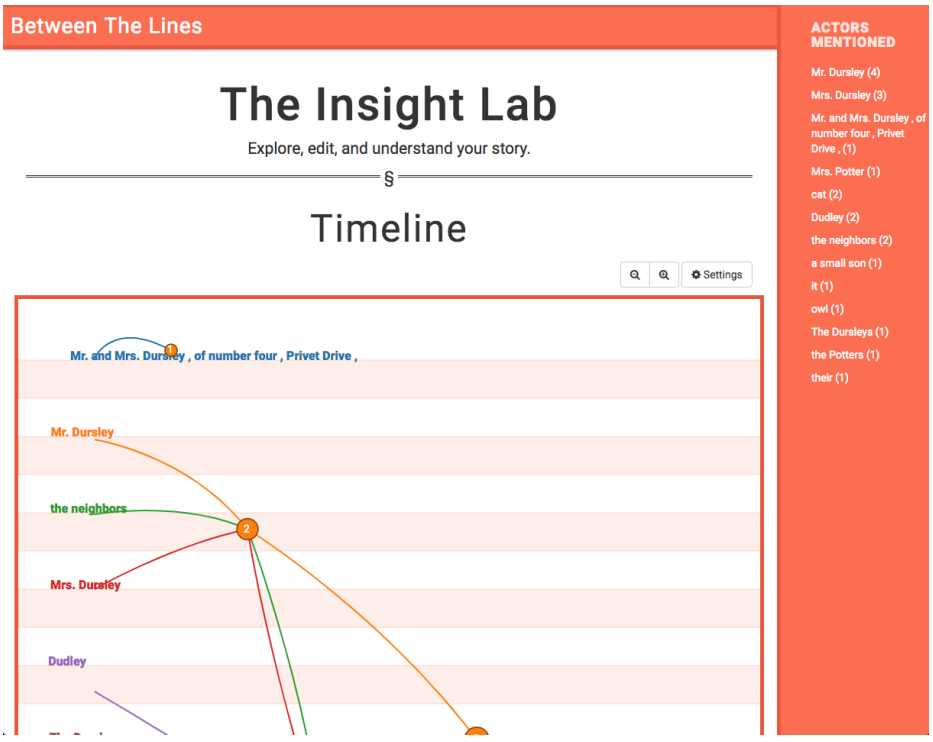
It is understandable that the coreference resolution system has left these as distinct entities, however for our timeline we'd like these mentions to reflect the individual actors they encompass rather than leaving them as distinct entities. A quick edit via the *edit-actor* panel allows us to quickly rename "*Mr. and Mrs. Dursley*" to its constituent entities: "*Mr. Dursley*" and "*Mrs. Dursley*" in *2* clicks: selecting the mention to rename in the side-panel, typing the new names, and clicking *Save*. Thus, we can soon update these mentions to reflect only the actors of interest to us.

Figures 7.9 and 7.10 show the resulting event sets at the *extra-fine* and *coarse* levels of detail. We see that both of these event sets are nicely grouped based upon the topic of discussion in each event. This result highlights the use of repetitive language even in more sophisticated novels, allowing our clustering algorithm to identify natural boundaries between distinct events. However, it does appear that the coarse set of events are perhaps a little bloated, and we may have perhaps liked all the sentences regarding *the cat* to have been grouped, yet despite this we still obtain a fairly good set of events. Event 5's lack of relationship to its surrounding sentences causes it to act as a barrier here, and leaves this event distinct. A facet of our clustering approach that could perhaps be improved.

Finally, generating the timeline for the extra-fine grained set of events identified yields the result shown in Figure 7.11a. While this may appear a little tangled due to the effect of having more characters involved during a short period of time, we can still quite clearly see at which points each character is introduced to the story. Additionally, it is quite clear that the text concludes with a number of events focussing on *Mr. Dursley*, and his interaction with a cat. Highlighting the main characters here, as shown in Figure 7.11b, again helps us to focus on the main protagonists of the plot and perhaps identify further points of interest.



(a) Our initial input and the automatically identified characters in our brief extract from Harry Potter and the Philosopher's Stone.



(b) The initial results on *the insights page* for the Harry Potter text.

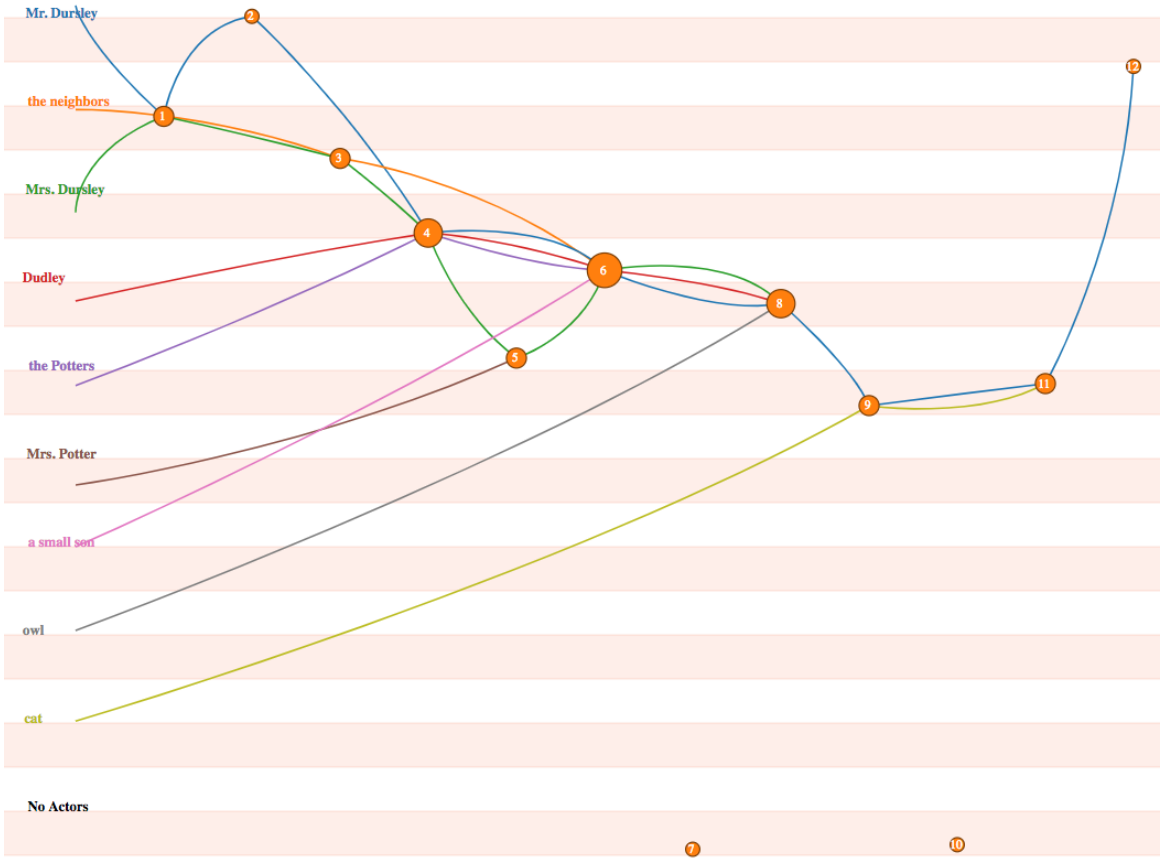
Figure 7.8

#	Event text	Actors mentioned
1	Mr. and Mrs. Dursley, of number four, Privet Drive, were proud to say that they were perfectly normal, thank you very much. They were the last people you'd expect to be involved in anything strange or mysterious, because they just didn't hold with such nonsense.	Mr. and Mrs. Dursley , of number four , Privet Drive ,
2	Mr. Dursley was the director of a firm called Grunnings, which made drills. He was a big, beefy man with hardly any neck, although he did have a very large mustache.	Mr. Dursley
3	Mrs. Dursley was thin and blonde and had nearly twice the usual amount of neck, which came in very useful as she spent so much of her time craning over garden fences, spying on the neighbors.	the neighbors; Mrs. Dursley
4	The Dursleys had a small son called Dudley and in their opinion there was no finer boy anywhere. The Dursleys had everything they wanted, but they also had a secret, and their greatest fear was that somebody would discover it. They didn't think they could bear it if anyone found out about the Potters.	Dudley; The Dursleys; their; it; the Potters
5	Mrs. Potter was Mrs. Dursley's sister, but they hadn't met for several years; in fact, Mrs. Dursley pretended she didn't have a sister, because her sister and her good-for-nothing husband were as unDursleyish as it was possible to be.	Mrs. Potter; their; Mrs. Dursley
6	The Dursleys shuddered to think what the neighbors would say if the Potters arrived in the street. The Dursleys knew that the Potters had a small son, too, but they had never even seen him. This boy was another good reason for keeping the Potters away; they didn't want Dudley mixing with a child like that.	The Dursleys; the neighbors; the Potters; a small son; Dudley
7	When Mr. and Mrs. Dursley woke up on the dull, gray Tuesday our story starts, there was nothing about the cloudy sky outside to suggest that strange and mysterious things would soon be happening all over the country.	
8	Mr. Dursley hummed as he picked out his most boring tie for work, and Mrs. Dursley gossiped away happily as she wrestled a screaming Dudley into his high chair. None of them noticed a large, tawny owl flutter past the window. At half past eight, Mr. Dursley picked up his briefcase, pecked Mrs. Dursley on the cheek, and tried to kiss Dudley good-bye but missed, because Dudley was now having a tantrum and throwing his cereal at the walls.	Dudley; Mr. Dursley; Mrs. Dursley; owl
9	"Little tyke," chortled Mr. Dursley as he left the house. He got into his car and backed out of number four's drive. It was on the corner of the street that he noticed the first sign of something peculiar -- a cat reading a map. For a second, Mr. Dursley didn't realize what he had seen -- then he jerked his head around to look again. There was a tabby cat standing on the corner of Privet Drive, but there wasn't a map in sight. What could he have been thinking of?	Mr. Dursley; cat
10	It must have been a trick of the light.	
11	Mr. Dursley blinked and stared at the cat. It stared back. As Mr. Dursley drove around the corner and up the road, he watched the cat in his mirror. It was now reading the sign that said Privet Drive -- no, looking at the sign; cats couldn't read maps or signs. Mr. Dursley gave himself a little shake and put the cat out of his mind.	cat; Mr. Dursley
12	As he drove toward town he thought of nothing except a large order of drills he was hoping to get that day.	Mr. Dursley

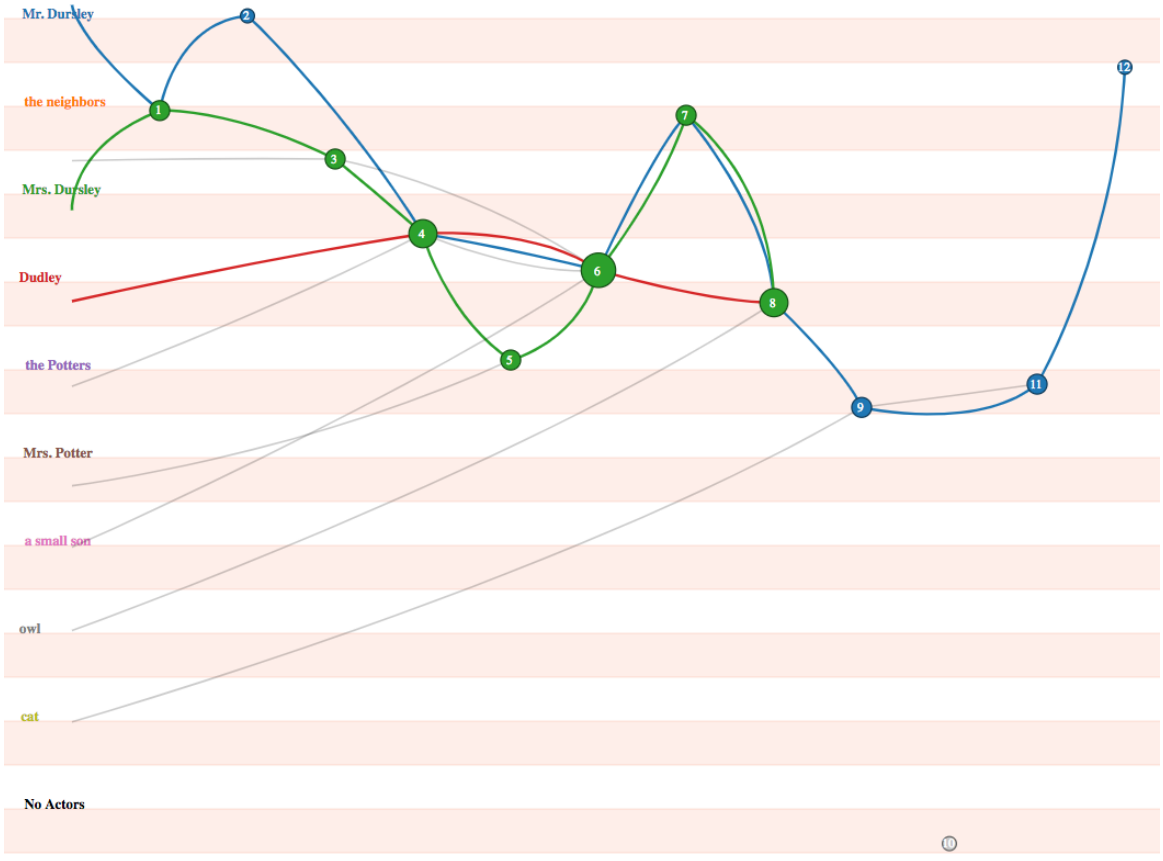
Figure 7.9: The set of automatically identified events at the *extra-fine* level of detail on the Harry Potter text.

#	Event text	Actors mentioned
1	Mr. and Mrs. Dursley, of number four, Privet Drive, were proud to say that they were perfectly normal, thank you very much. They were the last people you'd expect to be involved in anything strange or mysterious, because they just didn't hold with such nonsense.	Mr. and Mrs. Dursley , of number four , Privet Drive ,
2	Mr. Dursley was the director of a firm called Grunnings, which made drills. He was a big, beefy man with hardly any neck, although he did have a very large mustache. Mrs. Dursley was thin and blonde and had nearly twice the usual amount of neck, which came in very useful as she spent so much of her time craning over garden fences, spying on the neighbors.	Mr. Dursley; the neighbors; Mrs. Dursley
3	The Dursleys had a small son called Dudley and in their opinion there was no finer boy anywhere. The Dursleys had everything they wanted, but they also had a secret, and their greatest fear was that somebody would discover it. They didn't think they could bear it if anyone found out about the Potters. Mrs. Potter was Mrs. Dursley's sister, but they hadn't met for several years; in fact, Mrs. Dursley pretended she didn't have a sister, because her sister and her good-for-nothing husband were as unDursleyish as it was possible to be. The Dursleys shuddered to think what the neighbors would say if the Potters arrived in the street. The Dursleys knew that the Potters had a small son, too, but they had never even seen him. This boy was another good reason for keeping the Potters away; they didn't want Dudley mixing with a child like that. When Mr. and Mrs. Dursley woke up on the dull, gray Tuesday our story starts, there was nothing about the cloudy sky outside to suggest that strange and mysterious things would soon be happening all over the country.	Dudley; The Dursleys; their; it; the Potters; Mrs. Potter; Mrs. Dursley; the neighbors; a small son
4	Mr. Dursley hummed as he picked out his most boring tie for work, and Mrs. Dursley gossiped away happily as she wrestled a screaming Dudley into his high chair. None of them noticed a large, tawny owl flutter past the window. At half past eight, Mr. Dursley picked up his briefcase, pecked Mrs. Dursley on the cheek, and tried to kiss Dudley good-bye but missed, because Dudley was now having a tantrum and throwing his cereal at the walls. "Little tyke," chortled Mr. Dursley as he left the house. He got into his car and backed out of number four's drive. It was on the corner of the street that he noticed the first sign of something peculiar – a cat reading a map. For a second, Mr. Dursley didn't realize what he had seen – then he jerked his head around to look again. There was a tabby cat standing on the corner of Privet Drive, but there wasn't a map in sight. What could he have been thinking of?	Dudley; Mr. Dursley; Mrs. Dursley; owl; cat
5	It must have been a trick of the light.	
6	Mr. Dursley blinked and stared at the cat. It stared back. As Mr. Dursley drove around the corner and up the road, he watched the cat in his mirror. It was now reading the sign that said Privet Drive – no, looking at the sign; cats couldn't read maps or signs. Mr. Dursley gave himself a little shake and put the cat out of his mind. As he drove toward town he thought of nothing except a large order of drills he was hoping to get that day.	cat; Mr. Dursley

Figure 7.10: The set of automatically identified events at the *coarse* level of detail on the Harry Potter text.



(a) The timeline produced for the Harry Potter text having removed any non-actor mentions and renamed *Mr. and Mrs. Dursely* and *The Dursleys* to their constituent entities.



(b) Highlighting the paths of *Mr. Dursley*, *Mrs. Dursley*, and *Dudley* through the text.

Figure 7.11

7.2 Evaluation Overview

While the case studies above provide a useful overview of the application as a whole and demonstrate the potential of our resulting application, they fail to comprehensively evaluate the performance of our application. Thus, in the following sections we evaluate each component of our application individually, and attempt to quantify the performance of the system.

We finally reflect on some the overall strengths and weakness of our application in Section 7.6.

7.3 Event Clustering Evaluation

7.3.1 Aims

The aim here is to judge the *intuitiveness* of the events produced using our modified hierarchical clustering method by comparing our automated results with the events identified by real people. To perform this evaluation we enlisted the help of a number of volunteers, giving each participant 2 fairy tales and our Harry Potter extract from earlier and asked them to draw a line between the sentences where they'd identify a change of *event*. i.e. Breaking down the text into a series of events, similarly to our event clustering algorithm. Each participant was asked to repeat the process considering events at a fine level of detail, and then at a more general level of detail if they were to think about events at a higher level. However, we did not enforce a specific definition of a *fine-grained* or *coarse-grained* event. This is to allow us to judge how well our algorithm could adapt to different users' opinions as to what they consider to be the events in a narrative at different levels of detail.

7.3.2 Evaluation Metrics

To quantify the *intuitiveness* of the events extracted by our application, we employ the following metrics, treating the event set identified by the test subjects as our reference set:

- **Recall** *Proportion of sentences grouped in the reference set that are also contained within the same event in our system-produced results.*
- **Precision** *Of the events identified by the system, what proportion of the sentences are correctly grouped within the same event.*
- **Rand Index** *A measure of the similarity between two cluster sets, from 0 (complete dissimilarity) to 1 (complete similarity)* [69].

All of these metrics can be computed using the same basic 4 values: *true positives* (TP), *false positives* (FP), *true negatives* (TN), and *false negatives* (FN), which we summarise in Table 7.1 below.

	Same system event	**Different system event**
Same user event	TP	FN
Different user event	FP	TN

Table 7.1: Classification of a pair of sentences into one of the four possible categories.

That is, this evaluation is performed at the level of sentence pairs. For example, calculating the number of *true positives* is simply a matter of calculating the number of pairs of sentences that are contained within a single event in our system-produced results and are also present within a single event in our reference set. The number of *false positives* is, in contrast, the number of sentence pairs that appear within a single event in our result set, yet have a boundary between them in the reference set of events. The other classifications follow similarly.

Having calculated these 4 fundamental values, we can then compute our three metrics as follows:

$$Recall = \frac{TP}{TP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$RandIndex = \frac{TP + TN}{TP + FP + TN + FN}$$

7.3.3 Results and Remarks

Our results are presented in Table 7.2, where we list the resulting metrics for each sample text individually at each level of granularity and additionally average the entire results to get an overall view of the efficacy of our clustering method. Each value presented is the average result from the reference events produced by all test participants. Additionally, before calculating the metrics identified above, we first found the most similar cluster set to the reference set in the cluster hierarchy. Thus, these results reflect the intuitiveness of the closest possible event set to the user's set; if the cluster set produced by our application at some point were to match exactly the set of events identified by the user, then all scores should reflect a 100% accuracy.

	Recall	**Precision**	**Rand Index**
Goldilocks (fine)	0.45	0.39	0.98
Goldilocks (coarse)	0.75	0.75	0.91
The Gingerbread Man (fine)	0.6	0.47	0.97
The Gingerbread Man (coarse)	0.6	0.56	0.92
Harry Potter (fine)	0.36	0.6	0.85
Harry Potter (coarse)	0.58	0.2	0.919
Average	***0.56***	***0.49***	***0.92***

Table 7.2: Hierarchical Clustering performance results over three example texts.

From the table above we can immediately see the general difference in performance between the *fine-grained* events and the *coarse-grained* events. Our particular examples were all relatively short texts, and as a result the majority of participants identified fine-grained events at almost the level of individual sentences, only ever grouping at most two or three sentences into a single event. This makes the resulting figures very sensitive to the event boundaries as there is little margin for error. However, the majority of sentences are indeed correctly separated at this level of granularity, which is what contributes to such high *rand index* scores at the fine-level of granularity. This is a natural artefact of our constraint to only consider merges within a pre-defined lookahead radius of each sentence, significantly reducing the number of possible merges and slightly skewing this metric as we largely avoid catastrophically different results. Thus, recall and precision better reflect the mismatched event boundaries at this level of detail.

At a coarser level of detail we see far better results, largely down to the increased margin of error: the results are not affected as much if we draw our event boundaries a sentence earlier or later than the reference set if the majority of sentences are still clustered in the same way as the reference set. Additionally, unlike at the fine-level of granularity where every single boundary matters, as we reach a coarser level of granularity our results rely less upon the accuracy of our underlying data extraction tools. Consequently, the results are not impacted as greatly by the inaccuracies of coreference resolution. Additionally, as the texts increase in size we again see that participants start to group the text into larger events, revealing that our current implementation obtains a fairly similar set of events to those expected on the whole.

Again, the subjective nature of this topic must be taken into account as everyone has there own opinion as to what they'd identify as the *events* in a text. In our experimentation we obtained peak values of 0.905 and 0.711 for recall and precision, respectively. Overall, our modified clustering algorithm appears to yield a fairly natural set of events, achieving a relatively good balance between recall and precision in the majority of cases and closely emulating the set of events identified by test subjects.

7.4 Timeline Visualisation Evaluation

7.4.1 Aims

The idea of this evaluation is to judge the *clarity* and *expressiveness* of the resulting timelines themselves when the user does not have the corresponding text in front of them. Are the timelines expressive enough to highlight specific events and jog a user's memory just from the plot of character interactions alone?

This essentially evaluates the potential of these timelines as tools for revision, effectively refreshing the users' mind as to the plot of the story illustrated, and as an aid in understanding: seeing from the timeline alone who's interacting with who and at what points significant events may have occurred.

7.4.2 Test Set-up

Again, this is a rather subjective matter and as such we make use of a user survey to gather some qualitative feedback. In the survey we asked users to view the timelines of three texts: *Goldilocks and the Three Bears*, *Little Red Riding Hood*, and our extract from *Harry Potter*.

In each case, we asked participants to respond to three questions:

- Is the timeline clear? (*Asking for a rating from 1 to 5, 5 being clear.*)
- Can you recognise any significant moments from this timeline?
- Does highlighting the paths of particular actors help you recognise any key moments you didn't before?

The exact timelines used in the survey are shown in Appendix C.4.

7.4.3 Results and Remarks

Goldilocks and the Three Bears

The timeline clarity was initially rated poorly, with confusion surrounding:

- What the numbers in each event mean (*where at this point the numbers reflected the number of characters involved in the event rather than the event number*)
- What actually happens in each event (*Participants understood the purpose of the survey, but would like a short summary or some other indicator of what happens in each of the events.*)
- In some cases, the somewhat "unpredictable" results of using a force-directed layout leads to slightly unnatural results (*i.e. The vertical ordering of the edges between events doesn't always remain the same, making it a little harder to follow.*)

However, having provided a brief explanation of the various aspects of the resulting visualisation the responses improved significantly, with users now able to more readily make insights from this timeline representation of the story. Figure 7.12 shows the results of these responses.

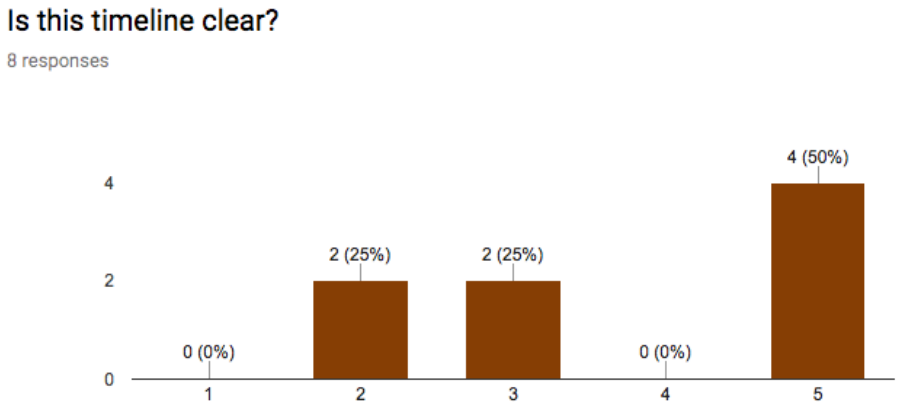


Figure 7.12: Responses to the clarity of the *Goldilocks* timeline. Participants rated the clarity from 1 (unclear) to 5 (clear).

In this example, the majority of respondents were indeed able to recognise a number of key events from this timeline, remembering *Goldilocks'* interactions with the *porridge*, *chairs*, and *beds*, and clearly recognising the ending when the *three bears* return. However, one particular response summarised the general consensus well:

> *"Difficult to count exactly what the characters are going through. As I remember there are three tests of each the porridge, the chairs, the beds, but I can't count that many number of events from Goldilocks. I don't understand what the grouping is based on, why are the three bears always together if they each realise individually that someone has touched their stuff? But you CAN see story progression, where Goldilocks discovers the home on her own, the moment the Bears come back and discover things by themselves, and finally, Goldilocks running away, which involves everyone."*

This is a particularly insightful response. Again, it is clear that the respondent can recognise roughly what is happening in the plot, but the plot fails to highlight:

- What's happening at each event (Highlight whether the event involves the *porridge*, *chair*, or *beds*), which would further aid memory recall.
- They perhaps would not have chosen this level of detail initially, instead preferring a finer level of detail with the reaction of each of the bears maintained as distinct events.

However, the respondent can clearly see the story progression and get a general insight into the story flow, which is our overall aim here.

Little Red Riding Hood

Figure 7.13 shows the clarity ratings for the timeline of this example. We again see a similar response with respect to plot clarity; marks are lost by the lack of any contextual information other than who's involved. The introduction of a short text summary over each event or some other meaningful annotation to each event would clearly greatly aid the users "at-a-glance" understanding of the text.

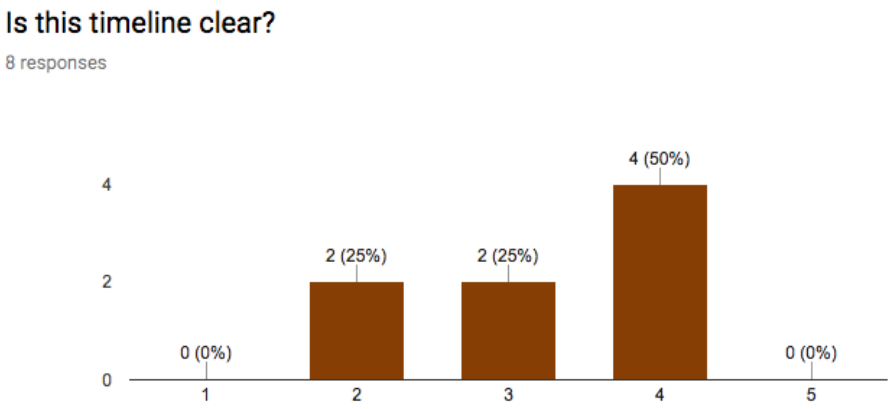


Figure 7.13: Responses to the clarity of the *Little Red Riding Hood* timeline. Participants rated the clarity from 1 (unclear) to 5 (clear).

Despite this, the majority of participants could quite clearly make out the distinct events in this fairy tale from the character interactions alone. Thus, the visualisation certainly appears to be achieving this aim. In this example, we also asked respondents whether they felt highlighting the paths of particular individuals in the timeline benefited the insight obtained. This received a mixed response, but did reinforce the purpose of this feature. Some subjects felt that highlighting only particular paths loses some context, however, others could also more clearly see the interactions between the characters highlighted, spotting divergences and convergences of the paths of these characters through the story. Thus, providing both the original timeline *in addition* to the ability to then highlight particular paths and interact with the plot certainly achieves both of these benefits.

Harry Potter

Lastly, we see a more mixed response to this example, although this was to some extent expected with the increase in the number of characters in the text making the resulting timeline somewhat more complex and sacrificing some clarity. The responses here shown in Figure 7.14.

Additionally, our results showed that gaining any real insight from this timeline alone relies upon some prior knowledge of the Harry Potter story, with most participants failing to recognise any particular occurrences. Despite this, one participant was able to recall a number of events from this text from the plot alone, and cited the following benefit of a timeline representation of this text:

> *" It's interesting to see how JKR introduces the components progressively and how they relate."*

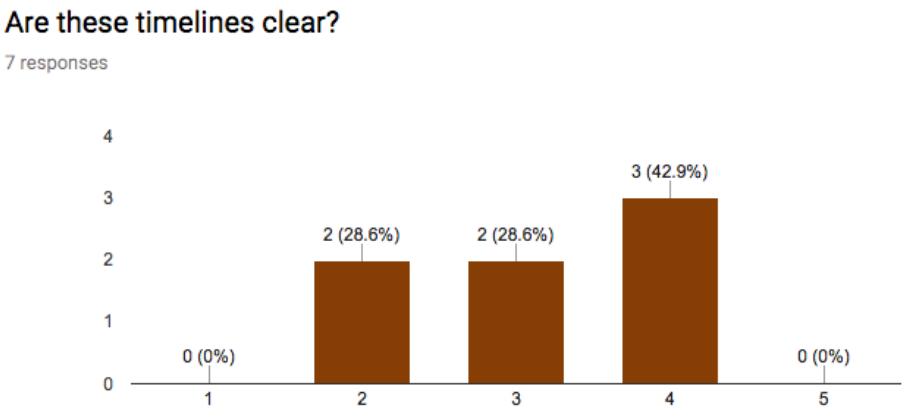


Figure 7.14: Responses to the clarity of the *Harry Potter* timeline. Participants rated the clarity from 1 (unclear) to 5 (clear).

It is thus clear, and perhaps expected under these test conditions, that this timeline representation is particularly valuable as a *revision aid*, more so than in obtaining an initial understanding of the story.

General Feedback

All respondents recognised the potential of such a visualisation in education, both in English Literature classes and in History classes, particularly if labels were empires or countries rather than people. In particular, test participants also cited the potential of such a visualisation as a revision aid and for potentially remembering what happens throughout the course of an entire series of books, in this case perhaps summarising each chapter as an event to provide a high-level overview of the text.

Scaling to such large texts is something that we are yet to explore, and would likely require optimisations of both the text annotation and clustering procedure as well as adapting the resulting visualisation to better handle a large number of actors.

Thus, it appears that our current timeline representation certainly shows potential, and meets the objective of clearly highlighting the interactions between characters within a text and providing a high level view of the overall plot of a narrative text. However, at present the current visualisation approach is certainly more effective at visualising the series of events in stories with a relatively small number of characters. Additionally, while a general overview of the story can be obtained, it is clear that the lack of any specific event information makes it difficult to pinpoint exact occurrences from the timeline alone.

7.5 Web Application Evaluation

7.5.1 Aims

We lastly focus on the *usability* of our overall application, taking the definition of usability specified by ISO 9241-11 as the [70]:

> *"Extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use."*

While the results of the clustering process and the final timeline that can be produced themselves are vitally important to our application, it is also paramount that our target user can actually achieve these results themselves in order to provide any real value. We thus focus on two aspects that summarise the definition of *usability* listed above: the *efficiency* with which a user can complete the typical actions required to obtain a final timeline, and the *ease-of-use* of the application, reflecting the ease with which the user can obtain these results.

7.5.2 Efficiency

To evaluate the efficiency with which each typical action can be completed, we list in Table 7.3 the number of clicks required to achieve each action.

In general, the number of clicks required to complete any function is kept minimal, with the last operation perhaps requiring a slightly excessive number of clicks: a potential opportunity for optimisation here by performing the re-drawing operation automatically immediately after updating the ordering, removing an additional click. We have also already optimised part of the initial actor tagging process when creating a new timeline by automatically selecting the gender of the actor if it is of the form: "*Mr. x*" or "*Mrs. x*".

Beyond these improvements, there is little room for further reduction in a large number of cases, however we shall see in the following evaluation of the *ease-of-use* whether we have managed to achieve our goal of providing efficient access to these functions without obscuring the user interface.

7.5.3 Ease of Use

To evaluate the *ease-of-use* of the application, we employ the System Usability Scale (SUS). This is a Likert scale that provides a general view of a system's usability, and has been shown to be relatively effective at distinguishing between systems that are usable and those that are deemed less-so [71].

Task	**Minimum Clicks Required**	**Comments**
Create an initial timeline	2	At a minimum the user must input their text and click the "Let's Go" button. Additionally, the user may wish to tag any automatically identified characters which typically requires 3 clicks per actor, or manually enter any names which again requires 3 clicks per actor. 1 click to add the new actor, and 2 clicks to select the gender.
Rename a character	3	Select the actor in the side-panel, change the name in the input box, and click save.
Tag a missing character in an event	3	Select the event to open the "edit-event" panel, select the mentions column of the sentence to update and type the new mention, and click save.
Change the selection of events to a level of detail of your liking	1	Simply drag the slider or select one of the event presets (extra-fine, fine, coarse, extra-coarse).
Manually group two or more events	2 + number of events to group	Click "Enter merge mode", select the events to group, and then click on the same button now labelled "Merge".
Undo a grouping of events you performed	1	Once a merge has been complete, the undo button becomes active next to it.
Delete a character/mention from the timeline	1	We can simply use the "x" next to the mention in the side panel to remove a mention completely.
Delete a character mention from a specific event.	3	This is much the same as adding an actor to a specific event, but we instead remove this character mention from the sentences it is mentioned in within the selected event.
Adjust the label ordering of characters in the timeline	4	Click the "Settings" button above the timeline, drag the actors in the ordering desired, and click "Save". We must then click the "Update Timeline" button to re-draw the timeline with the new changes.

Table 7.3: The number of clicks required to perform the typical actions necessary when constructing an accurate timeline using the application, with additional comments highlighting the steps involved.

For this evaluation we asked a small group of participants to complete the following 7 tasks after an initial introduction to the application, providing as little guidance as possible.

1. Create a timeline from a selected text.
2. Rename a character.
3. Tag “Bob” in one of the events.
4. Change the current set of events to those at a level of detail of your liking.
5. Merge two events into one in the current event set.
6. Update the timeline to reflect your changes.
7. Remove a character from the timeline.

Following this, each participant was then asked to respond to the following 10 statements defined by the SUS, rating each from 1 to 5 (Completely disagree to entirely agree).

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

The scores to each of these statements are then converted and combined to yield an overall *ease-of-use* score out of 100. More on the scoring process can be found in [72].

Table 7.4 shows our raw results yielded an average SUS score of 70.5, with individual scores ranging from 65 to 75. These scores put us slightly “above average”, which is considered as any score above 68, but falling short of the score of 80.3 recognised as exceptional [72]. One particular aspect highlighted by the responses received is that there is certainly some degree of learning curve with the application, which could be reduced either by some re-organisation of the user interface, or the incorporation of a user-guide to give a more insightful tutorial as to how to use the application to best achieve the desired results before the user first tries the system themselves. However, with a little guidance, users were soon able to complete the tasks requested and certainly saw some value in the application, particularly in the educational context.

Question	**Response (1 disagree - 5 agree)**				
1. I think that I would like to use this system frequently.	4	5	5	3	2
2. I found the system unnecessarily complex.	2	2	2	2	2
3. I thought the system was easy to use.	3.5	3	4	4	5
4. I think that I would need the support of a technical person to be able to use this system.	1	1	1	2	2
5. I found the various functions in this system were well integrated.	3.5	4	4	4	5
6. I thought there was too much inconsistency in this system.	2	3	2	1	2
7. I would imagine that most people would learn to use this system very quickly.	4	4	4	3	5
8. I found the system very cumbersome to use.	2	2	2	2	3
9. I felt very confident using the system.	3	3	3	3	3
10. I needed to learn a lot of things before I could get going with this system.	1	5	3	3	3
SUS Score	***75***	***65***	***75***	***67.5***	***70***

Table 7.4: Table of SUS responses. The values highlighted reflect the general need for some instruction before being able to fluently use the system, however all participants suggest that the learning process was relatively straight forward.

7.6 Evaluation Summary

Overall, the final application received a positive response from potential users and performed well when compared to the results expected by test participants. The novelty of this application makes it very difficult to compare with any existing tools, in particular our event identification approach which treats *events* in a far more general manner to existing event extraction tools, focussed more towards the task of information retrieval. In the following sections we highlight the key strengths and weaknesses of our final application.

7.6.1 Strengths

Obtaining a *natural* set of events

Our results show that our modified clustering method achieves the aim of identifying a *natural* set of events within a narrative text, consistently identifying the boundaries between distinct events to closely match those identified by test participants. This is particularly the case as we begin to group the text into increasingly large events, taking a higher-level view of overall story flow.

A clear and insightful timeline for short texts

The potential of this tool as a revision aid is clear. For short texts such as fairy tales, our force-directed timeline yields a clear and natural result, highlighting the interactions between characters in the story and also immediately revealing distinct sub-plots in the story. The result enables us to see how and where characters are involved in the story, and provides a higher-level view of a story than the corresponding text does alone. As a result, we see the particular potential of this tool in the primary school setting where teachers can use such a timeline as a teaching aid, helping students to understand the bigger picture in an easily digestible format and making it easy for the teacher to produce these timelines themselves.

Interaction to enable exploration

One of the novelties of our timeline is that it is also *interactive*. This enables further exploration of the events contained in the timeline and to guide investigation and insight into the underlying story. For example, as we saw in the example of Little Red Riding Hood, highlighting just the path of *Little Red Riding Hood* immediately revealed the *one* distinct event that did not involve her: this was in fact where her Grandma was eaten. Not only does this example illustrate the potential of this visual representation as an investigative tool, but also its ability to highlight the placement of such key events within a story and how the storyteller has intertwined sub-plots around these key events, again making this an extremely useful educational tool.

Consistent results regardless of the type of narrative text

Current training data for machine-learnt NLP tools consists largely of news and conversational speech data: not the typical type of input expected here. Our addition of a pre-processing stage to process any names that are unlikely to be recognised by these systems has lead to a more consistent experience regardless of whether the text contains characters named *Andrew*, *Goldilocks*, or *Mrs. Smith*. As a result, maximising the applicability of the application to the broadest range of narrative texts.

The application is quick to learn and easy to use

The resulting application is relatively intuitive, making it easy for anyone to use the tool and create a timeline for any piece of narrative text themselves. This unlocks the value of an underused and under-appreciated representation of information, significantly reducing the effort required to construct a timeline for a given text. While natural language processing tools may not be sufficiently accurate to allow the automation of the entire process of creating a timeline from text, the ease-of-use of our application makes it easy for the user to provide the extra 20% of work needed to obtain an accurate timeline for any narrative text.

Customisability

Finally, one of the key strengths of the application is that of customisability. While our ultimate objective was the *automated* creation of event timelines from text, we recognise the limitations of current tools and as such provide the means to fully customise the results produced if they're unsatisfactory. This ranges from relatively small tweaks such as renaming characters, to manually defining the entire set of events to visualise yourself by manually merging sentences into the event set desired. Thus, even if the *automated* results aren't what the user would hope for, they can still create the final timeline they want.

7.6.2 Weaknesses

Inaccuracies of coreference resolution

The nature of this application makes us heavily reliant on coreference resolution, which while improving are still at best achieving F_1 scores of 60% to 70%, and tend to perform worse than this under the context of narrative texts due to the different use of language to that used in the typical training data for such systems. Despite our attempts to improve this through pre-processing, there is still plenty of room for improvement here. As a result, the initial timeline may fail to correctly reflect the actual story flow at first, requiring the user to read the text themselves and make any necessary amendments.

Identifying event boundaries at the lowest level of detail

As we saw in Section 7.3, the intuitiveness of the events identified by our clustering method improves as we consider events at a coarser level of detail. Our current feature set does not appear to be elaborate enough to identify the far more subtle distinctions between events expressed at the lowest level of detail. For example, we currently fail to distinguish between *Goldilocks'* interactions with each of the *first* bowl of porridge, the *second* bowl of porridge, and the *last* bowl of porridge. Again, this is a distinction that could either be made through an improved coreference resolution system treating each of these as separate entities, or through the incorporation of additional features into the clustering process.

Handling a large number of characters

The results from our case studies show how the clarity of our current timeline visualisation degrades as the number of characters involved in a text increases beyond 6 or more, making it more difficult to recognise and follow the paths of particular characters through the text. As a result, the benefit of such a visualisation begins to diminish beyond this point. The incorporation of the ability to highlight only particular paths does help to combat this to some extent, but as acknowledged by test subjects, doing this sacrifices some of the contextual information to be gained from the timeline. However, this may be something that can be overcome through the use of an alternative node positioning strategy or illustration.

Lack of event semantics in the timeline

One of the most common responses to the timeline visualisation was *"What do the events actually mean?"*. While users can quite clearly see the high-level dynamics of interaction between characters in the plot, and if they know the text then they may well be able to identify certain specific events, in the majority of cases users then wanted to know exactly *what* was happening in each event. This is a piece of information the timeline currently fails to reflect. Our original inspiration of Figure 1.1 showing the Lord of the Rings story additionally highlighted some of the key details in the timeline, such as the location of a set of events or what happens at a particular point, such as the ring being destroyed. This is something that could certainly enhance the visualisation.

Semi-automatic creation of event timelines from text

As we mention above, while we have shown our event extraction process to produce a relatively natural set of events, and our timeline visualisation to provide useful insights into the underlying text, the process of getting from text to timeline still requires some additional input from the user. As such, we have not entirely reached our goal of completely automated construction of event timelines from text, but we have made it substantially easier for a user to generate a timeline from any input text. Our fundamental reliance on coreference resolution is unavoidable in this context, and the current inability to distinguish *character* coreferences from other objects mentioned means that we still require the user to provide the final 20% of effort to complete the timeline.

Chapter 8

Conclusions

8.1 Lessons Learnt

The broad scope of this project and sheer volume of information on natural language processing made this seem a rather ambitious project to begin with, and a great deal of time was spent experimenting with state-of-the-art tools and investigating related work in order to first identify our approach to this challenge. However, the potential value of this project made it worth while and I believe our results show that we're not far off delivering this value to real users. Along the way, we've learnt a great deal and wish to highlight some of these key lessons below.

It is clear that hierarchical clustering provides a good fit for event boundary detection under this context. The subjective nature of determining what we consider to be the *events* within a piece of text is evident, with boundaries changing based upon the level of detail the reader is considering or simply between different people that may use different criteria to collect and summarise a number of sentences into a single *event*. As a result, there is no single *right answer*, and instead hierarchical clustering provides the means to satisfy the majority of users. Additionally, different insights can be sought by considering *events* at different levels of granularity. A coarsely grained set of events, each summarising a large number of sentences may well give a good overview of the overall plot of a narrative, yet considering the series of far smaller events that take place may reveal more intricate sub-plots that are also present.

Our exploration of natural language processing and the current state-of-the-art has also shown the rapid progress that has been made in this field. However, it has also revealed the degree to which current tools are becoming tailored towards specific domains of language. This is something that needs to be addressed, by either developing tools based more generally on the fundamental rules of language to make them applicable to widest possible domain, or instead making such tools far easier to train over different datasets in order to adapt them to new domains. This would aid development in a far broader selection of domains. Of course, one of the major difficulties in this area is the development of sufficient training material to obtain accurate results using machine learning techniques. At present a large number of tools are tailored towards the domains of news and conversational speech due to the nature of existing development data sets. As a result, we have demonstrated the improvement obtained by pre-processing our narrative-domain input text to align more closely with

the typical language used in this training data. However, being able to instead *train* these tools to handle new domains of text may well have yielded further improvements.

Force-directed graphs also present real potential for creating predictable, emergent results. Rather than explicitly specifying the coordinate geometry of the resulting plot, we instead employ our more intuitive understanding of forces to encode the properties we'd like the result to exhibit and allow the laws of physics to translate this into our final layout. This is perhaps nothing new, however we do believe we've demonstrated how an explicit structure can be imparted on such graphs to obtain a balance between an *explicit* foundation with *emergent* properties. Resulting in natural and easily interpretable results that are not entirely stochastic, as is the typical nature of force-directed graphs.

Finally, while a number of challenges in natural language processing still remain to be overcome, the potential of current tools is quite clear. Despite it perhaps not being quite possible to fully automate tasks dominated by an understanding of natural language, we have certainly demonstrated that with a good enough platform around this technology we can still start to realise a great deal of value from these tools. In our case, users are quite happy to provide the finishing touches to create something that would have previously required a great deal more effort, particularly knowing that the system can only *improve* in future. The key is making it *easy* for the user to provide the final changes required to move from 80% accuracy to 100%.

8.2 Future Work

The wide scope of this project means that our work could be extended in a multitude of ways, and indeed one of our greatest challenges was in first identifying exactly what we wanted to achieve in this first iteration. We list below a number of extensions that we believe pose exciting opportunities for further development of this application:

- **Extending the feature set for hierarchical clustering** It is clear that hierarchical clustering is an appropriate method for event extraction, however our feature set could certainly be extended with a number of additional simple and complex features to improve the performance at the finer levels of granularity (in the early stages of clustering). These include the incorporation of recognising synonyms, tense, frame semantics, or other techniques such as semantic role labelling.

- **Enhance the timeline visualisation** User feedback clearly suggests that the incorporation of other information into the resulting infographic timeline could significantly improve user understanding. Thus, the incorporation of named entity recognition, semantic role labelling, or other techniques could provide the means to extract key pieces of information that could be reflected in the resulting timeline. For example, displaying key places, organisations, or items that are mentioned could help in understanding exactly what is happening in each of the events displayed.

- **Exploring other force-directed infographics** A lot of the real potential of force-directed graphs lies in data visualisation, where we impose no structure on the results and instead allow the force simulation itself show us the inherent structure of the data having encoded the various properties of interest to us as forces. The development of additional infographics using the information automatically extracted during our text processing pipeline could provide complementary insights to the more traditional timeline view, further guiding user investigation and exploration of a text.

- **Machine Learning to optimise feature weights** While our intuition and manual analysis may be able to guide us towards an approximate set of feature weights in order to yield the resulting event clusters we desire, this task will become more difficult as the feature set increases in size. Thus, the incorporation of machine learning could prove extremely valuable in optimising our feature weights to obtain the best results possible; training the system to produce results that most closely match the event sets expected by test participants at various points in the hierarchy.

- **Expanding the input domain** We could further expand our accepted domain of input text to additionally open up a number of new challenges, such as that of re-ordering the events within a text that does not exhibit the property of chronological text ordering that is so typical of narrative texts.

Bibliography

[1] Oxford Dictionary. Oxford Dictionary Definition of a Timeline.;. [Accessed 22nd May 2017]. Available from: https://en.oxforddictionaries.com/definition/timeline.

[2] xkcd. Lord of the Rings - Movie Narrative Chart;. [Accessed 5th June 2017]. Available from: https://xkcd.com/657/large/.

[3] Fillpot E. Teaching with Timelines;. [Accessed 6th January 2017]. Available from: http://teachinghistory.org/teaching-materials/teaching-guides/24347.

[4] Moline S. I see what you mean: children at work with visual information. Stenhouse Publishers; 1995. Cited in Hines, A. (2006) Using Timelines to Enhance Comprehension. http://www.colorincolorado.org/article/using-timelines-enhance-comprehension. Available from: https://books.google.co.uk/books?id=hQ5KAAAAYAAJ.

[5] Pustejovsky J, Castano JM, Ingria R, Sauri R, Gaizauskas RJ, Setzer A, et al. TimeML: Robust specification of event and temporal expressions in text. New directions in question answering. 2003;3:28–34.

[6] Mirza P. Extracting Temporal and Causal Relations between Events. ACL 2014. 2014;p. 10. Available from: http://www.aclweb.org/anthology/P14-3002.

[7] Jones KS. In: Zampolli A, Calzolari N, Palmer M, editors. Natural Language Processing: A Historical Review. Current Issues in Computational Linguistics: In Honour of Don Walker. Dordrecht: Springer Netherlands; 1994. p. 3–16. Available from: http://dx.doi.org/10.1007/978-0-585-35958-8_1.

[8] Manning CD, Raghavan P, Schütze H. 2.2.1. In: Introduction to Information Retrieval. 1st ed. Cambridge University Press; 2008. p. 22–26. Available from: http://amazon.com/o/ASIN/0521865719/;http://www-nlp.stanford.edu/IR-book/;.

[9] Palmer DD. 2. In: Tokenisation and sentence segmentation. Marcel Dekker, Inc., New York, USA; 2000. p. 11–17. Available from: https://books.google.co.uk/books?hl=en&lr=&id=VoOLvxyX0BUC&oi=fnd&pg=PA11&dq=Chapter+2:+Tokenisation+and+Sentence+Segmentation&ots=wugYLE2Ts1&sig=l95ZVLDfCKsbyc6qfADmsCvax1k#v=onepage&q=Chapter%202%3A%20Tokenisation%20and%20Sentence%20Segmentation&f=false.

[10] partofspeech org. Part of Speech Overview;. [Accessed 1st January 2017]. Available from: http://partofspeech.org/.

[11] Taylor A, Marcus M, Santorini B. In: The Penn Treebank: An Overview. Treebanks. Springer; 2003. p. 5–7.

[12] Grishman R, Sundheim B. Message Understanding Conference-6: A Brief History. In: COLING. vol. 96; 1996. p. 466–471. Available from: http://www.alta.asn.au/events/altss_w2003_proc/altss/courses/molla/C96-1079.pdf.

[13] Linguistic Data Consortium. LDC Past Projects: ACE;. [Accessed 4th January 2017]. Available from: https://www.ldc.upenn.edu/collaborations/past-projects/ace.

[14] Weischedel R, Palmer M, Marcus M, Hovy E, Pradhan S, Ramshaw L, et al. Ontonotes release 5.0 LDC2013T19. Linguistic Data Consortium, Philadelphia, PA. 2013;Available from: https://catalog.ldc.upenn.edu/docs/LDC2013T19/OntoNotes-Release-5.0.pdf.

[15] CoNLL. Language-Independent Named Entity Recognition (II); 2005. [Accessed 5th January 2017]. Available from: http://www.cnts.ua.ac.be/conll2003/ner/.

[16] Finkel JR, Grenager T, Manning C. Incorporating non-local information into information extraction systems by gibbs sampling. In: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics. Association for Computational Linguistics; 2005. p. 363–370.

[17] Ratinov L, Roth D. Design challenges and misconceptions in named entity recognition. In: Proceedings of the Thirteenth Conference on Computational Natural Language Learning. Association for Computational Linguistics; 2009. p. 147–155.

[18] Tjong Kim Sang EF, Meulder FD. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In: Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4. Association for Computational Linguistics; 2003. p. 142–147.

[19] Redman T, Sammons M, Roth D. Illinois Named Entity Recognizer: Addendum to Ratinov and Roth '09 reporting improved results. 2016;Available from: http://cogcomp.cs.illinois.edu/papers/ner-addendum-2016.pdf,.

[20] The Stanford Natural Language Processing Group. Coreference Resolution;. [Accessed 6th January 2017]. Available from: http://nlp.stanford.edu/projects/coref.shtml.

[21] Pradhan S, Moschitti A, Xue N. CoNLL 2012 Task: Modelling Multilingual Unrestricted Coreference in OntoNotes;. Available from: http://conll.cemantix.org/2012/introduction.html.

[22] Pradhan S, Moschitti A, Xue N. CoNLL 2012 Task: Modelling Multilingual Unrestricted Coreference in OntoNotes - Task Description;. Available from: http://conll.cemantix.org/2012/task-description.html.

[23] Zitouni I. Natural Language Processing of Semitic Languages. Springer Berlin Heidelberg; 2014. Available from: https://books.google.co.uk/books?id=5ZS4BAAAQBAJ.

[24] Clark K, Manning CD. Entity-Centric Coreference Resolution with Model Stacking. In: Association of Computational Linguistics (ACL); 2015. .

[25] Chang KW, Samdani R, Roth D. A constrained latent variable model for coreference resolution. 2013;.

[26] Ng V, Cardie C. Improving Machine Learning Approaches to Coreference Resolution. In: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics. ACL '02. Stroudsburg, PA, USA: Association for Computational Linguistics; 2002. p. 104–111. Cited in Kai-Wei Chang and Rajhans Samdani and Dan Roth. A Constrained Latent Variable Model for Coreference Resolution; 2013. Available from: http://dx.doi.org/10.3115/1073083.1073102.

[27] Bengtson E, Roth D. Understanding the Value of Features for Coreference Resolution. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing. EMNLP '08. Stroudsburg, PA, USA: Association for Computational Linguistics; 2008. p. 294–303. Cited in Kai-Wei Chang and Rajhans Samdani and Dan Roth. A constrained latent variable model for coreference resolution; 2013. Available from: http://dl.acm.org/citation.cfm?id=1613715.1613756.

[28] Stanford NLP Group. Stanford Core NLP: CorefAnnotator;. [Accessed 18th January 2017]. Available from: http://stanfordnlp.github.io/CoreNLP/coref.html.

[29] Lee H, Peirsman Y, Chang A, Chambers N, Surdeanu M, Jurafsky D. Stanford's Multi-pass Sieve Coreference Resolution System at the CoNLL-2011 Shared Task. In: Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task. CONLL Shared Task '11. Stroudsburg, PA, USA: Association for Computational Linguistics; 2011. p. 28–34. Available from: http://dl.acm.org/citation.cfm?id=2132936.2132938.

[30] Clark K, Manning CD. Improving Coreference Resolution by Learning Entity-Level Distributed Representations. CoRR. 2016;abs/1606.01323. Available from: http://arxiv.org/abs/1606.01323.

[31] Derczynski L. Automatically Ordering Events and Times in Text. Studies in Computational IntelligenceSpringer. 2015;.

[32] Campos R, Dias G, Jorge AM, Jatowt A. Survey of temporal information retrieval and related applications. ACM Computing Surveys (CSUR). 2015;47(2):15. Available

from: http://dl.acm.org.iclibezp1.cc.ic.ac.uk/citation.cfm?doid=2658850.2619088;.

[33] Allen J, Derczynski L, Llorens H, Pustejovsky J, UzZaman N, Verhagen M. TempEval-3 Temporal Annotation - Task Description;. Available from: https://www.cs.york.ac.uk/semeval-2013/task1/.

[34] Strötgen J, Gertz M. Multilingual and cross-domain temporal tagging. Language Resources and Evaluation. 2013;47(2):269–298.

[35] Chang AX, Manning CD. SUTime: A library for recognizing and normalizing time expressions. In: LREC; 2012. p. 3735–3740. Available from: http://www-nlp.stanford.edu/pubs/lrec2012-sutime.pdf.

[36] Stroetgen J. HeidelTime GitHub page;. Available from: https://github.com/HeidelTime/heideltime.

[37] Kuzey E, Setty V, Strötgen J, Weikum G. As Time Goes By: comprehensive tagging of textual phrases with temporal scopes. In: Proceedings of the 25th International Conference on World Wide Web. International World Wide Web Conferences Steering Committee; 2016. p. 915–925.

[38] Kuzey E, Strötgen J, Setty V, Weikum G. Temponym Tagging: Temporal Scopes for Textual Phrases. In: Proceedings of the 25th International Conference Companion on World Wide Web. International World Wide Web Conferences Steering Committee; 2016. p. 841–842.

[39] Zhao R, Do QX, Roth D. A robust shallow temporal reasoning system. In: Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Demonstration Session. Association for Computational Linguistics; 2012. p. 29–32.

[40] Jurafsky D, Martin JH. In: Semantic Role Labeling. vol. 3 of Speech and Language Processing (3rd ed. draft). 3rd ed.;. Available from: https://web.stanford.edu/~jurafsky/slp3/.

[41] Punyakanok V, Roth D, Yih W. The Importance of Syntactic Parsing and Inference in Semantic Role Labeling. Computational Linguistics. 2008;34(2). Available from: http://cogcomp.cs.illinois.edu/papers/PunyakanokRoYi07.pdf;https://cogcomp.cs.illinois.edu/page/publication_view/183;.

[42] Sangeetha S, Thakur R, Arock M. Domain Independent Event Extraction System Using Text Meaning Representation Adopted for Semantic Web. International Journal of Computer Information Systems and Industrial Management Applications (IJCISIM) ISSN. 2010;p. 2150–7988.

[43] TempEval-2 Task;. [Accessed 18th January 2017]. Available from: http://www.timeml.org/tempeval2/.

[44] The Stanford Natural Language Processing Group. Stanford OpenIE;. [Accessed 18th January 2017]. Available from: http://nlp.stanford.edu/software/openie.html.

[45] Angeli G, Premkumar MJ, Manning CD. Leveraging linguistic structure for open domain information extraction. In: Proceedings of the Association of Computational Linguistics (ACL), 2015.. vol. 1. Department of Computer Science, Stanford University, United States. Association for Computational Linguistics (ACL); 2015. p. 344–354.

[46] Hiong SN, Kulathuramaiyer N, Labadin J. NATURAL LANGUAGE SEMANTIC EVENT EXTRACTION PIPELINE;Available from: http://www.icoci.cms.net.my/proceedings/2013/PDF/PID63.pdf.

[47] Llorens H, Saquete E, Navarro B. TIPSem (English and Spanish): Evaluating CRFs and Semantic Roles in TempEval-2. In: Proceedings of the 5th International Workshop on Semantic Evaluation. SemEval '10. Stroudsburg, PA, USA: Association for Computational Linguistics; 2010. p. 284–291. Available from: http://dl.acm.org/citation.cfm?id=1859664.1859727.

[48] Chambers N, Jurafsky D. Unsupervised Learning of Narrative Event Chains. In: ACL. vol. 94305. Citeseer; 2008. p. 789–797. Available from: http://nlp.stanford.edu/pubs/narrative-schema09.pdf.

[49] Liu P, Pan X. Text summarization with TensorFlow, Google Research Blog;. Available from: https://research.googleblog.com/2016/08/text-summarization-with-tensorflow.html.

[50] Vassilvitskii S, Arthur D. On the Worst-Case Complexity of the k-Means Method;. Available from: http://theory.stanford.edu/~sergei/slides/kMeans-hour.pdf.

[51] Ben-Hur A, Horn D, Siegelmann HT, Vapnik V. Support vector clustering. Journal of machine learning research. 2001;2(Dec):125–137.

[52] Greenacre M. 7. In: Hierarchical Cluster Analysis;. Available from: http://84.89.132.1/~michael/stanford/maeb7.pdf.

[53] Sahoo N, Callan J, Krishnan R, Duncan G, Padman R. Incremental Hierarchical Clustering of Text Documents. In: Proceedings of the 15th ACM International Conference on Information and Knowledge Management. CIKM '06. New York, NY, USA: ACM; 2006. p. 357–366. Available from: http://doi.acm.org/10.1145/1183614.1183667.

[54] Manning CD, Raghavan P, Schütze H, et al. 7. In: Hierarchical Clustering. vol. 1 of Introduction to information retrieval. Cambridge university press Cambridge; 2008. p. 377. Available from: https://nlp.stanford.edu/IR-book/pdf/17hier.pdf.

[55] Gansner E, Koutsofios E, North S. Drawing graphs with dot; 2006. [Accessed December 20th 2016]. Available from: http://www.graphviz.org/Documentation/dotguide.pdf.

[56] Massachusetts Institute of Technology and Contributors 2006-2009. MIT SIMILE, Web Widget for Visualizing Temporal Data;. Available from: http://simile-widgets.org/timeline/.

[57] Preceden;. [Accessed 4th January 2017]. Available from: https://www.preceden.com.

[58] McGinn D, Birch D, Akroyd D, Molina-Solana M, Guo Y, Knottenbelt WJ. Visualizing Dynamic Bitcoin Transaction Patterns. Big Data. 2016;4(2):109–119.

[59] Kobourov SG. Force-directed drawing algorithms. Handbook of Graph Drawing and Visualization. 2013;p. 383–408.

[60] Cui L. Extracting and Visualising Event Timelines from Text; 2016. Imperial College London - Computing MSc Thesis.

[61] Chasin R. Event and Temporal Information Extraction Towards Timelines of Wikipedia Articles. Simile. 2010;p. 1–9. Available from: http://cs.uccs.edu/~jkalita/work/reu/REUFinalPapers2010/Chasin.pdf.

[62] Etiene T, Pagliosa P, Nonato LG. Linea: Tailoring timelines by visual exploration of temporal text. In: Visual Analytics Science and Technology (VAST), 2014 IEEE Conference on. IEEE; 2014. p. 247–248. Available from: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7042513.

[63] Do QNT, Bethard S, Moens MF. Visualizing the Content of a Children's Story in a Virtual World: Lessons Learned; 2016. Available from: http://www.aclweb.org/anthology/W16-6009.

[64] Bostock M. D3 3.x API Reference - Force Layout;. Available from: https://github.com/d3/d3-3.x-api-reference/blob/master/Force-Layout.md.

[65] Node js Foundation. Node.js Home Page;. [Accessed 3rd June 2017]. Available from: https://nodejs.org/en/.

[66] University of Illinois Cognitive Computation Group. CogComp NLP Pipeline;. [Accessed 6th June 2017]. Available from: https://github.com/CogComp/cogcomp-nlp/tree/master/pipeline.

[67] Stanford University. Stanford CoreNLP Github Page;. [Accessed 6th June 2017]. Available from: https://stanfordnlp.github.io/CoreNLP/memory-time.html#depparse.

[68] Rowling JK. In: Chapter One. 1st ed. Harry Potter and the Philosopher's Stone. Bloomsbury Publishing PLC (26 Jun. 1997); 1997. p. 1–3.

[69] Manning CD, Raghavan P, Schütze H, et al.. Stanford IR-Book: Evaluation of Clustering;. [Accessed 4th June 2017]. Available from: https://nlp.stanford.edu/IR-book/html/htmledition/evaluation-of-clustering-1.html.

[70] International Standards Organisation. Guidance on usability - ISO 9241-11;. [Accessed 5th June 2017]. Available from: https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:ed-1:v1:en.

[71] System Usability Scale (SUS);. [Accessed 5th June 2017]. Available from: https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html.

[72] How To Use The System Usability Scale (SUS) To Evaluate The Usability Of Your Website;. [Accessed 5th June 2017]. Available from: http://usabilitygeek.com/how-to-use-the-system-usability-scale-sus-to-evaluate-the-usability-of-your-website/.

[73] Baxter N. In: The Gingerbread Man. My Little Treasury of Stories & Rhymes. Bookmark Limited; 1997. p. 102. Available from: https://www.amazon.co.uk/My-Little-Treasury-Stories-Rhymes/dp/1843229048.

Appendix A

Example Texts

Below we list the exact texts we used for our development and testing that we refer to repeatedly throughout the report.

A.1 Goldilocks and the Three Bears

To adhere to the copyright notice of the source, we refer the reader to `http://www.dltk-teach.com/rhymes/goldilocks_story.htm`[1] to find the Goldilocks text we used for development.

A.2 Little Red Riding Hood

Similarly, to the above, we refer the reader to `http://www.dltk-teach.com/rhymes/littlered/story.htm`[2] to find the Little Red Riding Hood text we used for development. Additionally, we adapted this text by replacing all synonyms of *mother* with *mommy* and all synonyms of *Grandmother* with *Grandma*.

A.3 The Gingerbread Man

For the exact text we use, I refer the reader to [73].

A.4 Harry Potter and the Philosopher's Stone

The following text is the short extract taken from the beginning of [68] to evaluate the performance of our application over a more sophisticated text.

> Mr. and Mrs. Dursley, of number four, Privet Drive, were proud to say that they were perfectly normal, thank you very much. They were the last people you'd expect to be involved in anything strange or mysterious, because they just didn't hold with such nonsense. Mr. Dursley was the director of a firm called Grunnings, which made drills.

[1] Accessed 8th June 2017.
[2] Accessed 8th June 2017.

He was a big, beefy man with hardly any neck, although he did have a very large mustache. Mrs. Dursley was thin and blonde and had nearly twice the usual amount of neck, which came in very useful as she spent so much of her time craning over garden fences, spying on the neighbors. The Dursleys had a small son called Dudley and in their opinion there was no finer boy anywhere. The Dursleys had everything they wanted, but they also had a secret, and their greatest fear was that somebody would discover it. They didn't think they could bear it if anyone found out about the Potters. Mrs. Potter was Mrs. Dursley's sister, but they hadn't met for several years; in fact, Mrs. Dursley pretended she didn't have a sister, because her sister and her good-for-nothing husband were as unDursleyish as it was possible to be. The Dursleys shuddered to think what the neighbors would say if the Potters arrived in the street. The Dursleys knew that the Potters had a small son, too, but they had never even seen him. This boy was another good reason for keeping the Potters away; they didn't want Dudley mixing with a child like that. When Mr. and Mrs. Dursley woke up on the dull, gray Tuesday our story starts, there was nothing about the cloudy sky outside to suggest that strange and mysterious things would soon be happening all over the country. Mr. Dursley hummed as he picked out his most boring tie for work, and Mrs. Dursley gossiped away happily as she wrestled a screaming Dudley into his high chair. None of them noticed a large, tawny owl flutter past the window. At half past eight, Mr. Dursley picked up his briefcase, pecked Mrs. Dursley on the cheek, and tried to kiss Dudley good-bye but missed, because Dudley was now having a tantrum and throwing his cereal at the walls. "Little tyke," chortled Mr. Dursley as he left the house. He got into his car and backed out of number four's drive. It was on the corner of the street that he noticed the first sign of something peculiar – a cat reading a map. For a second, Mr. Dursley didn't realize what he had seen – then he jerked his head around to look again. There was a tabby cat standing on the corner of Privet Drive, but there wasn't a map in sight. What could he have been thinking of? It must have been a trick of the light. Mr. Dursley blinked and stared at the cat. It stared back. As Mr. Dursley drove around the corner and up the road, he watched the cat in his mirror. It was now reading the sign that said Privet Drive – no, looking at the sign; cats couldn't read maps or signs. Mr. Dursley gave himself a little shake and put the cat out of his mind. As he drove toward town he thought of nothing except a large order of drills he was hoping to get that day.

Appendix B

Additional Results

B.1 Additional Coreference Resolution Experimentation Results

Table B.1 shows a comparison of results between the neural coreference resolution system using two different levels of *greediness*. The table shows the most significant differences, with the highlighted cells revealing the most significant differences between the two configurations. In particular in event 6, where the wolf was never correctly identified by any of the alternative Stanford coreference implementations.

B.2 Manually annotated Goldilocks event clusters

Figure B.1 shows the remaining manually annotated text for the Goldilocks example where we highlighted the features we intended to extract and our desired event boundaries. The first half is shown in Figure 4.1.

No.	Sentence	Mentions (Neural) - 0.45	Mentions (Neural) - 0.55	Comments
1	Remember, go straight to Grandma's house, her mommy cautioned.	her mommy, her Grandma	her mommy, the poor Grandma	
2	But when Little Red Riding Hood noticed some lovely flowers in the woods, she forgot her promise to her mommy.	her mommy, Little Red Riding Hood as she, The woods	her mommy, Little Red Riding Hood as she, The woods	
3	I'm on my way to see my Grandma who lives through the forest, near the brook, Little Red Riding Hood replied.	The wolf, Little Red Riding Hood as she, the forest	Little Red Riding Hood as she, a dark shadow approaching out of the forest behind her, the forest	Both appear to incorrectly detect a mention of the wolf.
4	Poor Grandma did not have time to say another word, before the wolf gobbled her up!	The wolf, Poor Grandma	a dark shadow approaching out of the forest behind her, the poor Grandma	
5	It's me, Little Red Riding Hood.	The wolf, Little Red Riding Hood as she	Little Red Riding Hood as she, a dark shadow approaching out of the forest behind her	Both incorrectly include the wolf.
6	Oh, I just have touch of a cold, squeaked the wolf adding a cough at the end to prove the point.		a dark shadow approaching out of the forest behind her	The neural (0.55) finally recognises the mention of the wolf here, which no other implementation recognised.
7	But Grandma! What big eyes you have, said Little Red Riding Hood.	Little Red Riding Hood as she	Little Red Riding Hood as she, a dark shadow approaching out of the forest behind her	Neural (0.55) incorrectly includes the wolf.
8	She ran across the room and through the door, shouting, "Help! Wolf!" as loudly as she could.	the door, a little girl who lived in a village near the forest	the door, Wolf, a little girl who lived in a village near the forest	The neural (0.55) correctly recognises the mention of the wolf here.
9	Little Red Riding Hood and her Grandma had a nice lunch and a long chat.	Little Red Riding Hood as she, her Grandma	Little Red Riding Hood as she, the poor Grandma	

Table B.1: Comparison of coreference annotations between the Stanford neural system with *greediness* values of 0.45 and 0.55.

Figure B.1: Initial analysis of the Goldilocks story, showing the second half of the story. **Yellow** highlighting reflects common features between nearby sentences, **green** highlighting reflects other features that could potentially be used as features, and **blue** highlighting reflects another possible overlap between sentences. The horizontal lines are the suggested event boundaries based upon this manual analysis.

B.3 Automatic Event Cluster Selection Analysis

The tables below show the raw data we analysed in order to identify whether there is a distinct pattern in the cluster hierarchies produced that can be used to automatically recognise good cluster sets at different levels of granularity.

Time step	**Score**	**% of max score**	**Distance between merged clusters**
T = 0	0.632	100	
T = 13 (3 merges with same score)	0.133333333	21.09704641	1
T = 16	0.08912896	14.10268354	6
T = 17 (optimal fine-grained clustering)	0.079579429	12.59168174	5
T = 18	0.0768	12.15189873	3
T = 19	0.074285714	11.75406872	1
T = 21	0.064	10.12658228	2
T = 22	0.063715556	10.08157525	5
T = 23 (optimal coarse-grained clustering)	0.06144	9.721518987	5
T = 24	0.035108571	5.555153707	5
T = 25	0.032622364	5.161766526	8

Table B.2: Analysis of the scores contributing to event merges in the *Goldilocks* text at each step of the hierarchical clustering process. The *score* column represents the similarity score between the sentences in the text that cause the merge at that time step. We highlight the rows that we regard as *optimal* event sets at two levels of granularity.

Time step	**Score**	**% of initial score**	**Distance between merged clusters**
T = 0	0.768	100	
T = 13	0.0903529	11.76	3
T = 14	0.076	9.89	1
T = 17	0.064	8.33	3
T = 18	0.0608	7.92	3
T = 19 (2 merges with same score)	0.06	7.8125	2
T = 21 (optimal fine grained clustering)	0.049152	6.4	5
T = 22	0.038912	5.066666667	5
T = 23	0.032622364	4.247703704	8
T = 24	0.029257143	3.80952381	4
T = 25	0.021370435	2.782608696	5
T = 26 (optimal coarse-grained clustering)	0.017096348	2.226086957	6
T = 27	0.012201612	1.588751515	8
T = 28	0.003979185	0.518123039	14

Table B.3: Analysis of the scores contributing to event merges in the *Little Red Riding Hood* text at each step of the hierarchical clustering process. The *score* column represents the similarity score between the sentences in the text that cause the merge at that time step. We highlight the rows that we regard as *optimal* event sets at two levels of granularity.

Time step	**Score**	**% of initial score**	**Distance between merged clusters**
T = 0	0.48	100	
T = 11	0.138971429	28.95238095	3
T = 12	0.121904762	25.3968254	1
T = 13	0.118857143	24.76190476	2
T = 14	0.108571429	22.61904762	1
T = 15 (optimal fine-grained clustering)	0.076	15.83333333	2
T = 16	0.0608	12.66666667	2
T = 17	0.052869565	11.01449275	2
T = 18	0.035576686	7.411809524	6
T = 19 (optimal coarse-grained clustering)	0.027670756	5.764740741	6
T = 20	0.02125114	4.427320889	8
T = 21	0.005627888	1.172476704	11

Table B.4: Analysis of the scores contributing to event merges in the *Gingerbread Man* text at each step of the hierarchical clustering process. The *score* column represents the similarity score between the sentences in the text that cause the merge at that time step. We highlight the rows that we regard as *optimal* event sets at two levels of granularity.

Time step	**Score**	**% of initial score**	**Distance between merged clusters**
T = 0	0.2432	100	
T = 8 (2 merges)	0.116923077	48.07692308	1
T = 10	0.077824	32	5
T = 11	0.071529412	29.41176471	2
T = 12 (optimal fine-grained clustering)	0.058461538	24.03846154	1
T = 13	0.046769231	19.23076923	2
T = 14	0.036623059	15.05882353	5
T = 15	0.028299636	11.63636364	5
T = 16 (optimal coarse-grained clustering)	0.021033514	8.648648649	4
T = 17	0.019922944	8.192	8
T = 18	0.012170971	5.004511278	6

Table B.5: Analysis of the scores contributing to event merges in the *Harry Potter* text at each step of the hierarchical clustering process. The *score* column represents the similarity score between the sentences in the text that cause the merge at that time step. We highlight the rows that we regard as *optimal* event sets at two levels of granularity.

Time step	Score	% of initial score
T = 0	0.509090909	100
T = 34	0.066086957	12.98136646
T = 35	0.064	12.57142857
T = 36	0.063333333	12.44047619
T = 37	0.058461538	11.48351648
T = 38	0.0512	10.05714286
T = 39	0.045279418	8.894171429
T = 40	0.044218182	8.685714286
T = 41	0.043428571	8.530612245
T = 42	0.038313354	7.525837363
T = 43	0.032768	6.436571429
T = 44	0.031380645	6.1640553
T = 45	0.027733333	5.447619048
T = 46	0.023058963	4.529439153
T = 47	0.022769079	4.472497633

Table B.6: Analysis of the scores contributing to event merges in a longer snippet from the *Harry Potter* text at each step of the hierarchical clustering process. The *score* column represents the similarity score between the sentences in the text that cause the merge at that time step. We highlight the rows that we regard as *optimal* event sets at different levels of granularity. In this case, we see that even in large texts we see good cluster sets emerge at similar score thresholds.

Appendix C

Additional Material

C.1 Raw Results of Coreference Resolution Experimentation

The raw results of coreference annotation for the example of Little Red Riding Hood can be found online at `https://docs.google.com/spreadsheets/d/1IZLB-h1c_a_mxZEbBOmhB_2fHExe5u7otwrt8x6m4bk/edit?usp=sharing`. Additionally, the raw results for the processed Little Red Riding Hood text featuring name replacements can be found at `https://docs.google.com/spreadsheets/d/1AIIczkS-JhM3ZSmKf1XN_BJTwRHlGK2Tc0ZsDsYmeU8/edit?usp=sharing`.

In these tables, **TP**, **FP**, and **FN** represent the number of *true positives*, *false positives*, and *false positives*, respectively.

C.2 Gingerbread Man Events

In Figure C.1 we provide the accompanying text for the *Gingerbread Man* example timeline shown in Figure 5.1.

C.3 Early Timeline Visualisations

Figures C.2 and C.3 below show our early example timelines produced using the GraphViz tool. However, the lack of dynamism and clarity expressed in these plots lead us to explore alternative approaches.

C.4 Timelines used in Evaluation

Figures C.4 to C.9 below show the timelines presented to test participants in evaluating the clarity and expressiveness of our resulting timelines. Note, when these plots were created the numbers within each event reflected the number of characters involved in that event. However, we have since redefined this to instead reflect the particular event number.

#	Event text	Actors mentioned
1	Once upon a time a little old man and little old woman lived in the country. One day, the little old woman made some ginger cookies.	the little old woman
2	She had some dough left over, so she made a little gingerbread man, with three buttons and two eyes made of raisins and a smiley mouth made out of a cherry.	the little old woman; a little gingerbread man
3	But when the little old woman went to take him out of the oven, the gingerbread man jumped from the tray and ran right out of the door! "Come back!" shouted the little old woman, running after him. "Come back!" shouted the little old man, who was working in the garden.	the little old woman; him; a little gingerbread man
4	But the gingerbread man called over his shoulder, "Run, run, as fast as you can! You won't catch me, I'm the gingerbread man!".	the gingerbread man called over his shoulder , `` Run , run , as fast as you can; a little gingerbread man
5	He ran down the garden path and out onto the road. As he ran, he passed a cow in a field. "Stop!" Mooed the cow. "You look good to eat!". And she ran after him.	the gingerbread man called over his shoulder , `` Run , run , as fast as you can; a cow
6	But the gingerbread man didn't stop for a single second.	a little gingerbread man
7	"A little old woman and a little old man couldn't catch me and neither will you! Run, run, as fast as you can! You won't catch me, I'm the gingerbread man!".	me; a little gingerbread man
8	In the next field he passed a horse. "Stop!" neighed the horse. "You look good to eat!".	a horse; a little gingerbread man
9	But the gingerbread man kept running. It was the same when he passed a rooster on a gate and a pig in a yard.	a little gingerbread man
10	The gingerbread man did not stop for anything...until he came to a fast-flowing river.	a fast-flowing river; a little gingerbread man
11	"I can't swim," cried the gingerbread man. "What can I do?". "Can I help you?" asked a quiet voice. It was a big red fox. "Jump on my back and I will carry you across". The gingerbread man did as the fox said, and the fox swam into the river. But soon the fox spoke again.	a little gingerbread man; a big red fox; a fast-flowing river
12	"The water is deeper here. Climb onto my head and you will stay dry." The little man did so. "The water is even deeper deeper now," said the fox soon. "Climb onto my nose and you'll stay dry."	The water; a little gingerbread man; a big red fox
13	As soon as the gingerbread man did so, the fox tossed him into the air. The little man fell right into the fox's open mouth.	a big red fox; a little gingerbread man
14	When the little old man and woman and came puffing along, only a few crumbs were floating in the water.	The water

Figure C.1: The set of events produced by our application for the timeline shown in Figure 5.1.

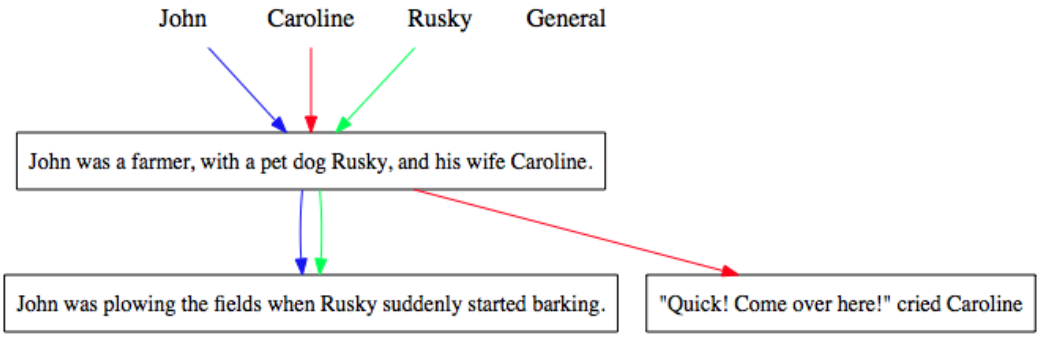


Figure C.2: One of our early timelines created with the GraphViz tool.

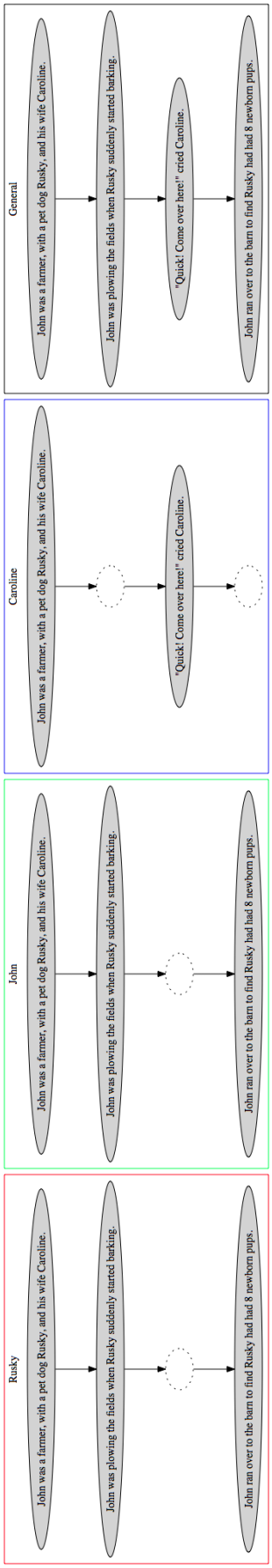


Figure C.3: Our initial *threaded* timeline with separate tracks for each character mentioned in the story.

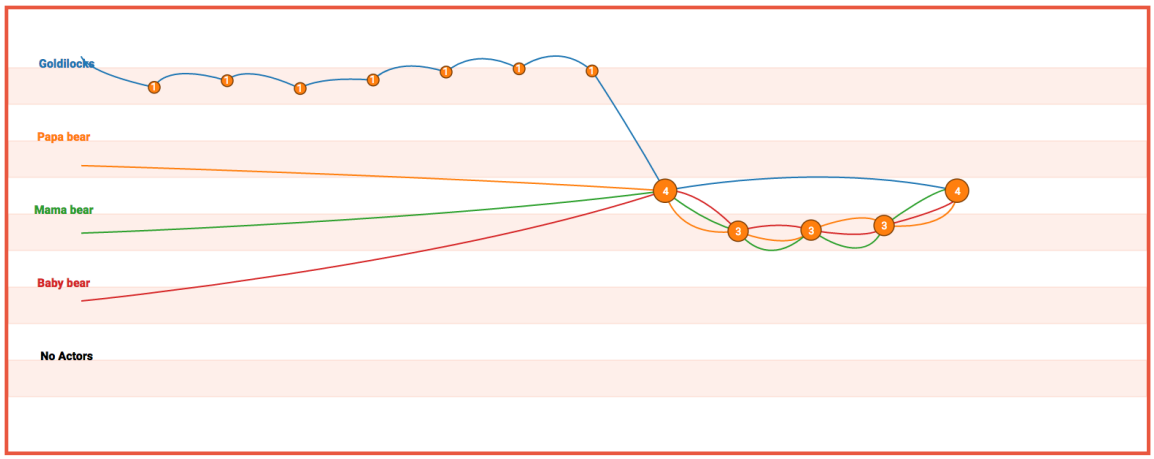


Figure C.4: Goldilocks timeline.

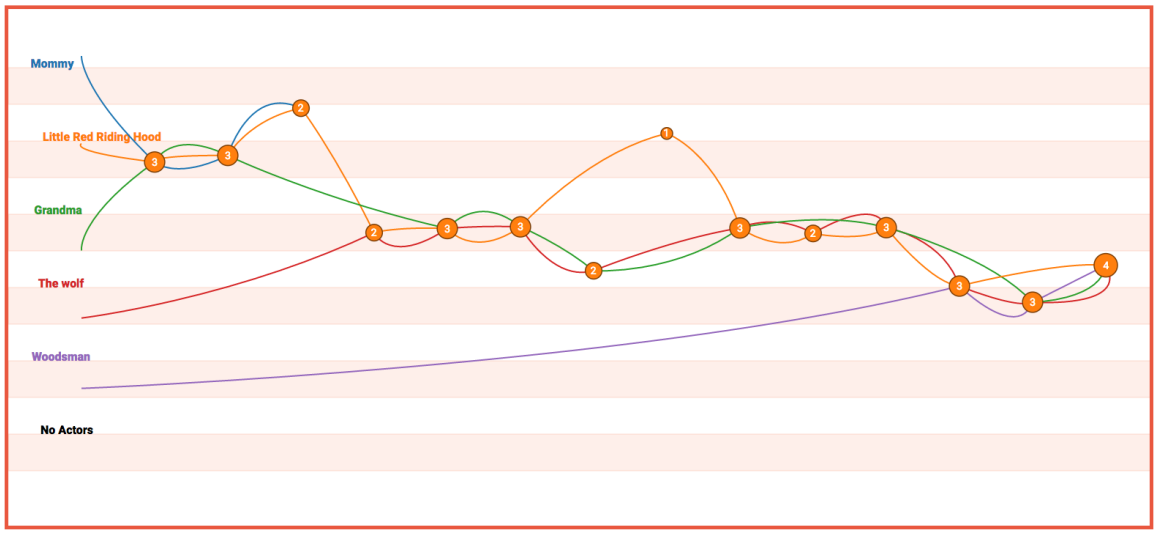


Figure C.5: Little Red Riding Hood timeline.

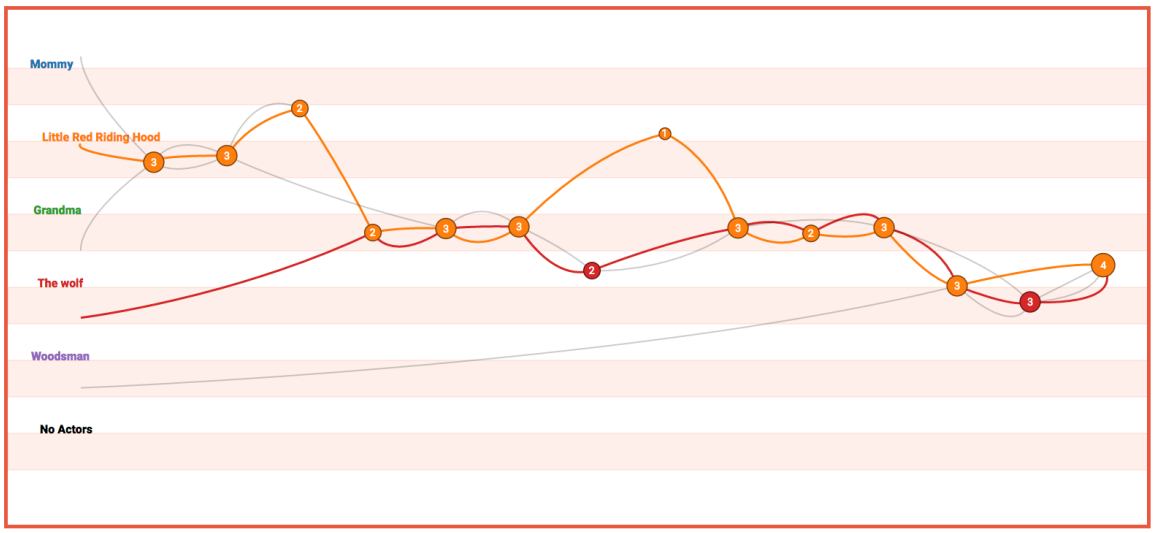


Figure C.6: Little Red Riding Hood timeline with *Little Red Riding Hood* and *The wolf* specifically highlighted.

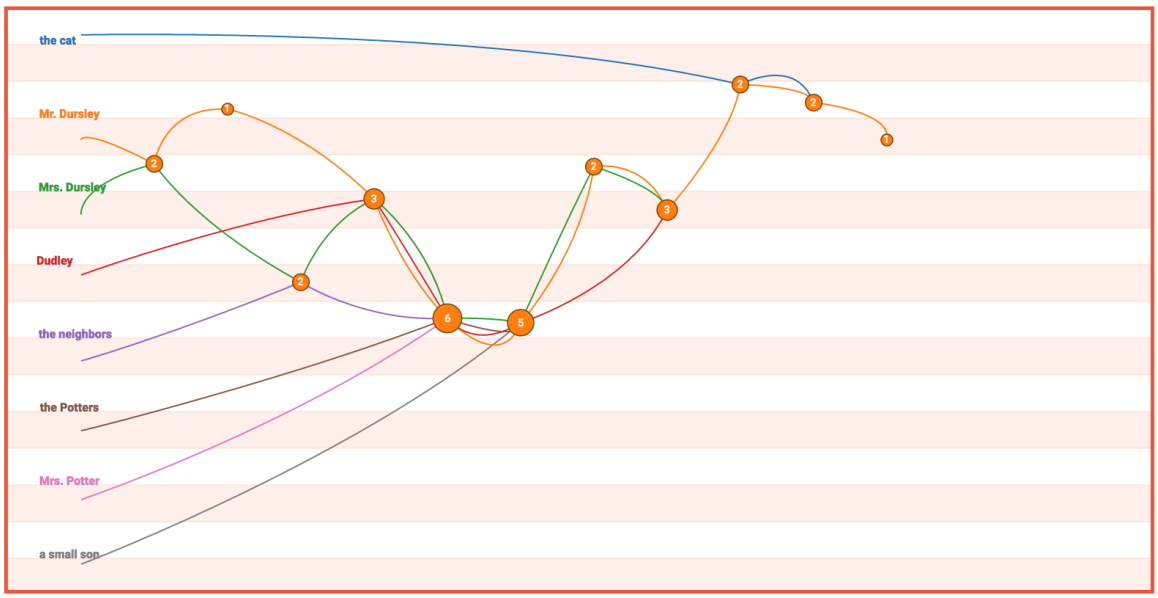


Figure C.7: Harry Potter timeline.

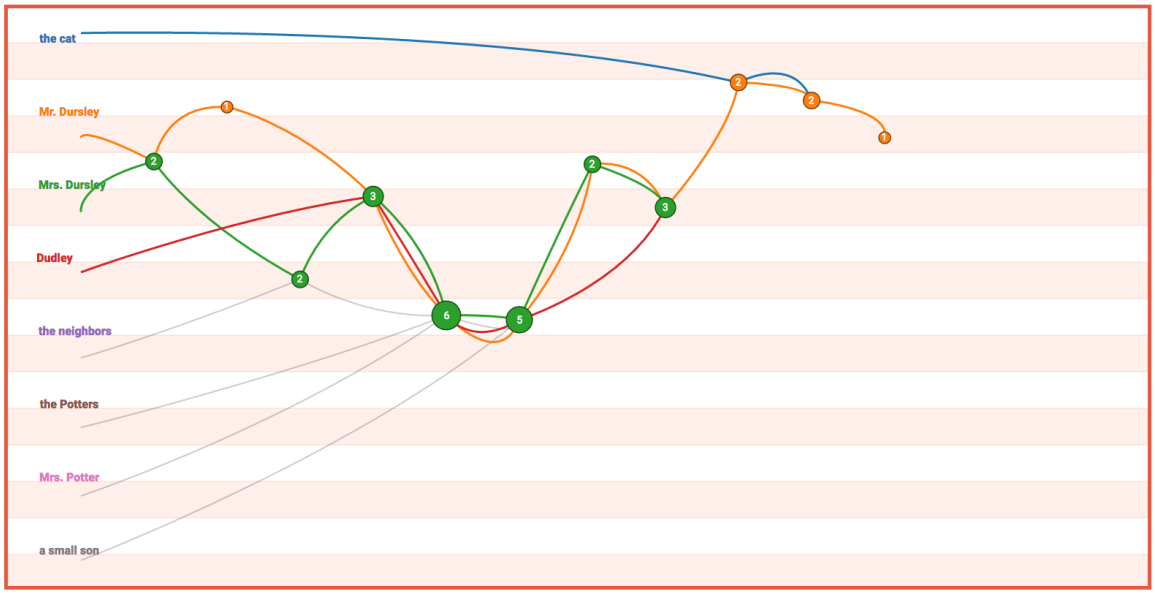


Figure C.8: Harry Potter timeline with the main characters highlighted.

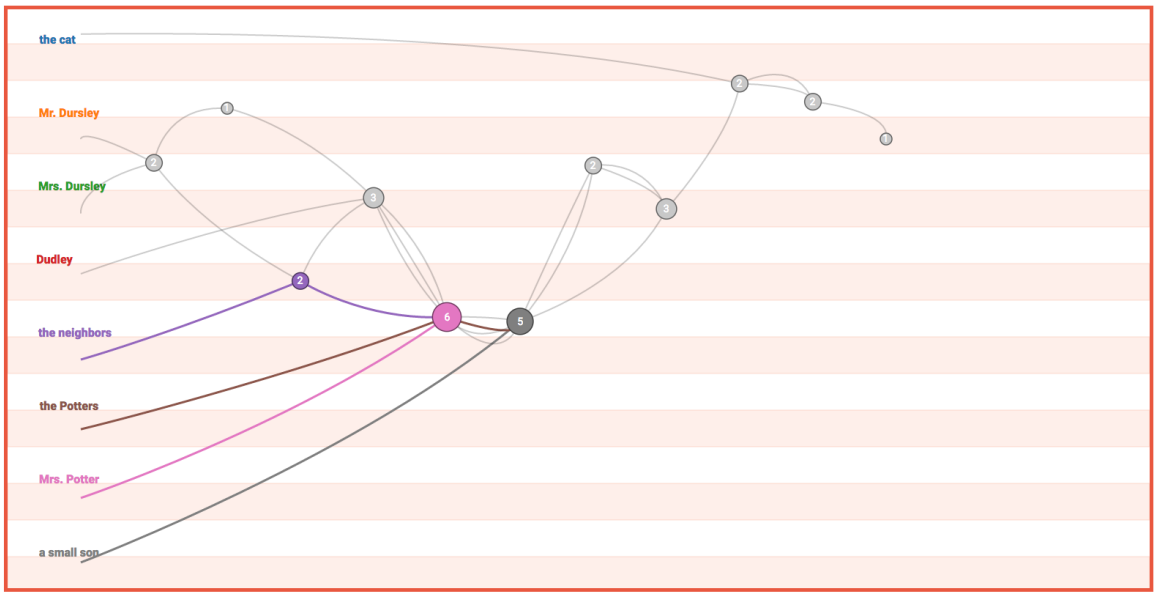


Figure C.9: Harry Potter timeline with the “niche” characters highlighted.

Appendix D

User Guide

D.1 Installation

Executing the application first requires the user to have Java 8 and Node.js installed. Then, please ensure you have the `timeline-generator-clustering-1.0-SNAPSHOT.jar` installed in the `betweenTheLines/application` directory. This is the back-end application that contains all the required dependencies.

Note, initially the application will run without using the TIPSem features. To incorporate TIPSem features, please install TIPSem separately and update the `PATH_TO_TIPSEM` variable in the `EventTagger` object appropriately. Having done this, then execute `mvn package -Dmaven.test.skip=true` to package the application into a JAR without executing the unit tests[1]. This will take a short while as all the dependencies are stored within the resulting JAR. Finally, move this new JAR to the `betweenTheLines/application` directory.

Executing `node app.js` from within the `betweenTheLines` directory will run the application at `http://localhost:3000/`. Additionally, the back-end can be run itself by executing `java -Xmx8g -cp timeline-generator-clustering-1.0-SNAPSHOT.jar App <input text .txt> <optional actor list .txt>`, assuming you're in the same directory as the jar file itself.

[1] Executing the EventTagger unit tests should confirm whether TIPSem has been successfully installed at this point. Additionally, please ensure that unit tests are invoked with the additional argument `-Xmx6g` to allow sufficient heap space.

D.2 Usage

D.2.1 Enter input text and list any specific characters

We begin by entering our story into the input box, and additionally adding any characters that we wish to ensure are included in the resulting timeline to the *actor list* on the right of the input box. Explicitly listing actor names is particularly effective when you have actors with fictional names, such as *Goldilocks*, nominal names, such as the Papa bear, or titled names, such as *Mr. Jones*. Additionally, for optimal results please also select the gender of each character if appropriate. However, please be aware that selecting the *wrong* gender is more detrimental than selecting *no* gender. This is illustrated in Figure D.1.

Clicking "*Let's Go!*" will process the input and redirect you to the insights page after approximately 1 minute; this does take a little while so enjoy the loading screen.

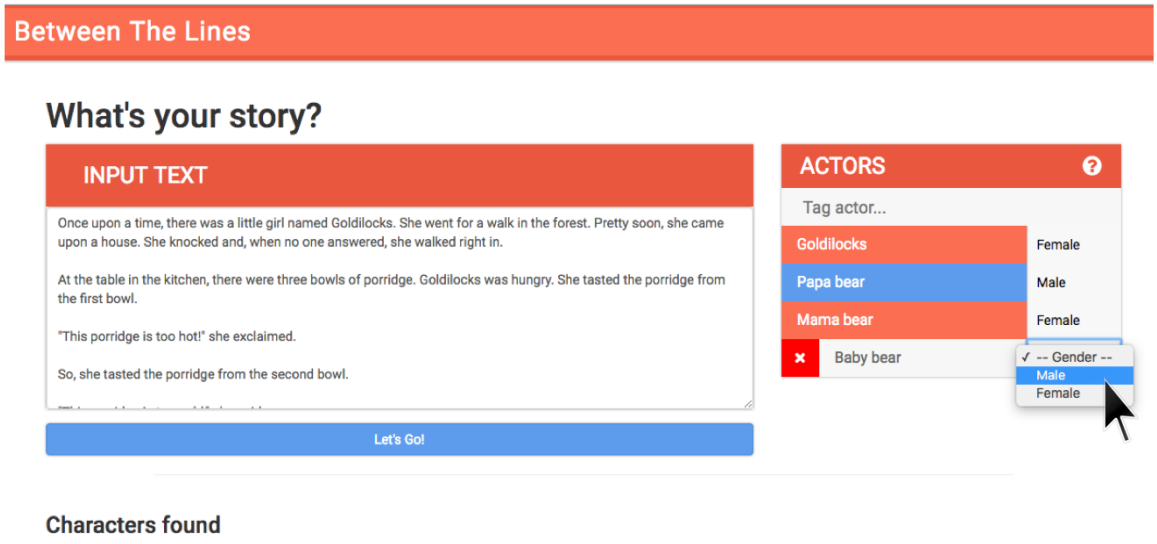


Figure D.1: Entering input to site, and tagging any actors. Adding an actor to the list is done by simply clicking in the input box labelled "Tag actor..." and typing the name.

D.2.2 Remove irrelevant mention annotations

We next remove any redundant actors from the plot using the *quick-delete* buttons in the actor side panel to the right. The application cannot currently distinguish between what is a character and what is some other entity, while we as a user can quickly distinguish between which mention annotations are relevant to our timeline and those that are not. Figure D.2 highlights the quick-delete button that enables us to quickly filter this list.

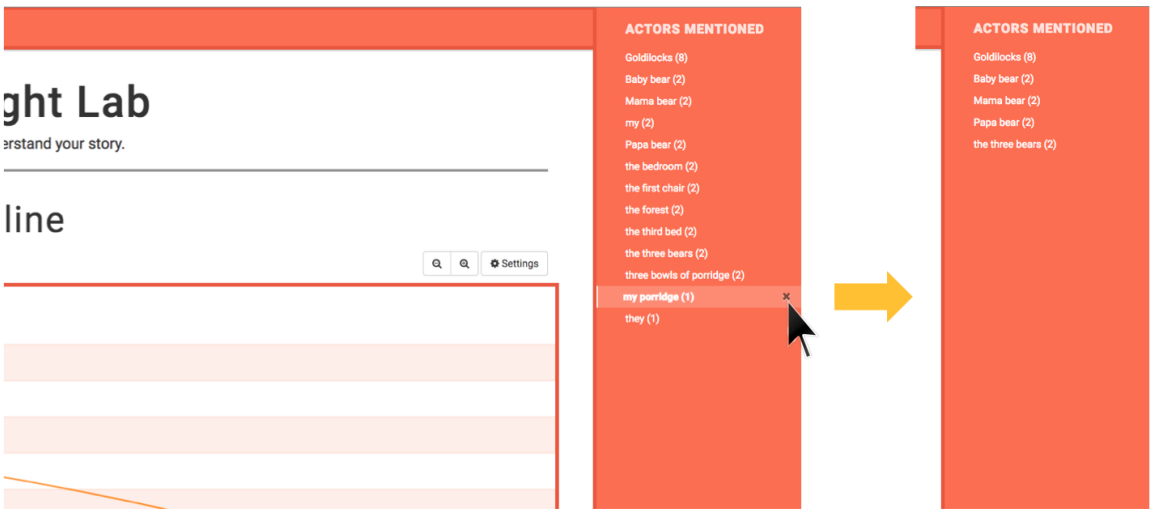
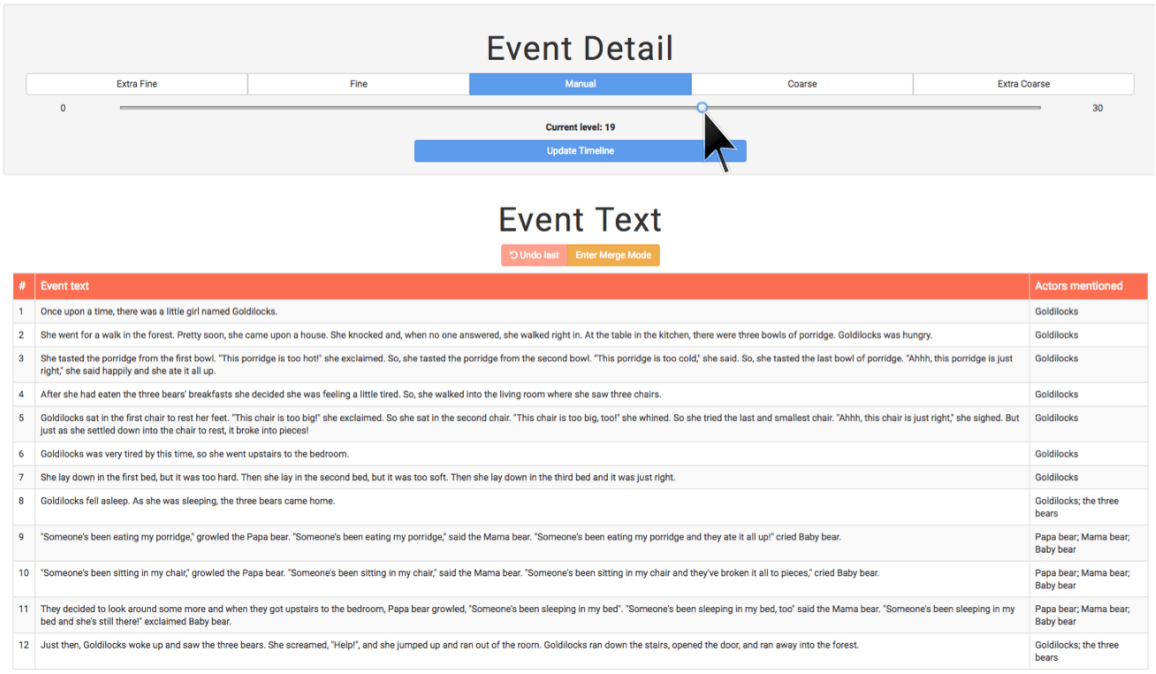


Figure D.2: Removing any irrelevant actors from the *actor side panel* to leave us with just those characters we're interested in.

D.2.3 Identify a nice collection of events

Next, either use the buttons to see the automatically selected event sets at the *fine* or *coarse* levels of detail, or drag the slider to browse through the events obtained at each level of detail.



#	Event text	Actors mentioned
1	Once upon a time, there was a little girl named Goldilocks.	Goldilocks
2	She went for a walk in the forest. Pretty soon, she came upon a house. She knocked and, when no one answered, she walked right in. At the table in the kitchen, there were three bowls of porridge. Goldilocks was hungry.	Goldilocks
3	She tasted the porridge from the first bowl. "This porridge is too hot!" she exclaimed. So, she tasted the porridge from the second bowl. "This porridge is too cold," she said. So, she tasted the last bowl of porridge. "Ahhh, this porridge is just right," she said happily and she ate it all up.	Goldilocks
4	After she had eaten the three bears' breakfasts she decided she was feeling a little tired. So, she walked into the living room where she saw three chairs.	Goldilocks
5	Goldilocks sat in the first chair to rest her feet. "This chair is too big!" she exclaimed. So she sat in the second chair. "This chair is too big, too!" she whined. So she tried the last and smallest chair. "Ahhh, this chair is just right," she sighed. But just as she settled down into the chair to rest, it broke into pieces!	Goldilocks
6	Goldilocks was very tired by this time, so she went upstairs to the bedroom.	Goldilocks
7	She lay down in the first bed, but it was too hard. Then she lay in the second bed, but it was too soft. Then she lay down in the third bed and it was just right.	Goldilocks
8	Goldilocks fell asleep. As she was sleeping, the three bears came home.	Goldilocks; the three bears
9	"Someone's been eating my porridge," growled the Papa bear. "Someone's been eating my porridge," said the Mama bear. "Someone's been eating my porridge and they ate it all up!" cried Baby bear.	Papa bear; Mama bear; Baby bear
10	"Someone's been sitting in my chair," growled the Papa bear. "Someone's been sitting in my chair," said the Mama bear. "Someone's been sitting in my chair and they've broken it all to pieces," cried Baby bear.	Papa bear; Mama bear; Baby bear
11	They decided to look around some more and when they got upstairs to the bedroom, Papa bear growled, "Someone's been sleeping in my bed". "Someone's been sleeping in my bed, too" said the Mama bear. "Someone's been sleeping in my bed and she's still there!" exclaimed Baby bear.	Papa bear; Mama bear; Baby bear
12	Just then, Goldilocks woke up and saw the three bears. She screamed, "Help!", and she jumped up and ran out of the room. Goldilocks ran down the stairs, opened the door, and ran away into the forest.	Goldilocks; the three bears

Figure D.3: Drag the slider or use the button pre-sets to find an event set of your liking.

D.2.4 Merge any odd events

If there are any events that you feel should have been merged into one in your chosen event set, click the "*Enter Merge Mode*" button above the event table and then select the set of contiguous events you'd like to merge. Finally, click the "*Merge Events*" button to make the change. Following this, you can also undo the change by clicking the "*undo*" button.

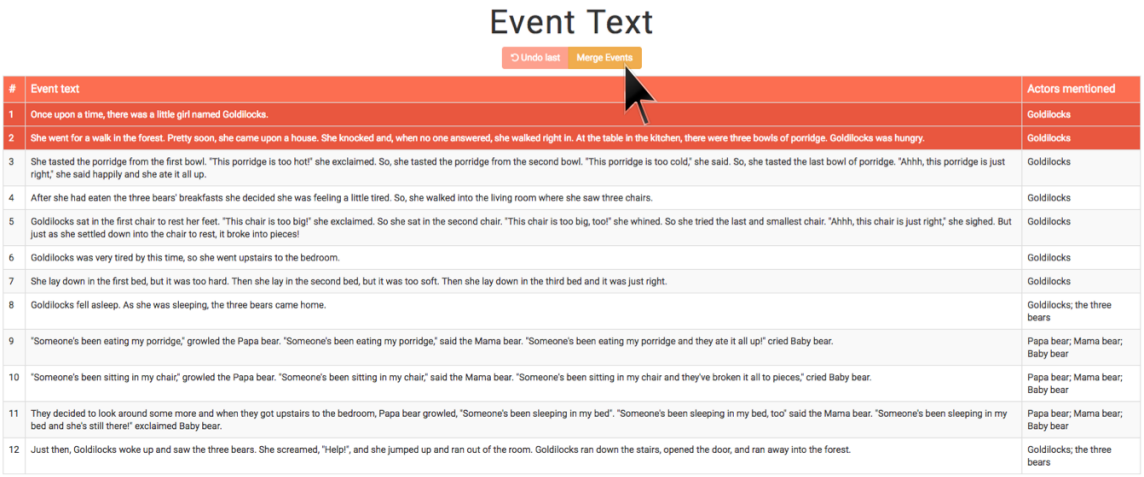


Figure D.4: We've entered *merge mode* here and selected the top two events to merge as one. Clicking the "*Merge Events*" button will merge these events into one.

D.2.5 Add missing mentions

We can now read through our set of events to check if there are any mentions the application has missed. If so, clicking anywhere on that event will open the "*Edit Event*" panel, where we can add these additional mentions ourselves. If the missing mention should match one of the characters already tagged elsewhere, we must be careful to spell the mention in exactly the same way so that the application correctly treats these as the same character. This is shown in Figure D.5.

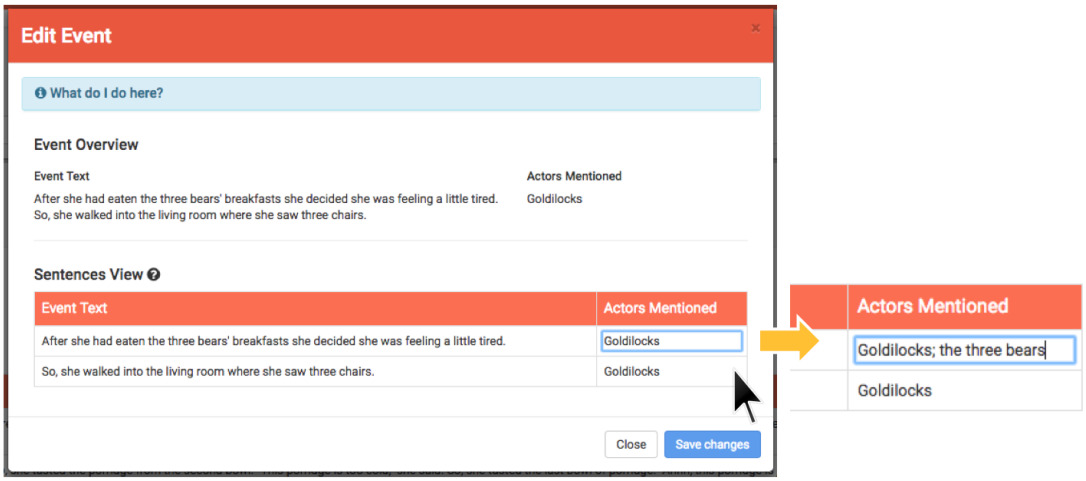


Figure D.5: The "*Edit Event*" panel, where we see a missed mention of *the three bears*. We subsequently tag this additional character by typing the name after a semi-colon and space character to separate the distinct characters mentioned in this sentence.

D.2.6 Update timeline

Following our minor tweaks, we can now update the timeline to see how it's looking and perhaps gain further insight as to any final refinements required. The timeline is not actively updated following every change we make, so we must click the "*Update Timeline*" button shown in Figure D.6 to refresh the timeline.

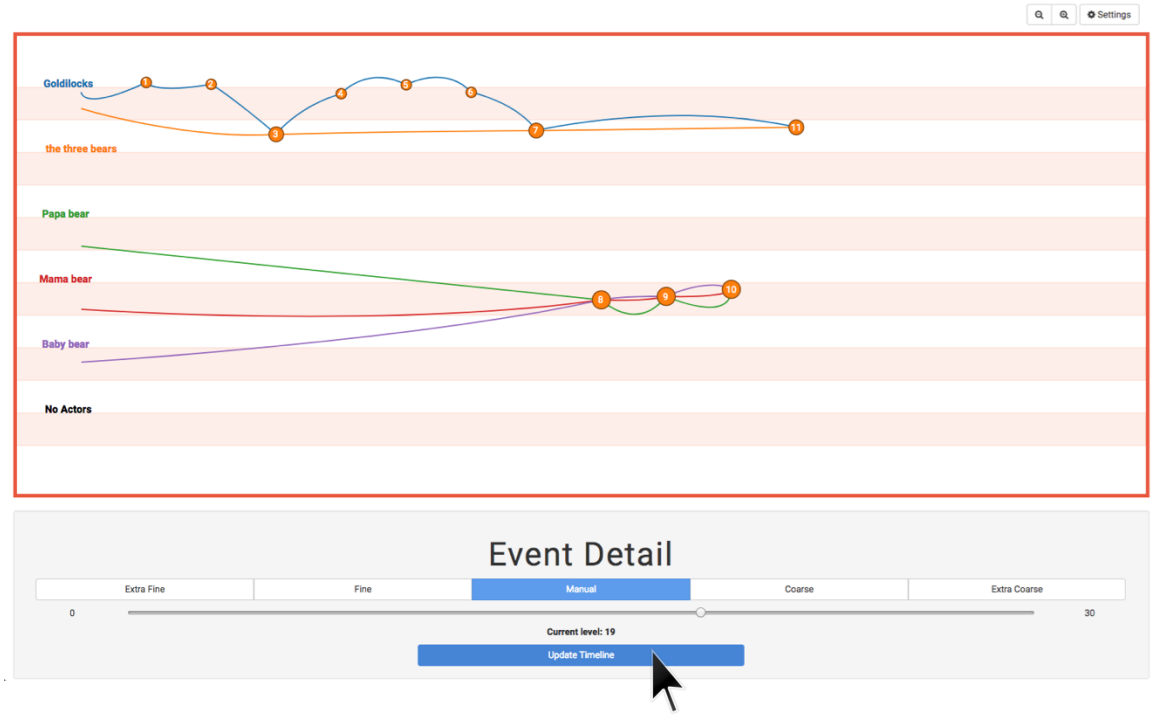


Figure D.6: Our resulting timeline reflecting all our changes having clicked the "*Update Timeline*" button.

D.2.7 Rename actors

In this example it is clear that "*the three bears*" is actually a collective reference to all three of the individual bears in the story. We'd thus rather rename this mention appropriately to make our timeline reflect this, rather than leaving these as distinct paths in our timeline. To rename an actor, we click on the actor's name in the *actor side panel* to the right of the screen, which opens the *Edit Actor* panel. Here, we can then rename "*the three bears*" to instead constitute each of the three bears, separating each name with a semi-colon and space. This process is illustrated by Figure D.7. Note, to see this change take effect in the timeline we must again click "*Update Timeline*".

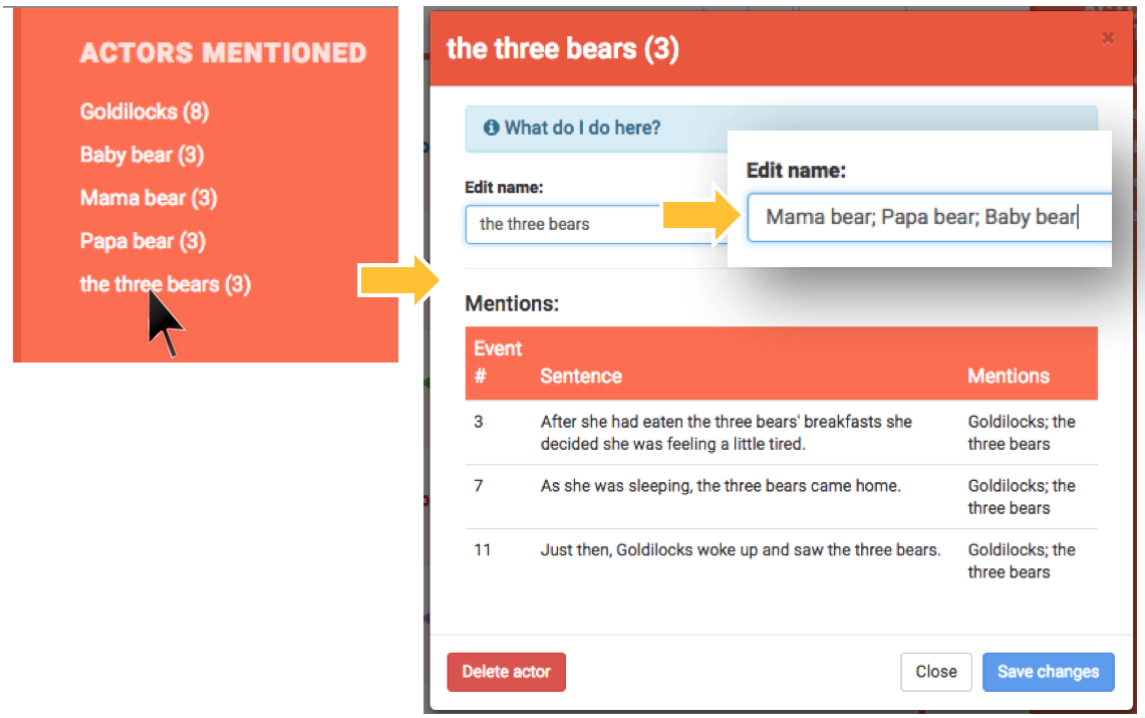
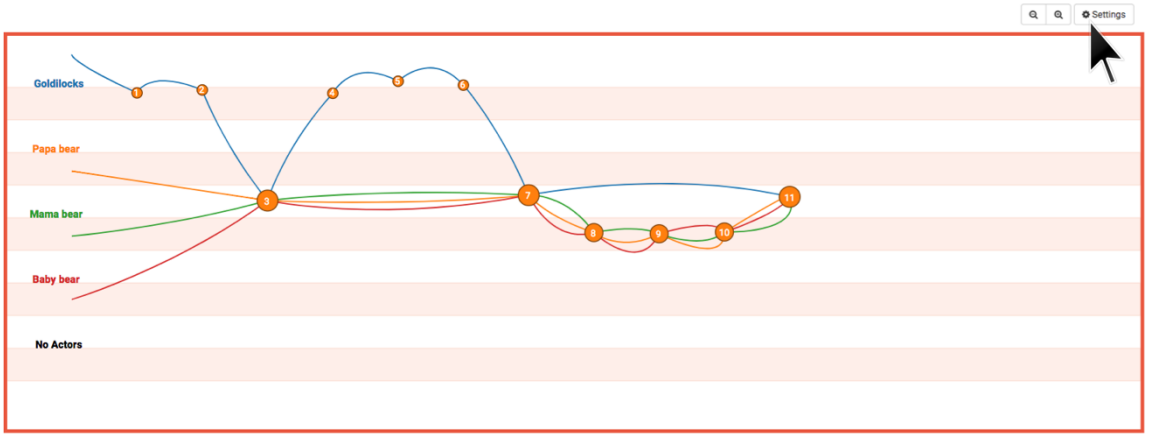


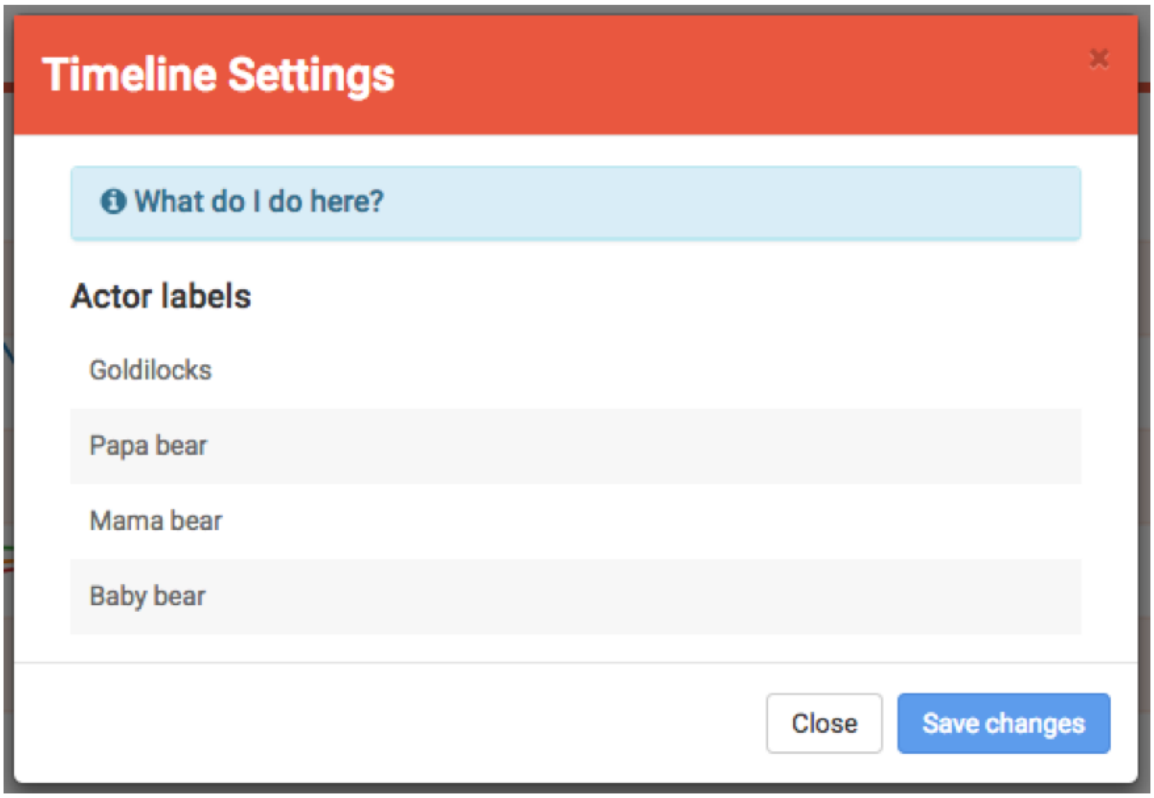
Figure D.7: Renaming "*the three bears*" to each of the three bears, separating names with a semi-colon and space.

D.2.8 Reorder characters in the timeline and zoom

In our case we have quite a clear timeline already by this point, but in some cases some reordering of the character labels on the left hand side of the timeline can help significantly. To do this, we click on the "*Settings*" button above the timeline. This opens the *Timeline Settings* panel where we can drag-and-drop actor names to reorder them in the timeline. This is shown in figures D.8a and D.8b. Additionally, the timeline zoom buttons are located to the left of the "*Settings*" button.



(a)



(b)

Figure D.8: (a) Opening the *Timeline Settings* panel, which is subsequently shown in (b).

D.2.9 Highlight paths

With our final timeline drawn, we can finally begin to explore and interact with the result. Hovering over any character name in the timeline will highlight the path of just that character through the timeline. Clicking on a character name will permanently keep that characters path highlighted, and subsequently hovering over any other character names will also highlight their paths through the timeline. This is shown in Figure D.9. Clicking anywhere else on the timeline canvas will reset the highlighting.

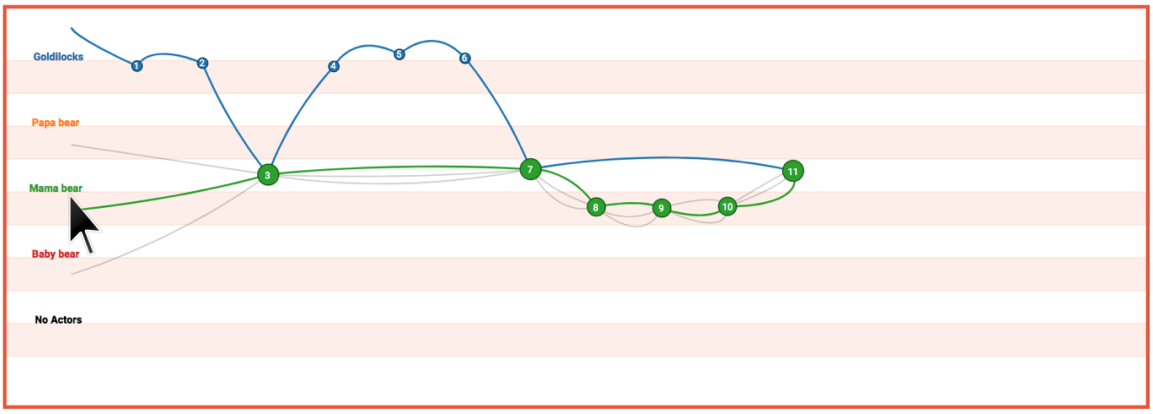


Figure D.9: Highlighting the paths of *Goldilocks* and the *Mama bear* in the timeline.

D.2.10 Hover-over text

Hovering over any of the event nodes in the timeline will display the text of that event to quickly allow you to see what's happening at various points in the timeline. Similarly, the event numbers in the timeline also correspond to the event numbers in the "*Event Table*" at the bottom of the page.

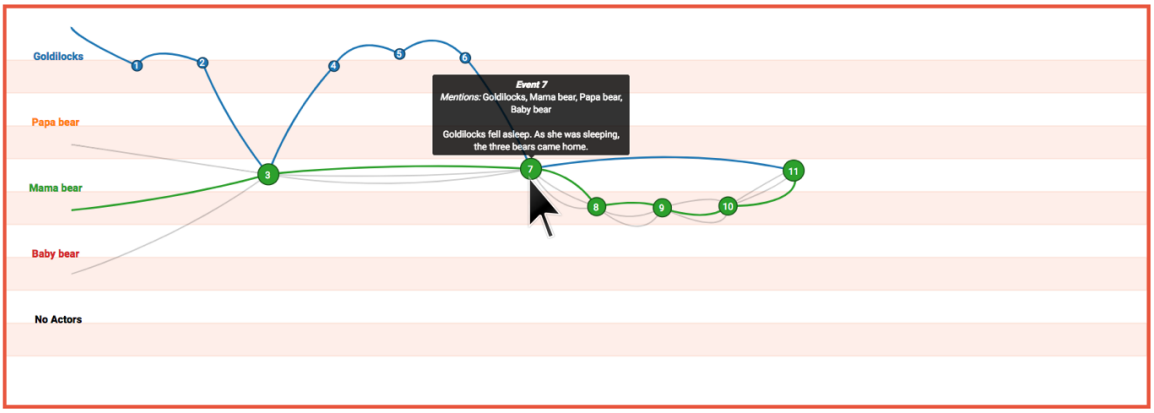


Figure D.10: Hovering over the nodes in the timeline to read what is expressed by that event.