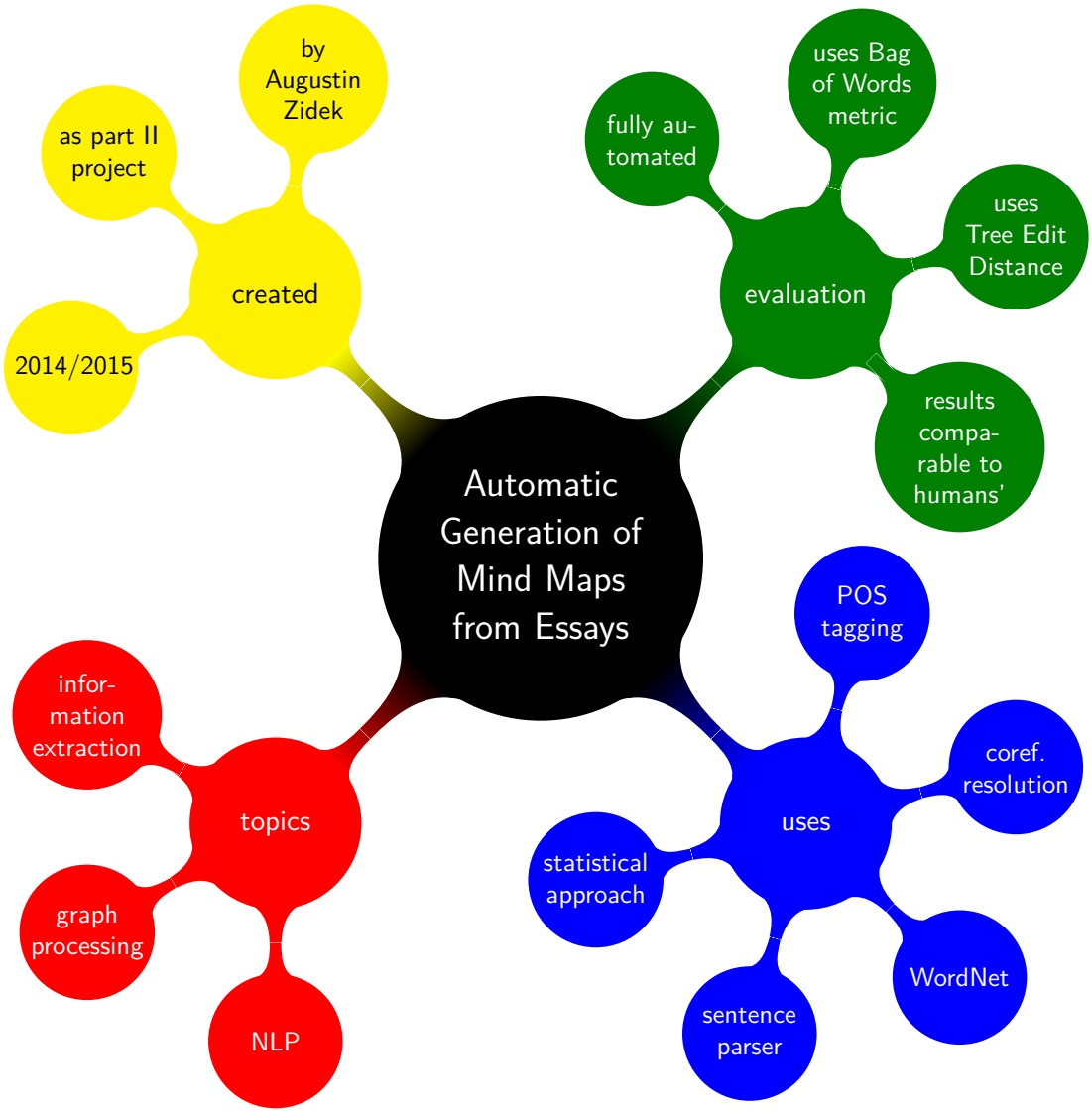
Augustin Zidek



Automatic Generation of Mind Maps from Essays

Computer Science Tripos, Part II

Gonville and Caius College

May 7, 2015

Porphyry of Tyre ($\Pi o \rho \varphi \upsilon \rho \iota o \varsigma$, c. 234 – c. 305 AD) was a Neoplatonic philosopher who graphically visualised the Aristotle's Categories, thus being the first known person who has created a mind map.

Proforma

Name:	**Augustin Zidek**
College:	Gonville and Caius College
Project Title:	**Automatic Generation of Mind Maps from Essays**
Examination:	Computer Science Tripos, July 2015
Word Count:	**11,850**[1]
Project Originator:	Augustin Zidek
Supervisor:	Paula Buttery

Original Aims of the Project

- Develop a system capable of mind map generation and keyword extraction from given plain text essays.
- Evaluate various versions of the system against mind maps produced by people.

Work Completed

The completed work includes four bag of words extractors, each using different technique for word scoring. The mind map extraction system consists of:

- triplet, attributed triplet, multi-object triplet and attributed multi-object triplet extractors using either OpenNLP or CoreNLP as the back-end,
- coreference resolver using CoreNLP,
- WordNet synonym finder,
- mind map from triplets generator,
- fully automated evaluation system.

Special Difficulties

None.

[1] Computed using TeXcount web service, http://app.uio.no/ifi/texcount/online.php

Declaration

I, Augustin Zidek of Gonville and Caius College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed:

Date:

Contents

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Context and related work . . . 2
1.3 Overview of the dissertation . . . 3

2 Preparation **5**
2.1 Introduction to Natural Language Processing . . . 5
2.1.1 Part-Of-Speech tagging . . . 5
2.1.2 Parsing . . . 6
2.1.3 Coreference resolution . . . 7
2.1.4 WordNet – synonym finding . . . 8
2.1.5 Corpora – statistical approach to NLP . . . 9
2.2 Bag of Words Distance . . . 9
2.3 Tree Edit Distance . . . 10
2.4 Requirement analysis . . . 11
2.5 Risk Analysis . . . 11
2.6 Software engineering . . . 12
2.7 Employed tools . . . 13
2.7.1 Choice of the programming language . . . 13
2.7.2 Libraries . . . 13
2.7.3 Graph visualisation . . . 14
2.7.4 Environment and other tools . . . 14
2.7.5 Testing . . . 14

3 Implementation **16**
3.1 System overview . . . 16
3.1.1 Overview of the pipeline . . . 16
3.1.2 Data structures . . . 18
3.1.3 Custom mind map file format . . . 19
3.2 Bag of words extractors . . . 20
3.2.1 BOW by frequency . . . 20
3.2.2 BOW by frequency with common words elimination . . . 21
3.2.3 BOW by frequency with function words elimination . . . 21
3.2.4 Keyword extraction . . . 21
3.3 The mind map pipeline . . . 23
3.3.1 Triplet extraction algorithm . . . 23
3.3.2 Coreference resolution . . . 27
3.3.3 Synonym finding and grouping using WordNet . . . 29

3.3.4 Building the mind map from triplets . . . 30
3.3.5 Exporting and visualising the mind map . . . 31

4 Evaluation **34**
4.1 Implementation of the evaluation system . . . 34
4.2 Methodology . . . 35
4.2.1 Lemmatisation . . . 36
4.2.2 Statistical analysis . . . 36
4.2.3 Explanation of abbreviations . . . 37
4.3 BOW approach evaluation . . . 37
4.3.1 Results . . . 38
4.3.2 Discussion . . . 40
4.4 Tree Edit Distance evaluation . . . 41
4.4.1 Results . . . 42
4.4.2 Discussion . . . 45
4.5 Alternative evaluation . . . 46
4.6 Summary . . . 47

5 Conclusion **49**
5.1 Achievements . . . 49
5.2 Outputs from the work . . . 49
5.3 Future work . . . 50

Bibliography **51**

A Most common 500 English words **53**

B English function words **54**

C Graph visualisation using GraphViz **55**

D Essays used for system evaluation **56**

E Instructions for human participants **60**

F Human participants' mind maps **61**

G Project Proposal **69**

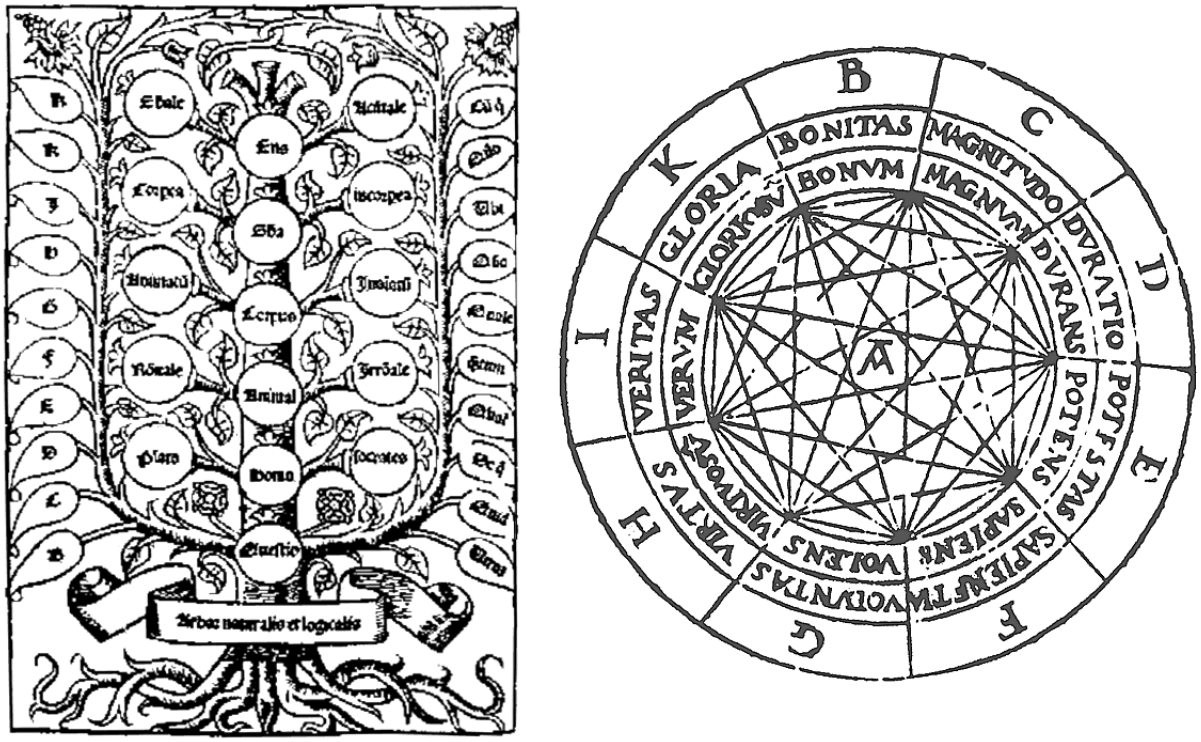
Acknowledgements

Many people have contributed to this project by giving advice, feedback or drawing mind maps for the evaluation. However, I owe special thanks to:

Paula Buttery for supervising this project and giving many good suggestions and pointers to relevant research papers and topics.

Petra Vlachynska for always being interested in my work, providing useful feedback, and making me tea when I was too busy, programming.

My father for giving valuable feedback and for teaching me good software design practices.



An early mind map by Ramon Llull (c. 1232 – c. 1315), Majorcan writer, philosopher and logician [1]

Chapter 1

Introduction

This dissertation describes the creation of a system that is capable of extracting important information from text and presenting it to the user in the form of a mind map.

1.1 Motivation

Since the invention of movable type by Johannes Gutenberg in 1452, the number of books that have been printed has been increasing. Moreover, with the growth of IT, the amount of written information that surrounds us is even greater – we live in the so called information age.

But with the huge amount of written information available, it is harder and harder to find the relevant pieces of information or to extract them from long texts.

Mind mapping is one of the many techniques designed to alleviate this problem. Mind mapping is a powerful technique of organising your information as a diagram. A mind map is often created around a single concept, drawn as an image in the centre of a blank landscape page, to which associated representations of ideas such as images, words and parts of words are added. Major ideas are connected directly to the central concept and other ideas branch out from those. [2]

Information stored in a mind map is easier to find, categorise, process and even remember. However, it takes some time to create a mind map and this is where my system comes in. The system I have developed implements a very simple contract: The input is a plain text and the system outputs a mind map for it, containing the most important pieces of information while representing the relationships between them.

Take the following text for instance: *EDSAC was an early British computer. It was constructed by Maurice Wilkes and his team at the University of Cambridge Mathematical Laboratory. EDSAC was the second electronic digital stored-program computer to go into regular service. EDSAC ran its first programs on 6 May 1949, when it calculated a table of squares and a list of prime numbers. EDSAC was finally shut down in 1958. It was capable of 650 instructions per second. It had 1024 17-bit words of memory in mercury ultrasonic delay lines. It had Paper tape input and teleprinter output at 6 2/3 characters per second. It had 3000 valves, 12 kW power consumption, needed a room 5m by 4m.*

The system that has been developed would generate this mind map:

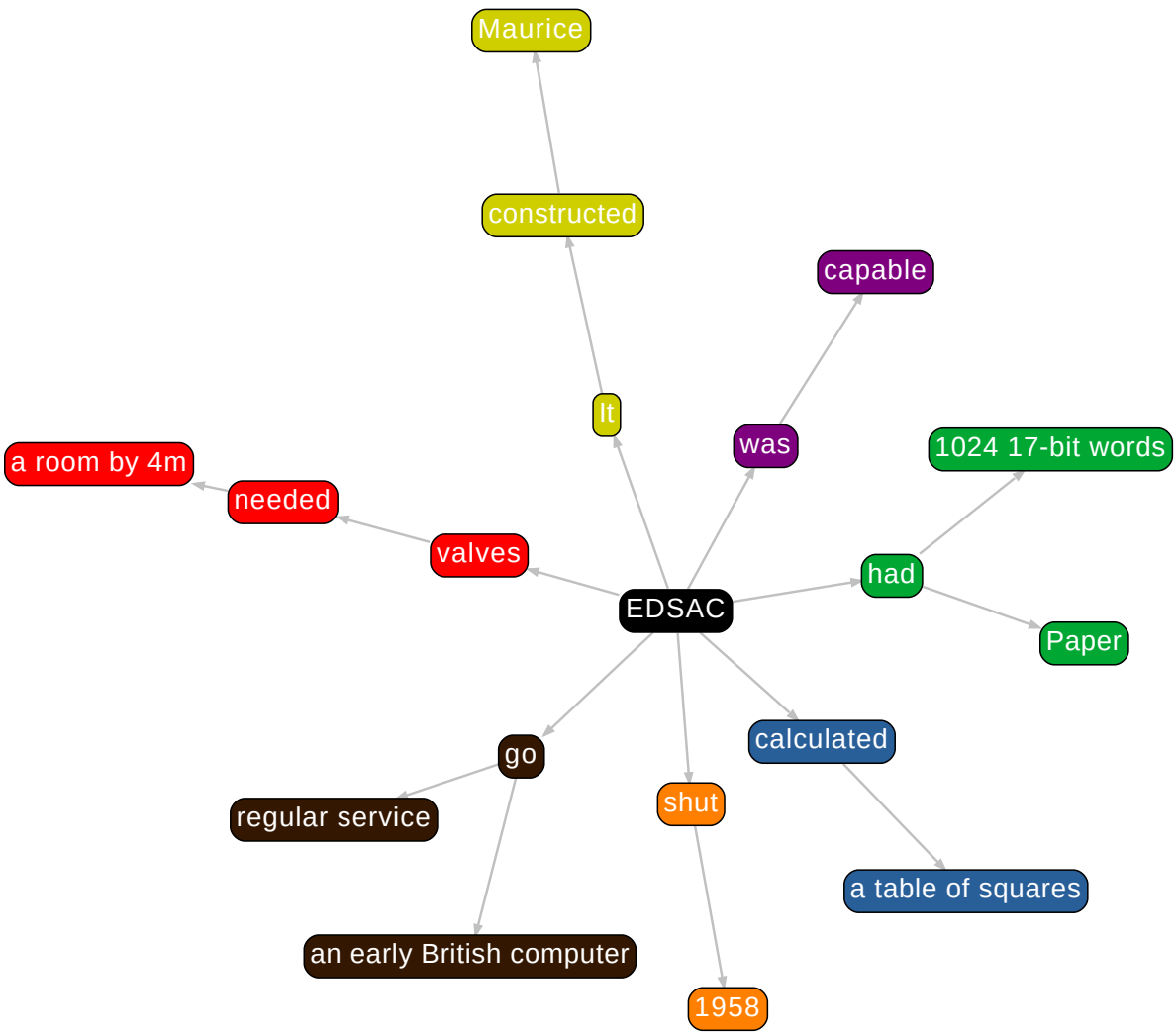


Figure 1.1: Mind map generated by the system for the essay about EDSAC

1.2 Context and related work

The project is a Natural Language Processing (NLP) project. It uses large state-of-art NLP libraries a utilities including CoreNLP, WordNet and Google Trillion Word Corpus.

Information retrieval, extraction and NLP are areas that are being actively researched, especially nowadays when there is enough computing power to process large texts and do things like sorting email, finding relevant pages in web search, summarising news articles, etc.

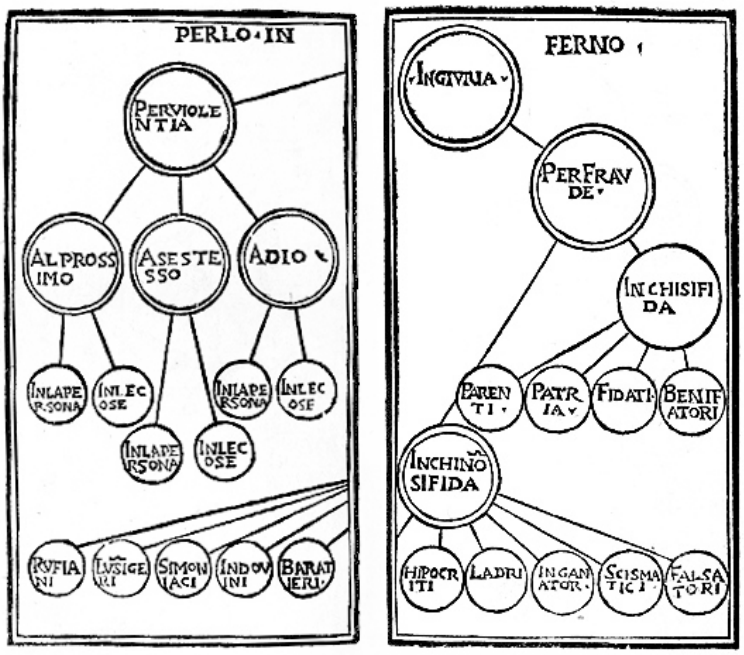
Similar work has been done by Brucks and Schommer in their Assembling Actor-based Mind-Maps from Text Streams [3]. Their work, however, focused more on the correct pronoun and coreference resolution. A more in-depth approach was taken by Saelan et al. in Generating Mind Map from Indonesian Text Using Natural Language Processing Tools [4].

Independent approach which is capable of processing long texts was taken by Elhoseiny et al. They developed the English2MindMap system [5], however, the paper describing it wasn't very descriptive and detailed.

1.3 Overview of the dissertation

The dissertation is split into five main chapters:

- **Introduction**.
- **Preparation** gives an introduction to relevant NLP techniques and presents an overview of the software engineering, structure of the system and of the libraries used.
- **Implementation** explains in depth how the important components of the system work.
- **Evaluation** describes methods for mind map similarity comparisons that were used (Bag of Words Distance and Tree Edit Distance) and then compares the mind maps generated by the system to the ones drawn by humans.
- **Conclusion** gives an overview of the work done and plans for future work.



Moral schema of Hell – a mind map drawn by Alessandro Paganini in 1527. Alessandro Paganini was an Italian printer and publisher. [1]

Chapter 2

Preparation

This chapter gives an introduction to relevant Natural Language Processing (NLP) algorithms. Two similarity metrics used for the evaluation are explained here together with the requirement analysis and planning of the project. The chapter concludes with a summary of the libraries used in the project.

2.1 Introduction to Natural Language Processing

The project relies heavily on NLP libraries and hence it is essential to explain how the underlying algorithms work. Descriptions of the algorithms and techniques were mostly adapted from Jurafsky and Martin [6].

2.1.1 Part-Of-Speech tagging

Part-Of-Speech (POS) tagging is the process of annotating words in the given sentence with their POS tags. Commonly, the Penn Treebank tag set [7] is used for tagging.

The problem with tagging is the ambiguity of certain words – i.e. in some cases it is not clear what POS tag a word should be tagged with. Take the word *map*. Are we talking about the map that is used when navigating around a city? Or about the process of creating a map, i.e. the verb *to map*? Therefore we want to create a POS tagger that takes into account the probability of having a word with a certain tag in a certain context and uses that to assign the most probable tag to the word.

One could think that the number of words that are ambiguous concerning their POS tag is small. However, according to Jurafsky and Martin the percentage of words that have 2 to 7 possible POS tags is about 40%. The most extreme example of such a word is the word "still", which has 7 possible tags!

There are multiple approaches to tackle the problem of POS tagging. The one that is used (with multiple additions and enhancements!) by CoreNLP is the Hidden Markov Model (HMM) for POS tagging. The HMM approach doesn't try to find a tag for a single word, but rather for a sequence of consecutive words and hence uses the information about the context of each word.

Hence, for a sequence of tags $T = t_1, t_2, \ldots, t_n$ given the sequence of words in a sentence $W = w_1, w_2, \ldots, w_n$, we want to find the most probable sequence of tags $\hat{T}$, such that

$$\hat{T} = \underset{T}{\operatorname{argmax}}\, P(T|W). \tag{2.1}$$

By Bayes Theorem, we obtain

$$P(T|W) = \frac{P(W|T)P(T)}{P(W)}. \tag{2.2}$$

The sequence of words is given, hence $P(W)$ will be a constant. Therefore to compare these and search for T which maximises the whole formula, we can use

$$\hat{T} = \underset{T}{\operatorname{argmax}}[P(W|T)P(T)]. \tag{2.3}$$

Now we need to estimate the values of $P(W|T)$ – the probability of a sequence of words given the sequence of tags and $P(T)$ – the probability of a sequence of tags. In order to do that two assumptions have to be made:

1. A word is dependent only on *its* tag: $P(w_i|w_1, t_1, \ldots, w_{i-1}, t_{i-1}, t_i) = P(w_i|t_i)$.
2. The tag history can be approximated by the two most recent tags: $P(t_i|w_1, t_1, \ldots, w_{i-1}, t_{i-1}) = P(t_i|t_{i-1}, t_{i-2})$.

Therefore, using these assumptions the equation 2.3 becomes

$$\hat{T} = \underset{T}{\operatorname{argmax}} \left[\left(\prod_{i=1}^{n} P(w_i|t_i) \right) \left(\prod_{i=3}^{n} P(t_i|t_{i-1}, t_{i-2}) \right) P(t_2|t_1)P(t_1) \right]. \tag{2.4}$$

Luckily, the equation 2.4 can be solved as it is possible to gather all the data it needs. To do this, a POS tagged corpus of training data has to be created. Such corpora exist – for instance the Brown Corpus – and hence the data needed for classification (i.e. 3-grams, 2-grams, 1-grams and word given POS-tag probabilities) can be extracted.

These methods can be further refined by **smoothing** (adding a small constant to all probabilities so that the probabilities are not zero for unknown words) and by applying additional methods such as **maximum-entropy-based POS tagging** (improves performance of the tagger especially on unknown words by expanding the information sources), as done in the state-of-art POS tagger in CoreNLP [8] the project uses. The accuracy of such POS taggers is then over 96%.

2.1.2 Parsing

Parsing, in the context of NLP, is the process of analysing a string of symbols (which form words and sentences) and trying to identify the underlying structure given by the grammar of the language being parsed. Hence the input of a parser is a sentence and its output is a parse-tree. The parse-tree consists of nodes defining the structure of the sentence, such as *Noun Phrase (NP)* or *Verb Phrase (VP)*, and of leaves, which are the original words of the sentence. Moreover, each word gets a POS tag (see Section 2.1.1).

An example of a parse tree for the sentence "*There was no possibility of taking a walk that day.*" [9] is in the Figure 2.1 below:

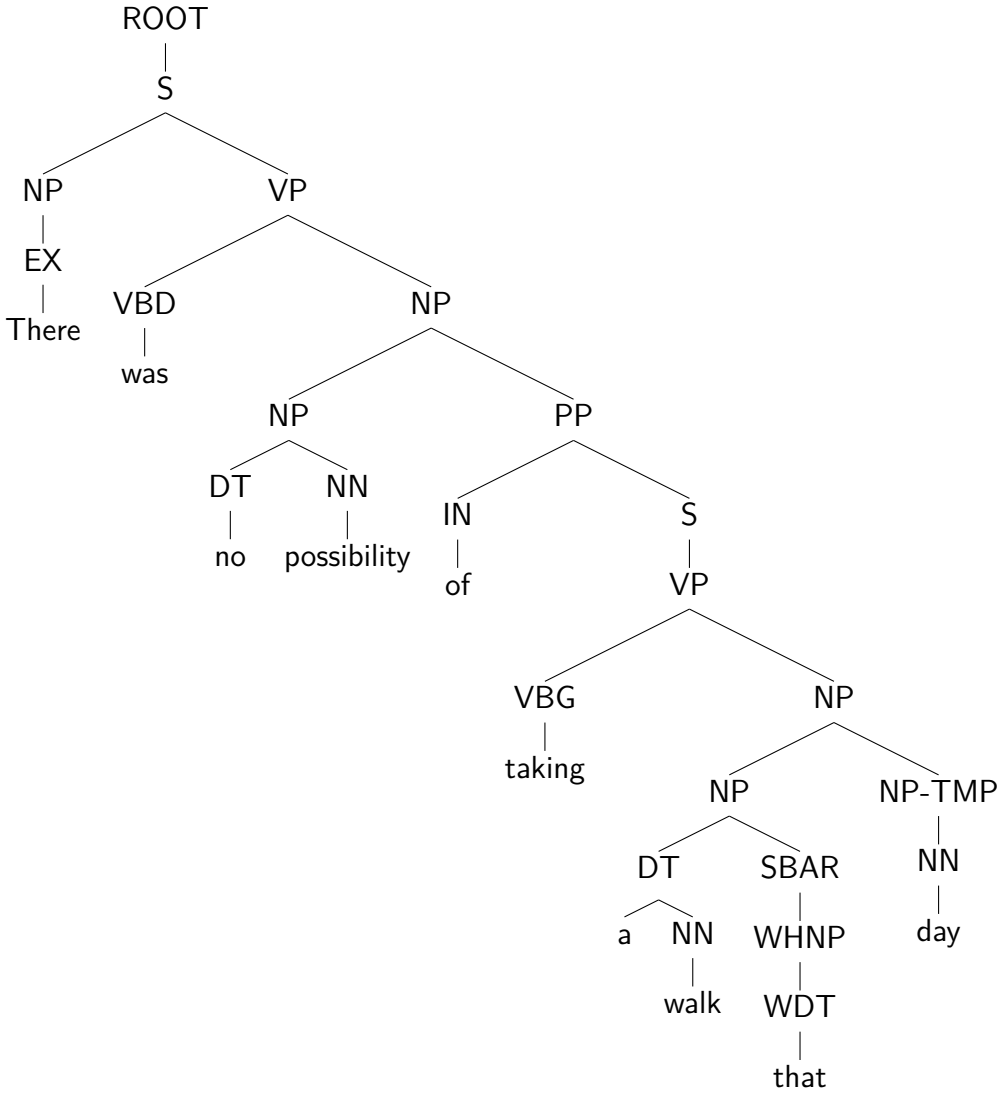


Figure 2.1: A sample parse tree

The project uses the CoreNLP parser which is a highly optimised Probabilistic Context-Free Grammar (PCFG) parser [10]. The parse tree obtained from CoreNLP is used in the project to find triplets in sentences.

2.1.3 Coreference resolution

Coreference resolution is the task of identifying phrases that refer to the same object. For instance, in the text "*'Laugh and be merry!* ***I*** *am* ***Goldberry***, ***daughter of the River****.' Then lightly* ***she*** *passed them and closing the door* ***she*** *turned* ***her*** *back to it, with* ***her*** *white arms spread out across it. 'Let us shut out the night!'* ***she*** *said.*" [11] The phrases in **bold** all refer to Goldberry and coreference resolution system should correctly identify that.

The CoreNLP coreference resolver uses the following steps to find the coreference sets [12].

1. Identify textual mentions – i.e. referring phrases. Those are usually noun phrases (NP) headed by nominal or pronominal terminals.

2. For each mention m_i the resolver will either deterministically propose one of the antecedent mentions from the set $m_1, \ldots, m_{i-1}$ as the coreferring mention or it will decline to find one if there is a high probability one will be found later in the text.
3. Candidate antecedent mentions are sorted using syntactic information provided by the parser.
4. The best mention is selected in a 7 stage sieve process, using features that give hints about the coreference such as: exact match, acronym, word inclusion, compatible modifiers and pronoun match which looks at matches in number, gender, person, animacy and Named Entity Recognition (NER) label.

The output of the CoreNLP coreference resolver is a list of coreference chains. This is a list of phrases which all refer to the same object.

The project uses its own custom heuristic processing on the obtained chains to select the best representative of the chain by comparing the uniqueness of single-word mentions. See Section 3.3.2 for a more detailed explanation.

2.1.4 WordNet – synonym finding

WordNet [13] is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations. [13]

An example of a group of synsets for the word "spring" is below.

```
(n) spring, springtime (the season of growth)
(n) spring (a metal elastic device that returns to its shape
    or position when pushed or pulled or pressed)
(n) spring, fountain, outflow, outpouring, natural spring (a
    natural flow of ground water)
(n) spring (a point at which water issues forth)
(n) give, spring, springiness (the elasticity of something
    that can be stretched and returns to its original length)
(n) leap, leaping, spring, saltation, bound, bounce (a light,
    self-propelled movement upwards or forwards)
(v) jump, leap, bound, spring (move forward by leaps and
    bounds)
(v) form, take form, take shape, spring (develop into a
    distinctive entity)
(v) bounce, resile, spring, bound, rebound, reverberate,
    recoil, take a hop, ricochet (spring back; spring away from
    an impact)
(v) spring (develop suddenly)
(v) spring (produce or disclose suddenly or unexpectedly)
```

An entry in WordNet provides a list of synonym groups (called synsets) with their description depending on the POS tag of the input word. E.g. if the given word "spring" was a noun then possible synonyms could be "springtime" or "fountain". If it was a verb possible synonyms could be "leap" or "jump".

The project uses the extJWNL library that provides Java interface for WordNet. The synonym finding is used in the project to reduce the number of verb-containing nodes. For a more detailed explanation see Section 3.1.1.

2.1.5 Corpora – statistical approach to NLP

A corpus is a large collection of texts from various sources. Corpora are useful when we need to collect statistical information about words, their context or collocations.

The project uses statistics of word frequencies extracted from the Google Web Trillion Word Corpus [14] [15], which is a collection of texts from all books Google has digitised. The resulting database contains a list of about 97,000 tuples $(word : count)$. Words were included only if their count was above 100,000 to eliminate extremely rare and uncommon words. First couple of entries in the database look like this:

```
THE   53097401461
OF    30966074232
AND   22632024504
TO    19347398077
IN    16891065263
A     15310087895
IS     8384246685
```

$\ldots$

This raw database of tuples $(word : count)$ has been processed in order to create a database of tuples $(word : relative\ frequency)$. This was done by adding all the word counts together and then, once the sum was obtained, replacing each count by $relative\ frequency = count/sum$.

The obtained database was then saved in a hash map, serialised to disk and a class was created for easily querying a word frequency via the method `getWordFrequency(String word)`.

This was later used in the project for keyword extraction. See Section 3.2.4 for more details on the implementation of keyword extraction algorithm.

2.2 Bag of Words Distance

To compare the similarity of two mind maps, one approach taken was to compare just the similarity of the words that were contained in these mind maps (i.e. the bag of words for these mind maps). To compare similarity of two bags of words, i.e. two lists of words possibly with duplicates, the following similarity metric (a modified version of the Jaccard similarity coefficient) was used

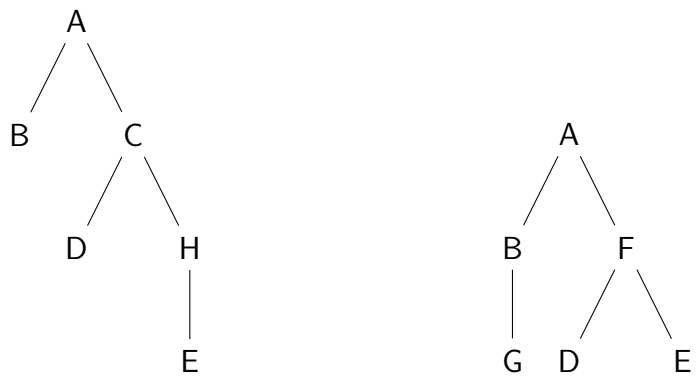
$$D_{BOW} = 1 - \frac{2 \cdot |A \cap B|}{|A| + |B|}, \tag{2.5}$$

where the list intersection $\cap$ is defined as a list of all elements that are both in A and B and the size of the list $|\cdot|$ is defined as the number of elements in the list. Therefore if both A and B contain certain duplicates, then $A \cap B$ will also contain these duplicates. This is desirable as mind maps often contain the same word multiple times in different contexts.

Also, the metric is defined so that $D_{BOW} = 0$ if $A = B$ and $D_{BOW} = 1$ if the two lists do not share any common element.

2.3 Tree Edit Distance

In the context of comparing two mind maps, consider the following problem: Given the two trees below:



(a) Tree 1 (b) Tree 2

What is their minimal Tree Edit Distance, i.e. what is the minimal number of edit operations (insert node, delete node or change label of a node) that need to performed in order to change the first tree into the other tree?

In this case, the answer is three:

1. Rename C $\rightarrow$ F.
2. Insert G as a child of B.
3. Delete H (which re-links C to E).

Tree Edit Distance is useful for comparing mind maps to each other. The reason this metric was used is that it captures well the effort a human would need in order to redraw one mind map into another one. It also takes into account the structure of the mind map. The RTED library was used to compute the Tree Edit Distance. RTED uses an algorithm proposed by Pawlik and Augsten [16] which runs in $O(n^3)$ time and $O(nm)$ space where n and m are the number of nodes in the first and the second tree, respectively.

Note that this metric is not normalised and returns a positive number which denotes the number of edits multiplied by their respective weights (each operation has a weight w such that $0 \leq w \leq 1$).

The costs of *insert* and *delete* were set to 1 and the cost of *rename* was set to 0.5 for the purposes of evaluation. The reason for this was to address problems with synonyms and huge penalisation for small differences that humans would consider negligible. For more details see Section 4.4 in the Evaluation chapter.

2.4 Requirement analysis

The requirement analysis, done at the beginning of the implementation, gave the project its structure and also determined the order in which things were implemented. Also, requirements for external libraries were clarified by the requirement analysis. The entire development process was split into goals, each goal corresponding to an independent functional module in the system. The description of the goals and the corresponding modules and dependencies is summarised in the Table 2.1 below. Since the modules form a pipeline and depend on each other, it was also necessary to implement them in order that would enable incremental testing of the system.

Goal	Package	Dependencies	Priority
Triplet extraction	`mindmapper`	Parser access	1
Building mind maps from triplets	`mindmapper`	Graph library	1
Coreference resolution	`mindmapper`	Coreference resolver	3
Synonym finding	`mindmapper`	WordNet access	3
Evaluation – Tree Edit Distance	`evaluation`	Tree Edit Distance	2
Bag of Words extraction	`bowextractors`	Corpus access	1
Evaluation – Bag of Words	`evaluation`	BOW extractors	1
Additional object extractors	`objectextractors`	Basic triplet extraction, parser	3
Tools: IO, CoreNLP convenience methods, corpus, export tools, etc.	`easyjava`, `util`	None	1

Table 2.1: Relationships between goals, packages and their dependencies

2.5 Risk Analysis

The project's weaknesses and strengths were analysed to make sure potential problems would be discovered soon and properly dealt with.

The project weaknesses (i.e. potential problems) were identified in the following:

1. Usage of external NLP libraries, as they are usually complex research and state-of-art projects which tend to be not well documented. Moreover, the project relies heavily on the correctness of the parses provided by the NLP library.
2. Dealing with natural language in the project – natural language processing is in general a complex task that is still very actively researched and for many tasks a standard solution hasn't been found yet.
3. The evaluation of this project is rather complex, because it deals with natural language and user-generated content.

The major identified strength of the project was its modularity. It was possible to split the project into many smaller sub-projects, each dealing with certain problem and each refining the results of the whole system. This increased the flexibility during the development and also lowered the risk of not completing the project.

2.6 Software engineering

The nature of the project implied the use of the *Spiral Development Model* [17], as the project dynamically evolved and the algorithms for mind map extraction were progressively refined.

The project uses Java, hence the Object-Oriented design approach was taken. The project is highly modular and loosely-coupled following the Single Responsibility Principle.

The project uses design patterns to ensure readability and to ease possible extending. The mainly used design patterns were:

- **Singleton pattern:** Used to ensure only one instance of `CoreNLPPipeline` is running and shared among all objects since its loading is slow and it is optimal to load it only once.
- **Strategy pattern:** Used heavily in the project due to the project's nature – the project has multiple algorithms implementing the same interface but providing different results.
- **Builder pattern:** Mind maps can be built using different algorithms and possibly skipping certain stages (e.g. coreference resolution) in the pipeline. The Builder pattern enables this by providing building methods which all return `this`. These methods can be chained in an arbitrary order and finally the `build()` method is invoked which performs the mind map extraction based on the built properties. See Section 3.1.2 for more details on how the Builder pattern was used in mind map building.
- **Factory pattern:** Used to enable iterating over all strategies and adding new strategies during the run time.

Since the project incorporates more than 50 classes, it was split into multiple packages. Each package groups classes with common purpose and provides a public (well-documented) interface for other packages. The main packages in the system are described in the following Table 2.2. Note that the common prefix `eu.zidek.augustin` was omitted for brevity and the last 5 packages are sub-packages of the package `mindmapper`.

Package name	Purpose of the package
`easyjava`	Tools which are part of my own library EasyJava. Classes in this package provide IO, parser, corpus and word statistics utilities and wrappers.
`mindmapper`	The main package of the system containing the synonym finder, coreference resolver, triplet extractor, parse tree manipulating classes and mind map assembling classes.
`bowextractors`	Package containing Bag of Words extractors.
`datastructs`	Package with various data structure objects.
`evaluation`	Package with tools for evaluation and processing of human generated mind maps.
`objextractors`	Package containing sentence object extractors, as there are multiple types of these. See the Implementation section for more information.
`util`	Various utility classes used for instance for Google Trillion Word Corpus pre-processing and graph manipulation (import/export) tools.
`test`	Package with tests.

Table 2.2: Packages in the system

2.7 Employed tools

2.7.1 Choice of the programming language

The main programming language used in this project was Java. There were multiple reasons for this decision:

1. **Availability of NLP libraries.** Java has a huge range of NLP libraries with large user base, active development and state-of-art performance.
2. **Availability of other libraries.** Other libraries used in this project are the WordNet Java library and graph drawing library GraphStream.
3. **Familiarity.** By using it I was able to instantly start learning how the libraries work, instead of having to first familiarise with the language itself.

Java version 8 was used as it offers lambda expressions, which make the code more legible and increase the performance in some cases when used instead of anonymous classes [18].

2.7.2 Libraries

Libraries were crucial for the development of the project as it needs to perform broad range of common NLP tasks which are difficult to implement.

The system is able to use either CoreNLP or OpenNLP as the parser back-end. However, CoreNLP was used as it offers better performance (about twice as fast parsing) and also since it was used for the remaining tasks (lemmatisation, coreference resolution).

Table 2.3 summarises the libraries used in the project.

Library	Version	Licence	Purpose
CoreNLP [19]	3.5.0	GNU GPL 2	Parsing, coref. resolution, lemmatisation
OpenNLP	1.5.3	Apache 2.0	Parsing
extJWNL	1.8.0	BSD	WordNet access, synonym finding
GraphStream [20]	1.2	CeCILL-C	Graph visualisation
Guava	18.0	Apache 2.0	Runtime timing, Table
RTED [16]	1.1	GNU Affero GPL	Tree Edit Distance
GraphViz[1] [21]	2.38.0	EPL	Mind map export and visualisation

Table 2.3: Libraries used in the project

[1] Used from Java using `Runtime.getRuntime().exec()`

2.7.3 Graph visualisation

GraphStream offers export into raster formats. Using that for mind map visualisation was, however, abandoned for these reasons:

1. The offered export formats were raster formats only.
2. The layout manager in GraphStream was not flexible enough for the project's use case. It wasn't able to prevent nodes from overlapping.
3. The shape of nodes and their colour was not easily modifiable.

Hence GraphStream's export into GraphViz was used and then GraphViz (`dot`) was used to export the graphs into `pdf`. The `neato` [22] layout was used. It uses the gravitational approach for graph layout computation, where each edge is simulated as a spring, each node as a point with repulsive force and a physical system simulation is run on the graph in order to compute its layout. Moreover, certain other options have been used to tweak the output. For their complete overview see Appendix C.

2.7.4 Environment and other tools

All development, execution and testing has been carried out on my personal laptop running 64-bit version of Windows 7. Eclipse SDK[2] was used for the Java development.

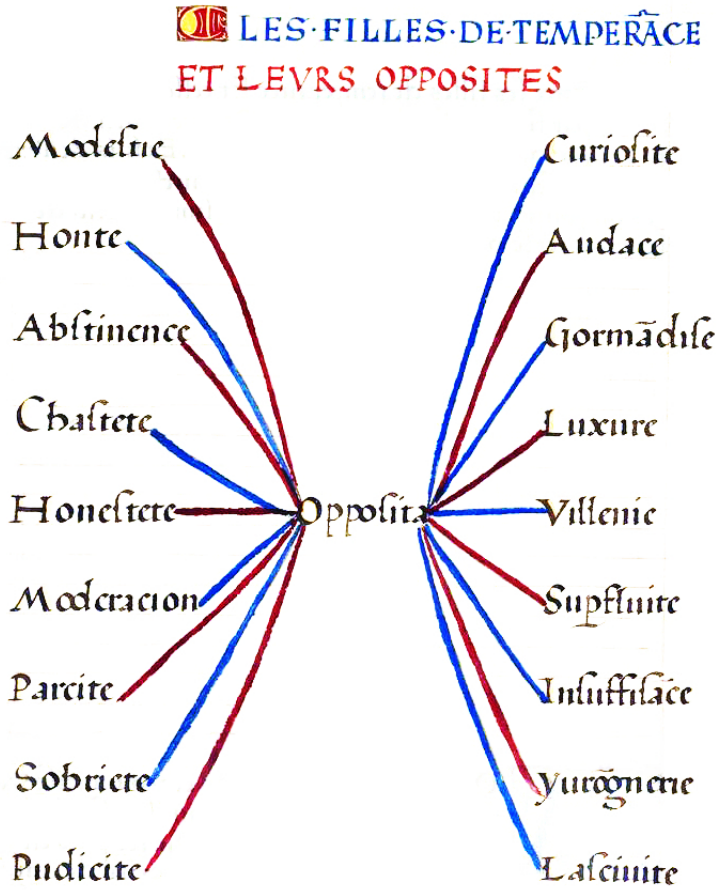
Git was used for the development and the project was therefore backed up after each commit and pushed to a remote private repository hosted on BitBucket (BitBucket was used instead of GitHub since it offers free private repositories). Moreover, the entire project was backed up daily to Dropbox and weekly to a USB flash drive. No data was lost during the development of the project.

2.7.5 Testing

Each stage of the pipeline was separately (unit) tested using short sample texts or graphs where the algorithm could have been carried out by hand and then compared to the result obtained by the system. Each part of the pipeline was tested both separately and also as a part of the pipeline. Moreover, the entire system was re-tested each time any changes were done to the pipeline's components. The outputs of each version of the pipeline were carefully compared to each other and re-evaluated to make sure no regressions are introduced into the system.

The evaluation system also needed testing in this project, as it involved a lot of coding and large tables of numbers were worked with. To verify the results of the evaluation are consistent with the wanted statistical analysis, critical values of the evaluation were calculated by hand and consistency of the results was verified.

[2] www.eclipse.org



Mind map on "Treatise on the virtues of excellence, and how one may acquire them." from 16th century by d'Anguerrande. [1]

Chapter 3

Implementation

This chapter describes the implementation of the project using the tools and techniques described in the previous chapter. The chapter discusses first the system as a whole, to give the reader an intuition on the various components used in the system. Later sections describe the implementation of these components in greater detail.

3.1 System overview

This section gives an intuitive overview of the whole system. This will be done using the following short text about Shakespeare:

"Shakespeare was an English writer. He wrote 38 plays. He composed 154 sonnets. Hamlet, one of his plays, is a tragedy."

This section will guide you through the same stages the system has to go through when processing texts and generating mind maps.

3.1.1 Overview of the pipeline

Coreference resolution [optional] Coreferring phrases are resolved and the original text becomes:

"Shakespeare was an English writer. Shakespeare wrote 38 plays. Shakespeare composed 154 sonnets. Hamlet, one of Shakespeare plays, is a tragedy."

Triplet extraction Triplets (subject, verb, object) are extracted from the main clause of each sentence and saved in a list. The number of triplets in the list is therefore the same as the number of sentences in the text:

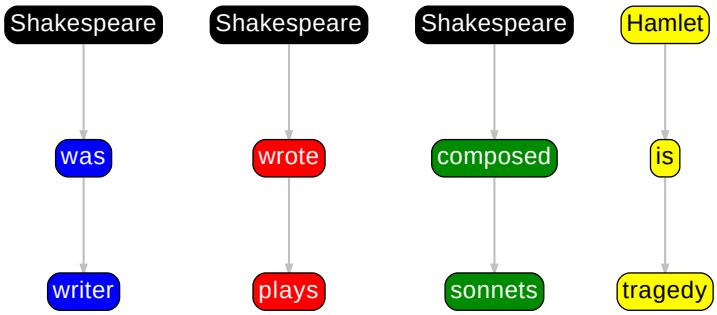


Figure 3.1: Extracted triplets

Different object extraction strategies [optional] There are multiple ways of extracting the object of a sentence. The simplest one is to extract only a single object. More advanced strategies involve extracting multiple objects, object attributes (e.g. "38 plays" would be extracted instead of "plays") or even multiple objects with attributes. An object's *attribute* is a word decorating it (such as an adjective, numeral or adverb) See Section 3.3.1 for a more detailed definition and implementation details. A demonstration of extraction of an object with attributes is in the Figure 3.2 below:

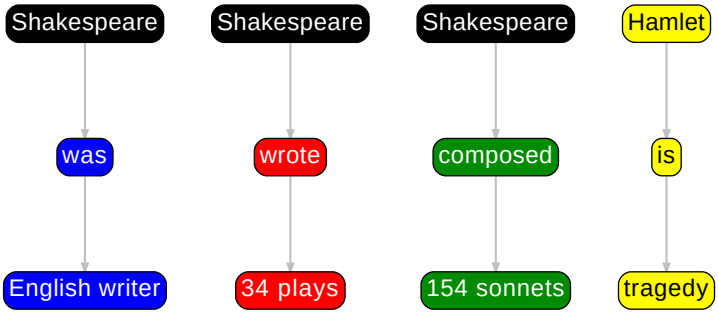


Figure 3.2: Extracted triplets containing objects with attributes

Synonym finding [optional] Sets of synonyms for all verbs are found. If two sets of synonyms for two verbs intersect, the intersecting element is used to replace them both. However, this is done only in the case if the two verbs share the subject. In the example, "wrote" and "composed" have a common synonym "wrote", hence the triplets would become:

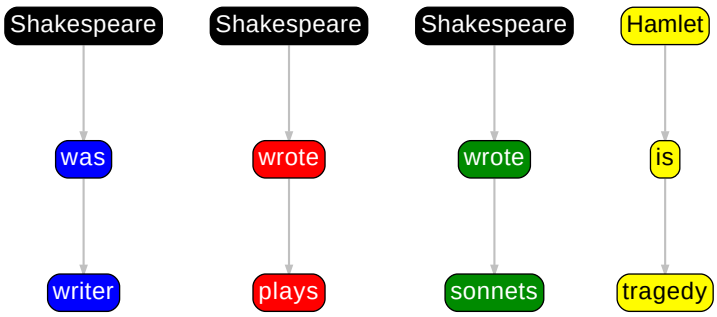


Figure 3.3: Extracted triplets with synonyms found

Mind map building All subject and verb nodes are collapsed together if they share the same label. Hence the three "Shakespeare" nodes collapse into a single node, same happens with the two "wrote" nodes:

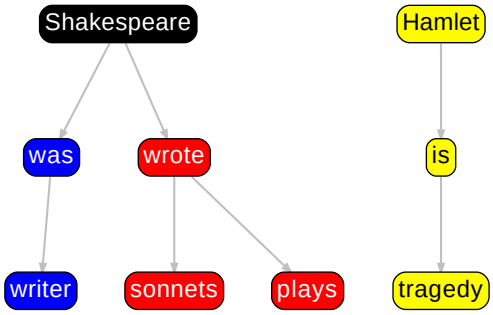


Figure 3.4: Triplet forest

Deforestation Now the mind map is not necessarily a single-rooted tree (which mind map has to be) but rather a forest of trees. In order to change it into a tree, the main topic of the mind map is identified using huge corpus frequency comparison (see Sections 3.2.4 and 3.3.4) and then all remaining lone trees ("Hamlet is tragedy" in this case) are connected to it:

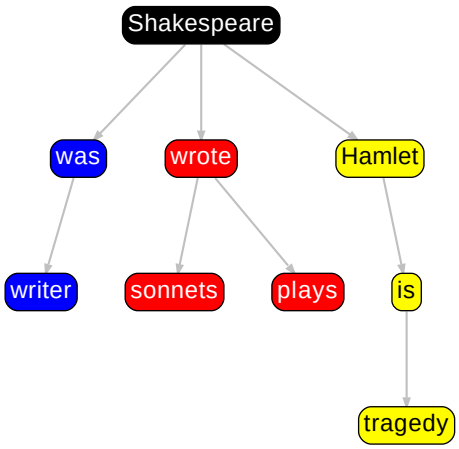


Figure 3.5: The final mind map

3.1.2 Data structures

Multiple data structures were used in the project:

`Essay` Data structure that holds an essay together with various information such as whether its coreferences have been resolved. Each essay, once loaded, is wrapped in this class for convenience and clean OOP design.

`MindMap` Data structure for storing and building a mind map. It uses the Builder pattern to build the mind map from plain-text essay into the final graph. For instance the following code snippet demonstrates the convenience and elegance of the Builder pattern: It resolves

coreferences, extracts triplets (extracts only single non-attributed objects), resolves synonyms, builds the mind map and then saves it as a pdf.

```
MindMap mm = new MindMap(...);
mm.resolveCoreferences()
  .extractTriplets(new PlainObjectExtractor())
  .resolveSynonyms()
  .build(debugStream)
  .save(outputDir, debugStream, new GraphGVandPDFExporter());
```

HumanMindMap Data structure similar to the MindMap (both extend AbstractMindMap), but it is used to store mind maps obtained from human participants. It provides methods for loading the mind maps from text files obtained from the participants (stored in a custom tab-indented tree format), methods for evaluating the mind map against computer-generated mind maps as well as methods for exporting the mind map into a pdf.

MultiObjectTriplet Data structure for triplets. Subject and verb are stored only as strings. However, object is stored as a Set of strings to enable extraction of multiple objects. If object is extracted including its attributes, the attributes are saved in the string together with the object.

WordAttributes Data structure for word attributes – e.g. the word "cat" in the phrase "a black cat with soft fur" has pre-attributes "a black" and post-attributes "with soft fur". Attributes can be either in front of the word of behind it and this data structure enables the differentiation between pre and post attributes.

WordNumberTuple The project needs to store pairs of word and a number (double) quite often, for instance when doing word histograms or when comparing words by their frequency. Another benefit of this class is it implements the Comparator interface on the word count. This is very convenient when words need to be sorted by their frequencies.

Evaluation, GlobalEvaluation These classes store the results of the evaluation and provide methods for computing statistics. They also provide methods for export into LaTeX tables or csv files.

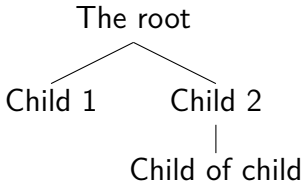
3.1.3 Custom mind map file format

A custom file format for storing mind maps (trees) was created with focus on being easily creatable by human participants and parsable (using stack-based parser). The file format is defined as follows:

1. The root of the tree is on the first line, unindented.
2. Children of a node n on level l are stored on lines directly below n, indented by $l + 1$ tab characters.
3. Lines containing only blank space characters (spaces, tabs) are ignored.

An example of such a tree is below:

```
The root
    Child 1
    Child 2
        Child of child
```



3.2 Bag of words extractors

Bag of words (BOW) extractors have been implemented partially to assist the mind map pipeline (e.g. keyword extractor is used to find the root of the mind map) and also to serve as a baseline in the evaluation. Each generated mind map is actually evaluated against human participants' mind maps using two methods:

1. Using BOW approach. This approach completely ignores any structure of the mind map and compares two mind maps only as two lists of words. In this case the 4 extracted bags of words are evaluated together with the 16 mind maps.
2. Using Tree Edit Distance. **BOW extractors are not evaluated in this case**, since they lack any structure.

Four BOW extractors have been implemented, each using different approach to sort the words by their importance. All four extractors extend an abstract class which defines an abstract method `List<String> getBagOfWords(String text)`.

3.2.1 BOW by frequency

The first implementation extracts and sorts words in the given text according to their count (frequency) and returns top n words which occur in the text most often. Because of the nature of human languages, the words extracted by this extractor are commonly used words, such as "the", "of", "and", "to", "in", etc. – these are the function words that have little lexical (or ambiguous) meaning and therefore do not have any special significance in the context of the given text.

Although this extractor doesn't extract words that humans would consider important in the given context, it servers as the baseline to demonstrate that the other (more advanced) extractors outperform it.The pseudo-code for the extraction algorithm is below:

Algorithm 1 BOW by frequency pseudo-code

```
procedure GetBagOfWords(String essay)
    words ← split the essay by spaces to get a list of string words
    wordCounts ← hash map with key = word, value = count
    for String w : words do
        if w ∉ wordCounts then
            add w into wordCounts with count = 1
        else
            increment w's count by 1
    return words from wordCounts sorted in descending order by count
```

3.2.2 BOW by frequency with common words elimination

The second BOW extractor uses the previous extractor to obtain the list of the most frequent words sorted in descending order by their count. Then it goes over this list and discards all words that are in the list of the most frequent 500 English words (see Appendix A). The reason for this is to eliminate words which are common in all texts. The words that are left are words that are not common English words, yet they have been used in the text. Therefore they must have certain importance in the context of the text:

Algorithm 2 BOW by frequency with common words elimination

```
procedure GetBagOfWords(String essay)
    words ← get BOW from essay as defined in Algorithm 1
    common ← a list of most common 500 English words
    bow ← empty list
    for String w : words do
        if w ∉ common then
            add w to bow
    return bow
```

This implementation extracts more reasonable words, however, it has problems with essays that are about common things. Say an essay on topics like "time", "year" or "day". All these words are in the most common 100 English words, yet they are also words which could surely be the main topic of an essay. This is refined by the next BOW extractor.

3.2.3 BOW by frequency with function words elimination

This BOW extractor is algorithmically similar to the previous one, but uses a list of function words (see Appendix B) rather than list of most common words. Function words are words that have little lexical meaning or have ambiguous meaning, but instead serve to express grammatical relationships with other words within a sentence. [23]

This deals with the problem of the previous BOW extractor which would fail to spot certain important words in the essay if they were present in the list of most common words.

3.2.4 Keyword extraction

This extractor uses keyword extraction algorithm as defined by Kilgarriff [24]. This algorithm compares frequency of a word in the text and in a large corpus. Say there is an article about Newton. The word "Newton" would be mentioned in the article often, with frequency f_{txt}. However, in the large corpus, "Newton" would not be such a frequent word and it would have frequency f_{corp}.

The ratio $R = f_{txt}/f_{corp}$ can serve as very good indicator whether a **word is in the text more often than it statistically should be**. And if that is the case, it is a good candidate to be a keyword in the given text.

This approach has, however, two problems which have been addressed:

1. If the word in the text is not present in the large corpus – i.e. $f_{corp} = 0$, then R is undefined.
2. If the word in text is very rare in the large corpus, it will have very high R. This is not what we want, as we don't want to extract *rare* words, but rather keywords.

Both problems are addressed by *smoothing* – adding a constant c to all frequencies f_{corp}, so that

$$R = \frac{f_{txt}}{f_{corp} + c}. \tag{3.1}$$

Any $c > 0$ addresses the first problem. By making c sufficiently large, the other problem is also addressed. This is because as c approaches values of f_{corp}, it starts to dominate the denominator in the expression for R. As that happens, f_{txt} gets more important and therefore words that are more frequent in the given text are getting larger values of R than those, which are very rare in both the text and the large corpus.

To determine the right value for c, frequencies of words in the corpus were analysed. The frequencies are distributed by the Zipf's Law [25]:

Word	Frequency	Rank
the	$7 \cdot 10^{-2}$	1
it	$8 \cdot 10^{-3}$	10
after	$1 \cdot 10^{-3}$	82
how	$8 \cdot 10^{-4}$	100
easy	$1 \cdot 10^{-4}$	1,000
charcoal	$7 \cdot 10^{-6}$	10,000
westergaard	$1 \cdot 10^{-7}$	97,565

Table 3.1: Table of frequencies of various words in the large corpus

A value of $c = 10^{-3}$ was chosen after initial experiments, since the words in the top 80 are function words only, hence we would like them to be eliminated for sure. This is achieved by setting c smaller than their frequencies. We see words like "people" and "man" appearing in the range 85–100, hence the reason for not setting c even smaller. If we had, this extractor would essentially behave in the same way as the one which eliminated the most common English words.

The pseudo-code for the keyword extraction algorithm is below:

Algorithm 3 BOW using keyword extraction

```
procedure GetBagOfWords(String essay)
    wordFreqs ← get tuples (word : frequency) from essay as defined in Algorithm 1
    wordQuots ← empty hash map with key = word, value = frequency quotient R
    for (word : f_txt) in wordFreqs do
        f_corp ← get frequency of word from the corpus
        R ← f_txt / (f_corp + c)
        add (word : R) to wordQuots
    return words from wordQuots sorted by values of R in descending order
```

3.3 The mind map pipeline

The core of the entire project follows, as the mind map extraction pipeline is described here. An overview of the pipeline was presented at the beginning of the chapter in Section 3.1.1.

3.3.1 Triplet extraction algorithm

The triplet extraction algorithm was implemented according to Rusu et all. [26] It traverses the parse tree and searches for subject, predicate and object according to the given heuristic. I have slightly modified the algorithm to improve its results and to add multi-object extraction capabilities.

The input of the main algorithm for triplet extraction is a parse tree. Methods such as `children()`, `getLeaves()`, `siblings()`, `value()` provided by the CoreNLP parse tree for traversal and information retrieval have been used.

Top-level triplet extraction method

The top-level method `extractTriplet()` basically calls three helper methods: `getSubject()`, `getVerb()` and `getObject()`. However, it does some additional processing to make sure the triplet is extracted even in cases the sentence doesn't have standard structure.

Algorithm 4 Triplet extraction

```
1:  procedure ExtractTriplet(Tree sentence, ObjectExtractionStrategy strategy)
2:      children ← sentence.children()
3:      NPSubtrees ← empty list for NP subtrees
4:      VPSubtrees ← empty list for VP subtrees
5:      for p : children do
6:          if p.value() matches S or ROOT then          ▷ Recurse to get inside the sentence
7:              return ExtractTriplet(p, strategy)
8:          if p.value() = NP then                                          ▷ NP subtree found
9:              add p to NPSubtrees
10:         if p.value() = VP then                                          ▷ VP subtree found
11:             add p to VPSubtrees
12:     if NPSubtrees.size() = 0 then          ▷ See Note below for detailed explanation
13:         add all VPSubtrees into NPSubtrees
14:         if VPSubtrees.size = 0 then
15:             search by BFS for VP subtrees and add the first into NPSubtrees
16:     if VPSubtrees.size() = 0 then                ▷ No VP subtree under S, BFS for it
17:         search by BFS for VP subtrees and add the first into VPSubtrees
18:     subject ← getSubject(NPSubtrees)
19:     objects ← empty set
20:     if VPSubtrees.size() > 0 then
21:         for p : VPSubtrees do
22:             verb ← findVerb(p, sentence)
23:             if verb ≠ null then                  ▷ Search for object(s) in siblings of p
24:                 add all objects found using strategy in p's siblings to objects
25:                 break
26:     return (subject, verb, objects)
```

Note We need to deal with the case if no NP subtree has been found inside the S tree. This implies the sentence structure is not S → NP VP and hence the following heuristic should be used:

1. If a VP subtree has been found (but no NP subtree), the subject might be in the subclause near the verb and hence within the VP subtree. sentence. E.g. the sentence *"Going home was an option."* has no direct NP subtree, however, the subject "option" is located in the VP subtree with the verb "was".
2. If no VP subtree has been found either (lines 14–15), breadth-first search for any VP subtree and search there. This is used for sentences with uncommon structure where nor NP, nor VP subtrees can located directly in the S tree.

Subject extraction

Subjects are found in the NP subtree. The subject is found by performing breadth-first search and selecting the first descendant of NP which is a noun (i.e. NN, NNP, NNPS, NNS).

However, to improve the efficiency of the algorithm, I made two slight modification to the version provided by Rusu:

- If no noun is found in the `NP` subtree, the algorithm searches for pronoun subject.
- The method for subject extraction takes only a single `NP` subtree as the argument in Rusu's paper. In my implementation it gets a list of all found `NP` subtrees and returns the first matching subject. This has been added to deal with cases, where the subject is not located within the first found `NP` subtree. E.g. the sentence *"100 or even more people were there."* would have both "100" and "even more people" in `NP` clauses, but this will make the algorithm extract "people" rather than the "100", as "100" has POS tag `CC` which is not considered as a subject by the algorithm.

Algorithm 5 Subject extraction

```
procedure GetSubject(Tree[] NPSubtree)
    for p : NPSubtrees do                          ▷ First search for noun subjects
        subject ← find first noun by BFS in p
        if subject ≠ null then
            return subject
    for p : NPSubtrees do                          ▷ No noun subject, search for pronoun subjects
        subject ← find first pronoun by BFS in p
        if subject ≠ null then
            return subject
    return null
```

Verb extraction

The verbs (predicates) of the sentence are found in the `VP` subtree. The **deepest** verb descendant in the `VP` subtree is the predicate of the sentence. Verbs we are searching for have these POS tags: `VB`, `VBD`, `VBG`, `VBN`, `VBP`, `VBZ`.

Since in this case the iteration over all found `VP` subtrees is done already in the method `extractTriplet()` in order to search for verb as well as the object in the same `VP` subtree, the `getVerb()` method takes only a single tree as an argument.

Algorithm 6 Verb extraction

```
procedure GetVerb(Tree VPSubtree)
    verbs ← BFS for all verbs in the VPSubtree
    if verbs.size() = 0 then
        return null
                ▷ Since BFS was used, this will be the deepest and the rightmost verb
    return the last verb in verbs
```

Object extraction

Object extraction is slightly more tricky than extracting the subject and the verb. This is partially due to its nature, but also due to the nature of mind maps, where lot of information is actually stored in the leaves – i.e. objects in this case.

In order to face these issues, strategy pattern was used for object extractors. There are four object extractors in the system:

1. Plain object extractor – extracts a single, undecorated object according to Rusu.
2. Object with attributes extractor – extracts object with attributes
3. Multiple plain objects extractor – extracts multiple plain objects. E.g. the sentence "*He had a blue **coat**, yellow **boots** and a long brown **beard**.*" has three objects: *coat*, *boots* and *beard* and all three of them should be extracted.
4. Multiple objects with attributes extractor – extracts multiple objects, each decorated by its attributes. In the above case the following should be extracted: blue coat, yellow boots, long brown beard.

Each object extraction strategy implements the method defined in the abstract class for object extractors: `Set<String> getObjectPhrases(Tree tree, Tree root)`, which is shown below for the plain object extractor.

I have implemented one further refinement for extraction of objects which include numbers. The algorithm by Rusu doesn't extract such objects. However, that leads to not extracting the object in sentences like "He was born in 1968".

To overcome this problem, if Rusu's algorithm terminates with no objects, then the method `searchForNumeralObject(Tree tree)` is invoked. This method searches the given tree by BFS and returns the first node having POS tag `CD`.

Algorithm 7 Single plain object extraction

```
procedure GetObject(Tree VPSubtree)
    toBeSearched ← {VPSubtree}
    while toBeSearched ≠ ∅ do
        p ← head(toBeSearched)
        if p is of type NP or PP then
            object ← find first noun found by BFS in p
        else if p is of type ADJ then
            object ← find first adjective found by BFS in p
        else
            add children of p to toBeSearched                ▷ Nothing found, recurse
    if object = null then
        object ← SearchForNumeralObject(VPSubtree)
    return object
```

Other object extractors

The multiple object extraction algorithm differs from the plain one only by returning all matching nodes, not only the first one. Versions of these two algorithms that include attributes call the `extractAttributes(Tree object, Tree root)` method before returning. Details on attribute extraction are provided below.

Attributes extraction

Attributes (modifiers) decorate objects. The attribute extraction algorithm chooses different strategy according to the POS tag of the word – e.g. attributes for nouns are mostly adjectives, for verbs adverbs, etc.

Again, I have refined the algorithm by Rusu in two ways:

1. When attributes are extracted the algorithm determines whether they lie in *front* of the object or behind it. This is done to enable proper phrase forming in the resulting mind map – the attributes found in front of the word should be in front of it in the mind map as well and the same goes for the attributes behind the word.
2. If attributes are not found in the word's sibling trees, the entire uncle tree is returned if it satisfies certain conditions. This leads to very long sentences being extracted which doesn't correspond to the nature of the mind map. Hence I have used a heuristic that allows the extraction of uncles only if the number of attributes found in sibling trees is not greater than one.

Algorithm 8 Attributes extraction

```
procedure GetAttributes(Tree word, Tree root)
    wordPos ← node number in root
    preAttrs ← empty list
    postAttrs ← empty list
    if word is adjective then
        attrs ← find all RB siblings
    else if word is noun then
        attrs ← find all siblings matching (DT|PRP$|POS|JJ|CD|ADJP|QP|NP)
    else if word is verb then
        attrs ← find all ADVP siblings
    add attrs into preAttrs or postAttrs according to their node numbers
    if size of preAttrs + postAttrs > 1 then
        return (preAttrs, postAttrs)
                                        ▷ No attributes found in siblings, get uncles
    uncles ← all siblings of parent of the word
    if word is noun then
        if uncle in uncles matches PP, add all uncle's leaves into attrs
    else if word is verb then
        if uncle in uncles matches "VB.*", add all uncle's leaves into attrs
    add attrs into preAttrs or postAttrs according to their node numbers
    return (preAttrs, postAttrs)
```

3.3.2 Coreference resolution

The coreference resolver sits on top of the coreference resolver provided by CoreNLP. If selected to run, it processes the text before the triplet extracting algorithm is run and replaces all coreferring mentions by the most representative mention. However, I have created my own algorithm for finding the most representative mention in the coreference chain and also introduced heuristic for coreference replacements to prevent the algorithm from replacing long phrases and hence loosing information.

CoreNLP provides method which returns a list of `CorefChains`. `CorefChain` is a set of mentions (i.e. coreferring phrases) which are essentially pointers into the given text. `CorefChain`

contains also the most representative (best) mention, which is a mention that most accurately describes the mentions in the chain.

The most representative mention selection algorithm

The problem with the most representative mention is that it can consist of multiple words. For instance, in an essay about Isaac Newton, the coreference set would look something like: `["Newton", "Sir Isaac Newton", "Isaac Newton", "he", "his", "Sir", "fellow of Trinity college", "most influential physicist"]`. The most representative mention would be "Sir Isaac Newton". However, if all mentions would get replaced by "Sir Isaac Newton", "Sir" would be then extracted as the subject of those sentences, which is clearly undesirable. Moreover, as one of the main principles of mind map design is brevity, we would rather prefer "Newton" to "Sir Isaac Newton" to be extracted.

Therefore, not to complicate the triplet extraction algorithm, I have decided not to use the most representative mention provided by CoreNLP, but create my own algorithm for determining the most representative mention if the most representative mention provided by CoreNLP is not single-worded:

Algorithm 9 The most representative mention selection algorithm

```
1: procedure GetRepresentativeMention(CorefChain chain)
2:     represent ← the most representative mention in chain according to CoreNLP
3:     if represent has more than 1 word then
4:         mentions ← list of mentions in the chain
5:         sort mentions by their uniqueness in a large corpus, see Section 3.2.4
6:         represent ← first single word in sorted mentions, leave original if none found
7:     return represent
```

The sort by uniqueness is used since the most unique single word in the coreference chain bears usually the most information and hence selecting it as the most representative usually doesn't lead to information loss. The algorithm would return "Newton" as the most representative mention, as it is a single word and it is more rare than "He" or "Sir".

Coreference replacement

Once coreference chains and their most representative mentions are found, some of the mentions will get replaced by the most representative mention. However, this has to be done carefully to prevent information loss, as demonstrated by the following example.

Say we are performing the coreference resolution on the text: "*Harry Potter is a wizard. He is called The Boy Who Lived. Harry is brave. Harry's owl is called Hedwig.*" The coreference chain will be: `["Harry Potter", "He", "The Boy Who Lived", "Harry", "Harry's"]`. As stated above, "Harry" will be chosen as the most representative mention.

However, the "The Boy Who Lived" and neither "Harry's" should get replaced by "Harry", as in the first case information would be lost and in the second case the possessive form (Harry**'s**) would be lost.

In order to deal with these two problems, I have introduced the following heuristic into the algorithm to determine which mentions should be replaced by the most representative mention M. The heuristic consists of three rules:

1. Replace mentions that consist of a single word by M. This deals properly with the case of pronouns. E.g. "he" $\rightarrow$ "Harry".
2. Don't replace mentions that contain possessive form of M. E.g. "Harry's" $\not\rightarrow$ "Harry".
3. Don't replace multi-word mentions, unless they directly contain M and rule 2 is not violated. E.g. "Harry Potter" $\rightarrow$ "Harry" but "The Boy Who Lived" $\not\rightarrow$ "Harry".

The algorithm guided by the three rules then processes all mentions and replaces them accordingly by the most representative mention.

3.3.3 Synonym finding and grouping using WordNet

To reduce the number of verb nodes in the mind map and to group several branches together, WordNet was used to find verb synonyms and group them (see pipeline overview in Section 3.1.1 for a graphical demonstration). The synonym finding is done only on verbs, as subjects are grouped by using coreference resolution and objects should not be grouped, as that could introduce diamond-shaped structures in the resulting mind map as shown in Figure 3.6b.

(a) Two objects as separate children (b) Synonym objects grouped together

The algorithm for the verb grouping finds a set of synonyms for each verb and then searches for intersections in these sets. If there is an intersection, both verbs will be replaced by that intersecting verb:

Algorithm 10 Verb synonym search and replace

```
procedure ResolveSynonyms(List of triplets)
    verbSynSets ← empty list of verb synonym sets
    for t : triplets do
        add synonym set from WordNet of t.verb into verbSynSets
    for i = 1, ..., |verbSynSets| do
        for j = i + 1, ..., |verbSynSets| do
            if verbSynSets[i] ∩ verbSynSets[j] ≠ ∅ then
                replace triplets[i].verb by the first verb in the intersection of their synsets
                replace triplets[j].verb by the first verb in the intersection of their synsets
    return triplets
```

This algorithm clearly has $O(n^2m)$ complexity (where n is the number of verbs in the essay, m is the average size of the synonym set). But since the number of verbs in the mind map was hardly greater than 50 and so was the synonym set, this never caused performance issues.

3.3.4 Building the mind map from triplets

Building the mind map from triplets consists of two main tasks:

1. **Grouping nodes** – Grouping together subjects, verbs and objects which have the same label. However, great care has to be exercised not to introduce cycles or diamonds.
2. **Deforestation** – The result of grouping is a set of trees, i.e. a forest. The forest needs to be converted into a single tree. In order to do that, the main topic of the mind map has to be found and then the remaining trees have to be connected to it.

Node grouping

The node grouping algorithm linearly processes all triplets and attempts to add the subject, verb, object(s) and edges between them. If a node with the given label is already present in the graph, the algorithm deals with it according to the type of the node being added.

Note that all edges are directed and that each node has a label (the text displayed to a user) and a name (internal representation that is used to determine if a node is in a graph). A label of an object node is for instance "dog" and a name of a node is "dog_OBJ". The algorithm below deals with names rather than labels, since they provide information about the type (subject, verb, object) of each node.

Algorithm 11 Node grouping

```
procedure GroupNodes(List of triplets, Graph G)
    for t : triplets do
        if t.subject ∉ G.nodes then
            add t.subject into G and set its name to label+"_SUBJ"
        if t.verb ∉ G.nodes then
            add t.verb into G and set its name to label+"_VERB"
        if (t.subject, t.verb) ∉ G.edges then
            add edge (t.subject, t.verb) into G
        for obj : t.objects do
            if obj ∈ G.nodes ∧ (t.verb, obj) ∈ G.edges then
                continue                    ▷ The verb already has such object as its child
            UID ← 0
            while obj + UID ∉ G.nodes do                    ▷ See Note below
                UID ← UID + 1
            add obj into G and set its name to label+"_OBJ_"+UID
            add edge (t.verb, obj + UID) into G
    return G
```

Note The unique ID (UID) is used to make sure that two different verbs won't share an object with the same label. Say there are two verbs in the graph v_1 and v_2. Verb v_1 has object o_1. Now we want to add object o_2 with the property $o_2.label = o_1.label$. But since they are both objects, it will also be the case that $o_2.name = o_1.name$.

However, we would like to create o_2 as a child of v_2, but not v_1. Hence a unique ID is appended to the name of the object and then it is added as a new child of v_2 without colliding in name with o_1, as in Figure 3.7b. The search for the unique ID is done by the while loop.

(a) Undesirable diamond shape (b) The correct way of adding an object

Deforestation

At this point the graph G is a forest of trees. The following algorithm processes the forest of trees and returns a single tree, rooted at the main topic of the mind map. Note that in-degree of a node is the number of edges pointing to that node:

Algorithm 12 Deforestation

```
procedure Deforest(Graph G)
    roots ← set of nodes with in-degree = 0
    keywords ← list of keywords sorted by their uniqueness (see Section 3.2.4)
    root ← node from roots which comes first in keywords
    for Node n : roots do                ▷ Link roots of all the trees to the main root
        if n ≠ root then
            add edge (root, n) into G
    return G
```

After this stage (assuming there were no cycles in the graph before this procedure) the graph is a tree and hence a valid mind map. See the transition from Figure 3.4 to Figure 3.5 for the visualisation of this algorithm.

3.3.5 Exporting and visualising the mind map

GraphStream, the library that was used to store graphs, offers export into multiple formats, including `png`, `gv`, `tikZ` and `svg`. After initial experiments, export into GraphViz format was chosen as the one giving the most visually pleasing results. For more details on the export and rendering see Appendix C.

The layout computing algorithm (force-directed graph drawing) is rather interesting and worth mentioning. The way mind maps are usually drawn dictates a radial tree layout. Edge crossing should not be present, as the mind map is a tree.

In order to do that, the `neato` layout engine [22] was used. It assigns a repulsive force to each node and attractive force acting on both ends to each edge, i.e. each edge is simulated as a spring. Then a simulation is run and when the system stabilises, the properties of the system cause exactly what is needed of the mind map layout:

- Nodes do not overlap, as they repulse each other.
- Nodes connected by edges are close to each other as they are attracted by a spring that is between them.
- The root of the tree is in the middle, as that is the only way each branch can be as far as possible from every other branch of the tree.

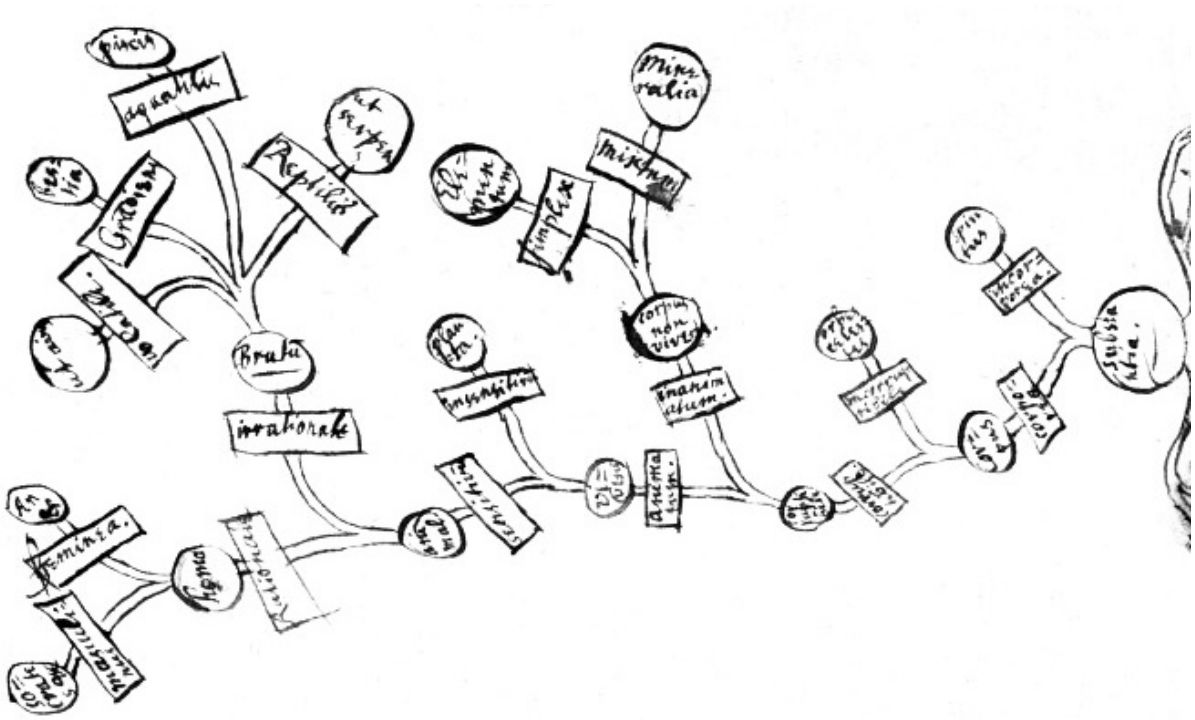
The Appendix C describes the template that has been created to style the resulting graph.

Colouring branches

To make the mind map visually pleasing, every one of n branches coming out of the root has a different colour chosen so that the n colours are visually pleasing together.

Since implementing a general algorithm for selecting n distinct visually pleasing colours is surprisingly difficult, a pragmatic approach has been taken – an array of 16 colours that satisfy the requirements was carefully hand-crafted. Each main branch i has therefore colour $C_{i \bmod 16}$. The colours are depicted below.

Figure 3.8: 16 colours used to colour main branches



One of the early Cambridge mind maps drawn by Isaac Newton. [1]

Chapter 4

Evaluation

This chapter describes the two methods – *Bag of Words Distance (BOW)* and *Tree Edit Distance (TED)* – that were used to evaluate the system against humans. The results of the two methods are compared and discussed. Algorithms are evaluated on per-essay and per-person basis. The BOW metric was used to show that even the worst algorithms that take into account the structure of the document outperform the BOW extracting algorithms. The TED was used to show statistical indistinguishability of the computer-generated mind maps with the ones created by humans.

4.1 Implementation of the evaluation system

As mentioned earlier, this project is interesting also due to the technical challenges presented by its evaluation. The evaluation was *fully automated* and all the LaTeX tables in the dissertation were generated by the system itself. The graphs were generated from raw `csv` data by LaTeX.

The evaluation system creates an `Evaluation` for every algorithm running on every essay. `Evaluation` is essentially a table of distances between the algorithm and all human participant's mind maps for the given essay. Since the evaluation had

- 8 essays,
- 4 BOW extracting algorithms, evaluated using only the BOW metric and
- 16 mind map extracting algorithms, evaluated using both BOW and TED metric,

the number of `Evaluations` was $8 \times (16 \times 2 + 4) = 288$.

The `Evaluation` object provides methods for statistical analysis of the results, such as:

- `getAverageDistanceToHumans()`,
- `getHumanAverage()`,
- `getHumanStDev()`, etc.

All `Evaluations` were then grouped in a `GlobalEvaluation`, which produces the LaTeX tables which are used in this chapter and `csv` files which supply data for the graphs used in this chapter as well.

The full automation of the evaluation proved very beneficial, as additional improvements to the system didn't require any manual changes in the Evaluation chapter other than re-compilation.

Moreover, two sets of plots were created for each metric:

- **Per-essay plots:** These plots show how well different algorithms performed on each of the essays compared to humans. This gave some insight into which algorithms might be suitable for which types of essays.
- **Per-person plots:** These plots show how well different algorithms performed compared to each of the human participants. These plots can be used to determine how well different algorithms approximate different human mind mapping strategies.

4.2 Methodology

Mind maps for each of the **8 essays** (see Appendix D) were created by **7 human participants**. The Appendix E contains the instructions that were given to the participants, the Appendix F contains the mind maps that were drawn by the human participants. All the mind maps were included to show the broad variety of structures the participants have created.

The eight texts that were used are described in Table 4.1 below:

	Topic	Words	Source	Comments
1	Computer Laboratory	74	www.cl.cam.ac.uk	Simple text containing lot of multi-object sentences and figures
2	Isaac Newton	118	Simple English Wikipedia	Limited vocabulary and simple sentences
3	Isaac Newton	434	English Wikipedia	Long nested sentences
4	Arthur Dent	196	The Hitchhiker's Guide to the Galaxy	Artistic with metaphors and absurd sentences
5	Winston Smith and the plot of 1984	341	English Wikipedia	Chronological story
6	EDSAC	113	English Wikipedia	A lot of numbers, the main topic is a rare word EDSAC
7	Pink Floyd	528	English Wikipedia	Text with multiple subjects (talks about each member of Pink Floyd)
8	Mind maps	143	English Wikipedia	Text which humans process according to their understanding of what a mind map is

Table 4.1: Eight texts used for the system evaluation

Two metrics were then used to compare similarity of mind maps – the BOW metric and the TED metric:

- **The BOW metric** was used as a baseline to show that the algorithms which take into account the structure as well as keywords outperform those which only extract bags of words.
- **The TED metric** was then used to compare the mind map extraction algorithms and show benefits of certain algorithms on certain essays. Naturally, as this metric requires a tree structure, the BOW extractors were not evaluated using this metric.

The TED metric was chosen to supplement the BOW metric as it captures the notion of "similarity" as perceived by humans. That is, it penalises both differences in the structure (by

having *insert* and *delete* operations) of the mind maps as well as differences in the individual node labels (by having the *rename* operation).

However, a problem with this approach was identified. **The metric would penalise small differences in phrasing humans would consider equivalent.** In order to address this issue, the text of each node was lemmatised before using the TED metric. See Section 4.2.1 for details.

4.2.1 Lemmatisation

Lemmatisation is the process of grouping together the different inflected forms of a word so they can be analysed as a single item[1].

As said above, it is important not to penalise human participants for small differences such as using plural or gerund of a verb instead of the verb. In order to prevent this, every mind map is normalised by lemmatising all words in it before comparison is performed.

Therefore, the following two mind maps would be considered equivalent (since humans would consider them equivalent as well):

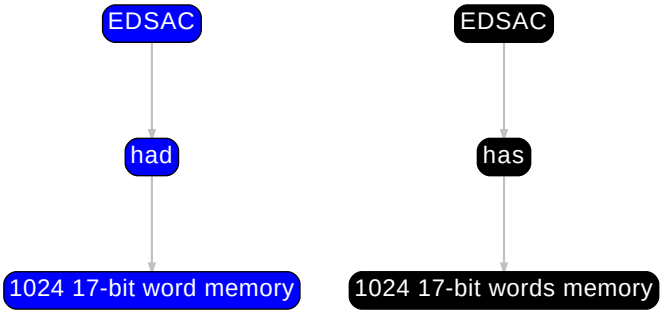


Figure 4.1: Two mind maps which are equivalent under TED metric if lemmatised before comparison

The lemmatisation was performed using the CoreNLP lemmatisation module.

4.2.2 Statistical analysis

Seven human participants were asked to create mind map for each of the essays. The instructions given to participants were deliberately vague, asking them to draw a mind map and a link to Wikipedia article about mind maps[2] was given to explain what a mind map is. The obtained mind maps were in the custom tree file format as described in Section 3.1.3.

For **each essay**, the system generated **16 mind maps** (as per Table 4.2 below, the total number of combinations is $2 \times 4 \times 2 = 16$) and **4 bags of words**.

[1] Collins English Dictionary, entry for "lemmatise"
[2] http://en.wikipedia.org/wiki/Mind_map

Tool name	*Pronoun resolver*	*Object extractor*	*Synonym finder*
Options	Use Don't use	Plain single Attributed single Plain multiple Attributed multiple	Use Don't use
No. of options	2	4	2

Table 4.2: Types of mind maps that have been generated

For each bag of words and each mind map generated for an essay, the distances to all human participants' mind maps were calculated:

1. All 20 (16 mind maps and 4 bags of words) were compared to the human participants' mind maps as a lists of words, completely ignoring their structure (using the BOW metric).
2. The 16 mind maps were compared also using the TED approach, where the their structure was taken into account as well.

To check **statistical indistinguishability from humans**, the average distance to all human participants' mind maps D_A was calculated for each algorithm. If D_A lied within the average human mind maps distance $\pm$ standard deviation, then the algorithm was declared to be statistically indistinguishable from humans.

4.2.3 Explanation of abbreviations

The Evaluation chapter uses abbreviations for the algorithm names in order to fit them in the tables and charts:

The object extraction algorithms have the following abbreviations:

- `obj1` – Single plain object extraction algorithm
- `obj1WAtts` – Single object with attributes extraction algorithm
- `objN` – Multiple plain objects extraction algorithm
- `objNWAtts` – Multiple objects with attributes extraction algorithm

In case synonyms have been grouped, `grpSyn` is added to the name. In case coreferences have been resolved, `prnRes` is added to the name.

4.3 BOW approach evaluation

This section gives the results of the evaluation using the BOW approach as described above and in the Preparation chapter. The BOW approach was used to demonstrate how important the structure is when comparing mind maps to each other and how even the most naive implementation of the algorithm that uses the information about the structure outperforms the most advanced BOW extraction algorithm. See Section 2.2 for the definition of the BOW metric. Each BOW extractor extracted 20 words.

4.3.1 Results

The Table 4.3 below shows average BOW distances between all human participants to all the remaining humans and their standard deviations.

Humans	E1	E2	E3	E4	E5	E6	E7	E8
Average distance	0.47	0.69	0.83	0.85	0.90	0.68	0.79	0.76
Std. deviation	0.13	0.06	0.04	0.05	0.04	0.06	0.05	0.06

Table 4.3: Average BOW distances between human participants, 0 = identical, 1 = no common element

Table 4.4 below gives results for the four BOW extraction algorithms and the 16 mind map extraction algorithms. Values that are statistically indistinguishable from humans have yellow background.

	E1	E2	E3	E4	E5	E6	E7	E8	**Avg**
BOW common elim	0.96	0.95	0.98	0.97	0.96	0.97	0.98	0.92	0.96
BOW function elim	0.96	0.96	0.97	0.98	0.96	0.96	0.97	0.92	0.96
BOW keyword	0.96	0.97	0.97	0.97	0.97	0.98	0.97	0.91	0.96
BOW plain	0.97	0.98	0.99	0.98	0.99	0.98	0.98	0.94	0.98
obj1	0.95	0.89	0.95	0.92	0.92	0.85	0.92	0.87	0.91
obj1-grpSyn	0.95	0.89	0.96	0.93	0.93	0.87	0.92	0.88	0.92
obj1-prnRes	0.95	0.88	0.96	0.94	0.90	0.85	0.93	0.87	0.91
obj1-prnRes-grpSyn	0.95	0.88	0.96	0.94	0.91	0.85	0.93	0.88	0.91
obj1WAtts	0.89	0.92	0.95	0.95	0.93	0.85	0.93	0.89	0.91
obj1WAtts-grpSyn	0.89	0.92	0.95	0.95	0.94	0.88	0.93	0.89	0.92
obj1WAtts-prnRes	0.89	0.94	0.96	0.97	0.90	0.86	0.93	0.89	0.92
obj1WAtts-prnRes-grpSyn	0.89	0.94	0.96	0.97	0.91	0.86	0.93	0.89	0.92
objN	0.90	0.80	0.93	0.93	0.93	0.88	0.90	0.86	0.89
objN-grpSyn	0.90	0.80	0.93	0.94	0.94	0.90	0.90	0.87	0.90
objN-prnRes	0.90	0.79	0.93	0.94	0.91	0.88	0.92	0.86	0.89
objN-prnRes-grpSyn	0.90	0.77	0.93	0.95	0.92	0.88	0.92	0.87	0.89
objNWAtts	0.86	0.83	0.93	0.95	0.94	0.84	0.91	0.88	0.89
objNWAtts-grpSyn	0.86	0.83	0.93	0.95	0.95	0.86	0.91	0.88	0.90
objNWAtts-prnRes	0.86	0.82	0.93	0.97	0.92	0.84	0.92	0.88	0.89
objNWAtts-prnRes-grpSyn	0.86	0.81	0.93	0.97	0.92	0.84	0.92	0.88	0.89

Table 4.4: Average BOW distance of various mind map and BOW extraction algorithms to human participants' mind maps

The two plots on the two subsequent pages give per-essay (Figure 4.2) and per-person (Figure 4.3) comparisons of the various algorithms. The per-essay plots are basically only a visualisation of Table 4.4 above, while the per-person plots give some insight into which algorithm is the most similar to different human approaches.

The grey area in the per-essay plots is the interval of human indistinguishability. Note that the grey area is cut-off in the plot for the Essay 1 in order to have the same y range in all the plots. Also note that the y axis has different scale for the Average and the All essays plots.

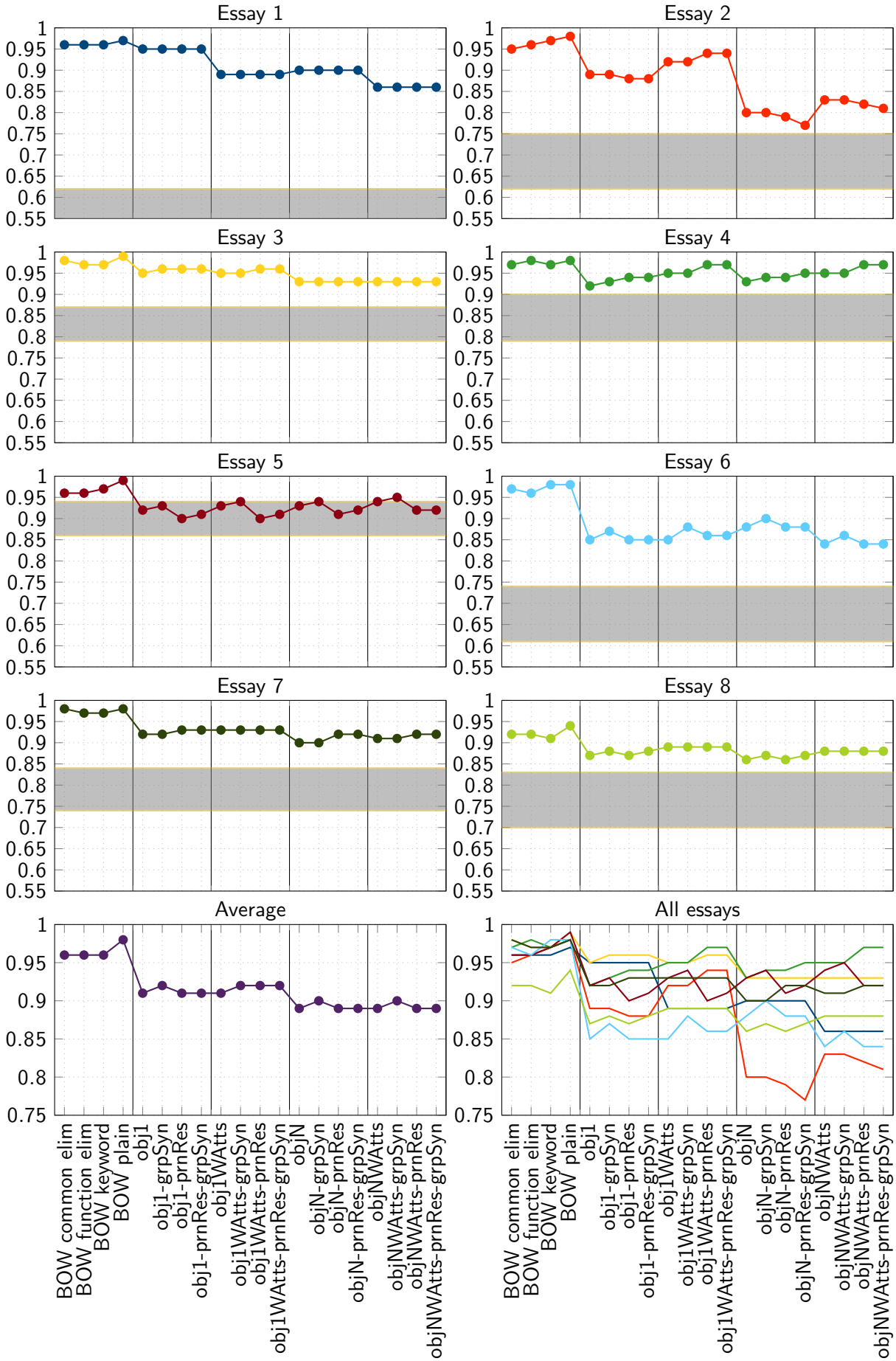


Figure 4.2: BOW per-essay evaluation

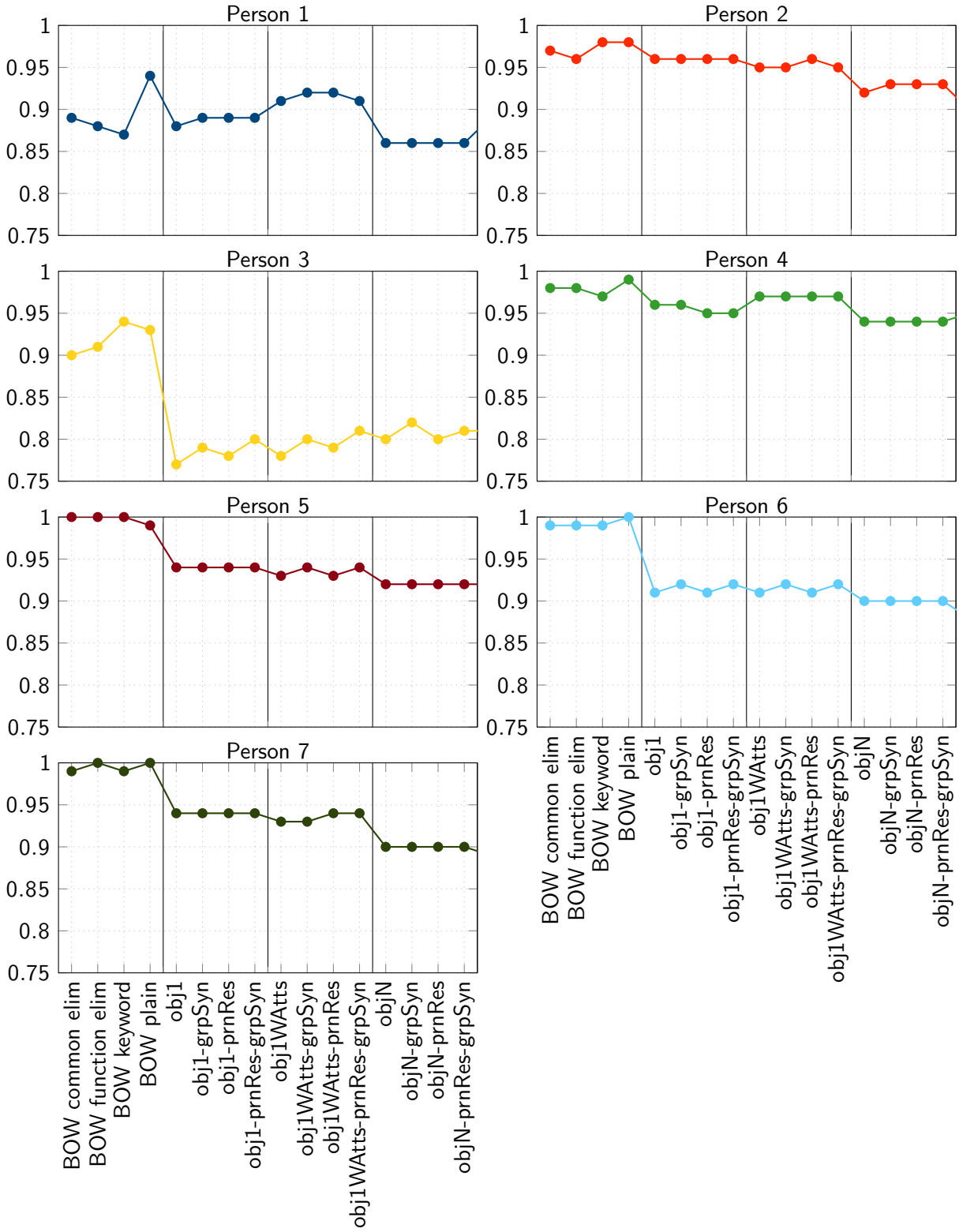


Figure 4.3: BOW per-person evaluation

4.3.2 Discussion

Clearly, by inspecting Table 4.4 or the plots in Figures 4.2 or 4.3, the system doesn't perform well under the BOW metric. The average distances of the system lie within 2 or 3 stdev.

rather than within 1 stdev., with the exception of Essay 5. However, that was caused more by dissimilarities in human participants' mind maps, as this essay was quite hard to create mind map from. Therefore there is only limited amount of information that can be deduced from the data.

However, even the BOW metric revealed that all the algorithms that take structure into account outperform the BOW-only extraction algorithms, i.e. the average distance for BOW extractors was 0.965, while the average for the other algorithms was 0.904 under the BOW metric.

The problems with this metric led to the choice of using another metric to properly evaluate the system. The BOW metric, however, provides intuition for what we want of the TED metric. The problems with the BOW metric were:

1. The BOW metric ignores the structure of the mind map.
2. High scores can be achieved in the BOW metric by simply returning the entire essay as a list of words. This is a similar problem as when using recall as a metric. I.e. if the number of words was different from 20, the results would be better for the BOW extractors. However, 20 was used as it is similar to the average number of nodes extracted by the mind map extraction algorithms. Also, this is why the most naïve BOW extraction algorithm outperformed the remaining three, although it just extracted words like "the" and "to".
3. BOW metric penalises nodes which have the same meaning (i.e. are considered equivalent by humans), but don't contain the same string. Although this is partially solved by lemmatisation, some cases are still not captured. This issue is addressed under the Tree Edit Distance metric by setting the cost of the *rename* operation to lower value of 0.5.

4.4 Tree Edit Distance evaluation

This section gives results of the evaluation using the Tree Edit Distance metric approach as described in the Preparation chapter. Tree Edit Distance evaluation improves on all three problems which have been pointed out in the discussion of the evaluation using BOW metric:

1. The TED metric takes into account the structure of the mind map as well.
2. Achieving high scores under the TED metric is not possible by returning all words in the essay because deletions are penalised as well in this case.
3. As said above, the cost of the *rename* operation has been decreased to 0.5. This makes the cost of two *rename* operations the same as cost of one *insert* or *delete*. This decreases penalisation of nodes containing synonyms or nodes which vary in phrasing, but not meaning (e.g. "two men" and "2 men" or "members of staff" and "staff members").

The reason for choosing 0.5 as the weight for *rename* operation was based on empirical observations. When using weight of value 1 for all the operations, about 2/3 of the operations were *insert* or *delete* operations. That means these operations were changing the actual structure of the mind map and they should be preserved in order to penalise the algorithm correctly for having wrong structure. In order to not make the *rename* operation too cheap so that it would have been cheaper to rename everything to change the structure instead of only dealing with local changes, the value 0.5 was chosen. This way the average distance

decreased by about 15% which means the total cost was still dominated by costs of *insert* and *delete* operations as it should.

Hence every Tree Edit Distance below between mind maps M_1 and M_2 is the minimal number of *insert*, *delete* and *rename* operations that change M_1 into M_2. The distance is then calculated as:

$$TED(M_1, M_2) = |delete| + |insert| + 0.5|rename|, \tag{4.1}$$

where $|\cdot|$ denotes how many times has the given operation been used.

4.4.1 Results

The results are presented in a similar format as in the evaluation using the BOW metric. Note that BOW extraction algorithms are not evaluated in this case as they have no structure and hence can't be evaluated using the TED metric. Table 4.5 below gives average human TED statistics for each essay with their standard deviations. The results are not normalised since normalising TED is not trivial and for the purposes of this evaluation unnecessary.

Humans	E1	E2	E3	E4	E5	E6	E7	E8
Average distance	11.3	17.0	35.1	17.8	35.1	18.2	67.7	26.0
Std. deviation	0.8	0.9	4.9	4.4	11.5	2.7	15.7	3.6

Table 4.5: Average Tree Edit Distances between human participants

The Table 4.6 below contains evaluation of the 16 mind map extraction algorithms. Cells with yellow background are those which are statistically indistinguishable from human participants under the TED metric.

	E1	E2	E3	E4	E5	E6	E7	E8	**Avg**
obj1	13.1	19.8	37.4	20.8	34.9	18.1	101.1	23.0	33.5
obj1-grpSyn	13.4	20.3	36.2	19.0	32.1	17.3	100.0	22.6	32.6
obj1-prnRes	13.1	18.5	35.4	19.3	33.4	17.7	101.5	22.9	32.7
obj1-prnRes-grpSyn	13.1	16.5	34.4	16.9	31.6	17.9	98.0	23.1	31.4
obj1WAtts	13.2	20.1	38.6	21.6	36.6	18.3	107.5	23.6	34.9
obj1WAtts-grpSyn	12.9	20.5	37.6	21.1	34.4	17.0	104.0	23.0	33.8
obj1WAtts-prnRes	13.2	19.3	35.1	19.9	34.9	17.6	106.9	23.4	33.8
obj1WAtts-prnRes-grpSyn	13.0	17.4	34.8	17.4	31.9	18.1	101.6	23.1	32.2
objN	11.8	32.2	69.8	27.9	42.5	30.0	153.0	23.0	48.8
objN-grpSyn	11.9	31.6	68.6	27.3	41.2	25.4	150.9	22.4	47.4
objN-prnRes	11.7	30.7	56.3	25.5	42.6	26.4	149.4	22.1	45.6
objN-prnRes-grpSyn	11.9	27.6	55.4	23.8	40.9	26.6	142.7	22.6	43.9
objNWAtts	11.7	32.2	69.4	29.2	44.8	30.5	157.8	23.0	49.8
objNWAtts-grpSyn	11.8	32.3	69.0	28.4	42.6	24.8	154.9	22.1	48.2
objNWAtts-prnRes	12.6	30.3	55.7	26.4	44.4	27.6	154.9	23.7	47.0
objNWAtts-prnRes-grpSyn	11.6	28.6	55.5	25.1	42.9	27.3	146.8	22.4	45.0

Table 4.6: Average Tree Edit Distances of various mind map extraction algorithms to human participants' mind maps

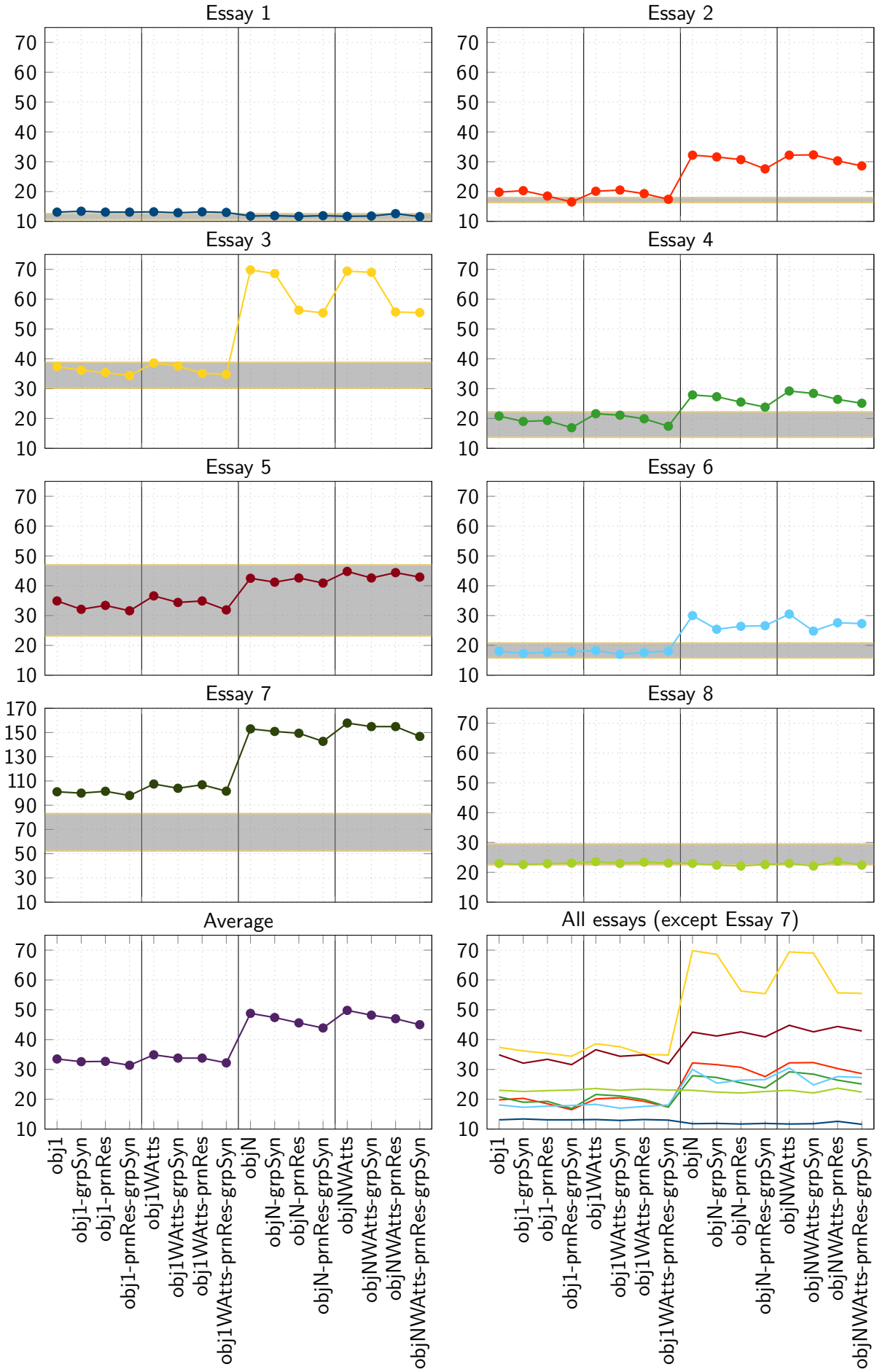


Figure 4.4: TED per-essay evaluation. The grey area denotes again the human-indistinguishable interval.

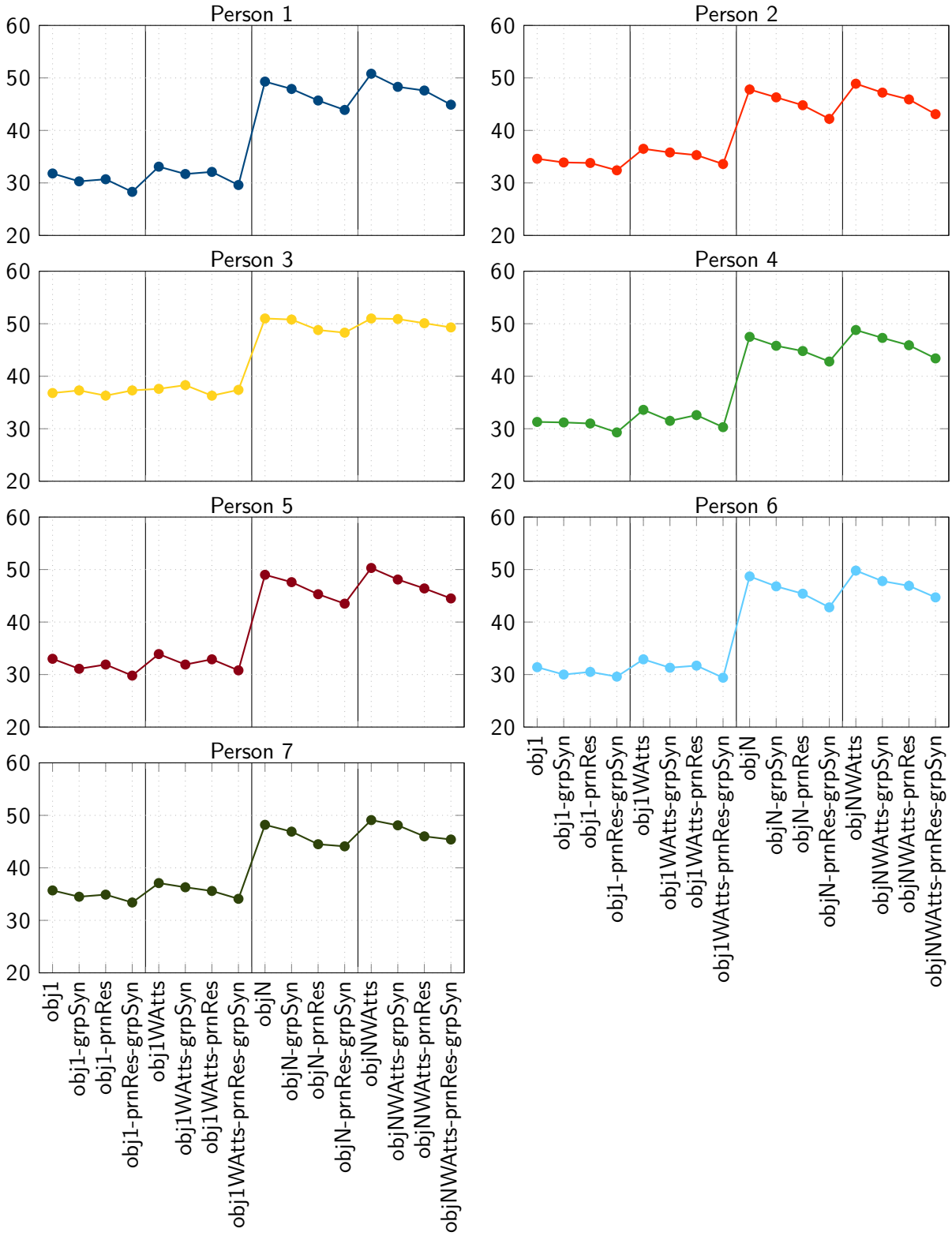


Figure 4.5: TED per-person evaluation

Again, the data is visualised in the per-essay and per-person plots on the two pages above. This time the data gives us more relevant information which will be discussed in the next section.

4.4.2 Discussion

This time more conclusions can be drawn from the data and the plots.

Statistical Indistinguishability

We see that under the TED metric, in 62 cases the algorithms performed in a manner that was statistically indistinguishable from human participants. Since there were $8 \times 16 = 128$ evaluations in total, the algorithms were statistically indistinguishable from human participants in about **48% of cases**.

The following Table summarises performance of each of the algorithms, the second column contains the percentage with which the algorithm was statistically indistinguishable from humans under the TED metric:

Algorithm	**%**
obj1	62.5
obj1-grpSyn	62.5
obj1-prnRes	62.5
obj1-prnRes-grpSyn	**75.0**
obj1WAtts	62.5
obj1WAtts-grpSyn	62.5
obj1WAtts-prnRes	62.5
obj1WAtts-prnRes-grpSyn	**75.0**
objN	37.5
objN-grpSyn	25.0
objN-prnRes	25.0
objN-prnRes-grpSyn	37.5
objNWAtts	37.5
objNWAtts-grpSyn	25.0
objNWAtts-prnRes	37.5
objNWAtts-prnRes-grpSyn	25.0

There are three interesting things to notice here:

- The single object extraction algorithms (with attributes or without) performed the best. This is due to the multiple-object extraction algorithms introducing a lot of irrelevant objects which had to be deleted. This can be also seen from the average plot over all essays. This result is opposite to the result of the BOW metric. This is not surprising as the BOW metric really favours those algorithms that simply extract more words.
- Synonym grouping and coreference resolution had positive impact on the performance of the algorithms. That indicates that problems with the simple algorithms were well spotted and partially solved by these additions. This can be also seen from the average plot over all essays.
- The overall success rate of the single-object extraction algorithms satisfies the acceptance criteria of the project.

Additional observations

Clearly, the Essay 7 had much greater average distance. This is due to its length – the mind maps that were created by both human participants and the system contained more nodes and had more complex structure. Hence the larger distance, as more nodes had to be edited in order to match these mind maps using the TED metric.

It is also important to explain, why the system didn't perform well on essays 2 and 7. Essay 2 was very simple and didn't contain much information. Therefore the variation between human participants' mind maps was small and hence the algorithms had difficulties fitting into that thin range.

In case of Essay 7 it was due to the multi-topical nature of the essay. It was the essay about Pink Floyd and it talked about Pink Floyd's members. While the system wasn't able to spot this, human participants were. And hence their structure was quite radically different from the structure of computer-generated mind maps (i.e. human mind maps created "sub-mind maps" for each of the band's members).

Per-person evaluation discussion

The per-person evaluation enables analysis of similarity of different algorithms to different approaches taken by different humans. We see that single object extraction with pronoun resolution and synonym grouping either with attributes or without performs the best compared to humans. These two algorithms offer almost the same performance, however, by inspecting the mind maps the ones that contain object attributes are subjectively better. So if only one algorithm should be used, I would use the obj1WAttrs-prnRes-grpSyn. We also see that the Person 3 was the most distant from all the algorithms.

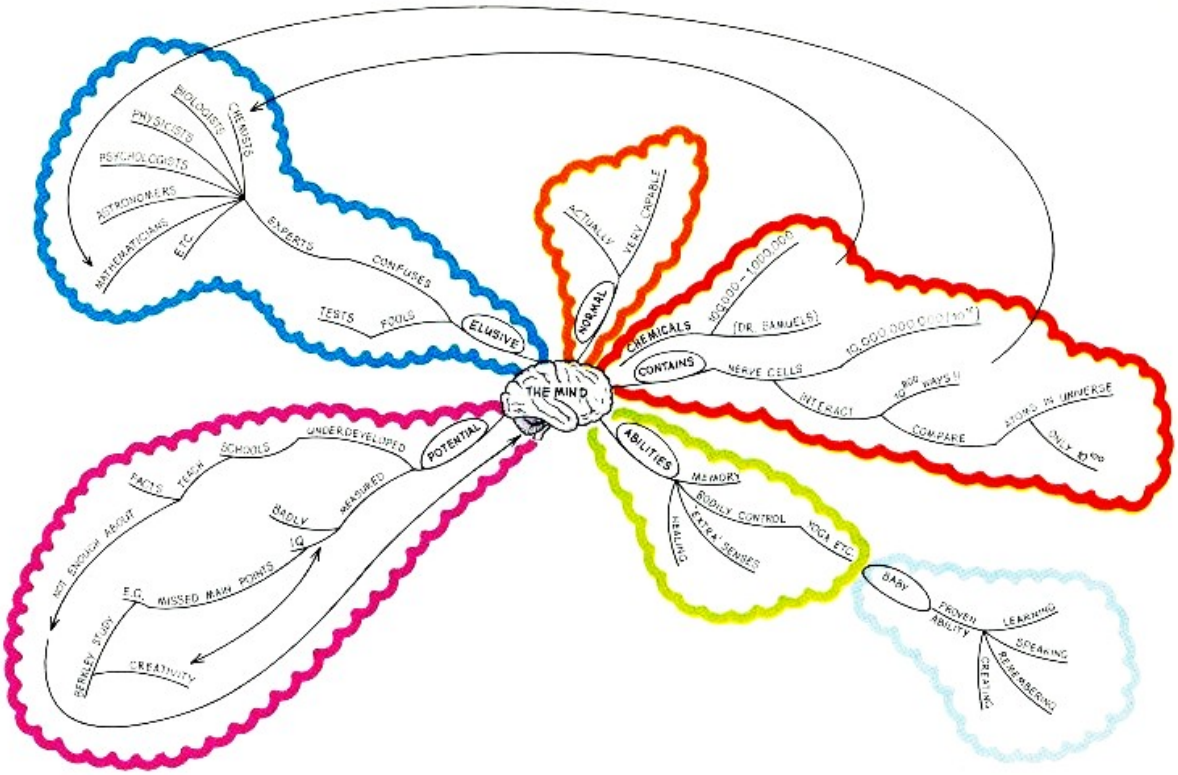
4.5 Alternative evaluation

The two evaluations showed only pure statistical results under the given metrics. The TED metric even showed statistical indistinguishability from humans in 48%. However, it is questionable whether the mind maps generated by the system are really indistinguishable by "common human sense".

In order to truly verify this, the system would have to be evaluated using human judges (different from those who created the mind maps) who would essentially have to perform the Turing Test [27] – i.e. decide which mind maps for a given essay were created by computer and which by humans. However, I decided not to use this as evaluating the system using a distance metric approach was technically more challenging and less time consuming.

4.6 Summary

The system was evaluated using two methods – BOW and TED. The first showed that mind map extractors performed better than BOW extractors. However, the BOW evaluation didn't work well since it favoured systems with high recall. The TED evaluation revealed differences between mind map extractors and showed that synonym grouping and coreference resolution improved the system performance.



The first "official" mind map presented by Tony Buzan in 1974 in BBC TV series called *Use Your Head*. [1]

Chapter 5

Conclusion

I was rather sceptical at the beginning of the project that the mind maps generated by the system would prove to be informative and relevant to the texts they were extracted from. However, as the project went and I refined the system, the results were more and more pleasing. During the project I learned a lot about NLP and how to use major NLP libraries. The project was very enjoyable and because of the nature of NLP there was always something to be improved. I enjoyed inventing *systematic* solutions for the problems in my system.

5.1 Achievements

The project achieved all its goals and even got to implementing some of the extensions. The system was also evaluated using two methods which wasn't expected at the beginning.

The system offers decent performance, processing all the essays and evaluating all the outputs in less than 5 minutes, which makes the system completely process an essay in less than a minute. The evaluation is completely automated and generates the tables and plots that are used in the dissertation.

The system performs very well on short and simple texts and because of its good architecture it can be easily extended.

The entire project has over 6,000 lines of well-documented and tested code and connects together number of interesting libraries, tools and ideas.

5.2 Outputs from the work

The most notable output are the things I have learnt:

- I deepened and broadened the theoretical knowledge I acquired in the Part II NLP course with practical knowledge.
- I created my own Java database of word frequencies based on the Google Books Trillion Word Corpus.
- I learnt how to deal with graphs in Java and how to visualise them using GraphViz.
- I am glad I discovered Guava and learnt how to use it, it is a truly powerful library.

I am going to publish the work as open-source and parts of the work will be included in my Java library called SimpleJava.

5.3 Future work

Since the project is dealing with natural language, there is always something to improve and I would indeed like to continue developing it. A brief summary of possible improvements is below:

- Long text support
- Improvements of the multiple-object extraction
- GUI
- Support for other languages

Bibliography

[1] V. Gee, "Roots of visual mapping." http://www.mind-mapping.org/blog/mapping-history/roots-of-visual-mapping/. [Online; accessed 25-January-2015].

[2] Wikipedia, "Mind map — Wikipedia, The Free Encyclopedia." http://en.wikipedia.org/wiki/Mind_map. [Online; accessed 21-April-2015].

[3] C. Brucks and C. Schommer, "Assembling actor-based mind-maps from text stream," *arXiv preprint arXiv:0810.4616*, 2008.

[4] A. Saelan and A. Purwarianti, "Generating mind map from indonesian text using natural language processing tools," *Procedia Technology*, vol. 11, pp. 1163–1169, 2013.

[5] M. Elhoseiny and A. Elgammal, "English2MindMap: An automated system for mindmap generation from English text.," in *ISM*, pp. 326–331, 2012.

[6] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1st ed., 2000.

[7] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz, "Building a large annotated corpus of English: The Penn Treebank," *COMPUTATIONAL LINGUISTICS*, vol. 19, no. 2, pp. 313–330, 1993.

[8] K. Toutanova and C. D. Manning, "Enriching the knowledge sources used in a maximum entropy part-of-speech tagger," in *Proceedings of the 2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora: Held in Conjunction with the 38th Annual Meeting of the Association for Computational Linguistics - Volume 13*, EMNLP '00, (Stroudsburg, PA, USA), pp. 63–70, Association for Computational Linguistics, 2000.

[9] C. Brontë, *Jane Eyre*. Random House, 1943.

[10] D. Klein and C. D. Manning, "Accurate unlexicalized parsing," in *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, ACL '03, (Stroudsburg, PA, USA), pp. 423–430, Association for Computational Linguistics, 2003.

[11] J. R. R. Tolkien, *The Fellowship of the Ring*. Houghton Mifflin Co., 1993.

[12] K. Raghunathan, H. Lee, S. Rangarajan, N. Chambers, M. Surdeanu, D. Jurafsky, and C. Manning, "A multi-pass sieve for coreference resolution," in *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, (Stroudsburg, PA, USA), pp. 492–501, Association for Computational Linguistics, 2010.

[13] G. A. Miller, "WordNet: A lexical database for English," *Commun. ACM*, vol. 38, pp. 39–41, Nov. 1995.

[14] A. Franz and G. M. T. T. Thorsten Brants, "All our n-gram are be-

long to you." http://googleresearch.blogspot.co.uk/2006/08/all-our-n-gram-are-belong-to-you.html. [Online; accessed 26-January-2015].

[15] P. Norvig, "Natural language corpus data: Beautiful data." http://norvig.com/ngrams/. [Online; accessed 26-January-2015].

[16] M. Pawlik and N. Augsten, "RTED: A robust algorithm for the tree edit distance," *CoRR*, vol. abs/1201.0230, 2012.

[17] B. W. Boehm, "A spiral model of software development and enhancement," *Computer*, vol. 21, pp. 61–72, May 1988.

[18] S. Kuksenko, "Jdk 8: Lambda performance study." http://www.oracle.com/technetwork/java/jvmls2013kuksen-2014088.pdf. [Online; accessed 17-January-2015].

[19] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky, "The Stanford CoreNLP natural language processing toolkit," in *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 55–60, 2014.

[20] Y. Pigné, A. Dutot, F. Guinand, and D. Olivier, "GraphStream: A tool for bridging the gap between complex systems and dynamic graphs," *CoRR*, vol. abs/0803.2093, 2008.

[21] E. R. Gansner and S. C. North, "An open graph visualization system and its applications to software engineering," *SOFTWARE – PRACTICE AND EXPERIENCE*, vol. 30, no. 11, pp. 1203–1233, 2000.

[22] S. C. North, "Drawing graphs with neato." http://www.graphviz.org/pdf/neatoguide.pdf, 2004. [Online; accessed 28-February-2015].

[23] Wikipedia, "Function word — Wikipedia, The Free Encyclopedia." [Online; accessed 02-February-2015].

[24] A. Kilgarriff, "Simple maths for keywords," in *Proceedings of Corpus Linguistics*, 2009.

[25] E. W. Weisstein, "Zipf's Law from MathWorld – A Wolfram Web Resource." [Online; accessed 02-February-2015].

[26] D. Rusu, L. Dali, B. Fortuna, M. Grobelnik, and D. Mladenic, "Triplet extraction from sentences," in *Proceedings of the 10th International Multiconference Information Society-IS*, pp. 8–12, 2007.

[27] A. M. Turing, "Computing machinery and intelligence," *Mind*, pp. 433–460, 1950.

Appendix A

Most common 500 English words

Obtained from http://www.world-english.org/english500.htm.

the	do	work	want	should	children	above	am	cry	object
of	their	part	air	country	begin	ever	remember	dark	decide
to	time	take	well	found	got	red	step	machine	surface
and	if	get	also	answer	walk	list	early	note	deep
a	will	place	play	school	example	though	hold	wait	moon
in	way	made	small	grow	ease	feel	west	plan	island
is	about	live	end	study	paper	talk	ground	figure	foot
it	many	where	put	still	often	bird	interest	star	yet
you	then	after	home	learn	always	soon	reach	box	busy
that	them	back	read	plant	music	body	fast	noun	test
he	would	little	hand	cover	those	dog	five	field	record
was	write	only	port	food	both	family	sing	rest	boat
for	like	round	large	sun	mark	direct	listen	correct	common
on	so	man	spell	four	book	pose	six	able	gold
are	these	year	add	thought	letter	leave	table	pound	possible
with	her	came	even	let	until	song	travel	done	plane
as	long	show	land	keep	mile	measure	less	beauty	age
I	make	every	here	eye	river	state	morning	drive	dry
his	thing	good	must	never	car	product	ten	stood	wonder
they	see	me	big	last	feet	black	simple	contain	laugh
be	him	give	high	door	care	short	several	front	thousand
at	two	our	such	between	second	numeral	vowel	teach	ago
one	has	under	follow	city	group	class	toward	week	ran
have	look	name	act	tree	carry	wind	war	final	check
this	more	very	why	cross	took	question	lay	gave	game
from	day	through	ask	since	rain	happen	against	green	shape
or	could	just	men	hard	eat	complete	pattern	oh	yes
had	go	form	change	start	room	ship	slow	quick	hot
by	come	much	went	might	friend	area	center	develop	miss
hot	did	great	light	story	began	half	love	sleep	brought
but	my	think	kind	saw	idea	rock	person	warm	heat
some	sound	say	off	far	fish	order	money	free	snow
what	no	help	need	sea	mountain	fire	serve	minute	bed
there	most	low	house	draw	north	south	appear	strong	bring
we	number	line	picture	left	once	problem	road	special	sit
can	who	before	try	late	base	piece	map	mind	perhaps
out	over	turn	us	run	hear	told	science	behind	fill
other	know	cause	again	dont	horse	knew	rule	clear	east
were	water	same	animal	while	cut	pass	govern	tail	weight
all	than	mean	point	press	sure	farm	pull	produce	language
your	call	differ	mother	close	watch	top	cold	fact	among
when	first	move	world	night	color	whole	notice	street	
up	people	right	near	real	face	king	voice	inch	
use	may	boy	build	life	wood	size	fall	lot	
word	down	old	self	few	main	heard	power	nothing	
how	side	too	earth	stop	enough	best	town	course	
said	been	does	father	open	plain	hour	fine	stay	
an	now	tell	head	seem	girl	better	certain	wheel	
each	find	sentence	stand	together	usual	true	fly	full	
she	any	set	own	next	young	during	unit	force	
which	new	three	page	white	ready	hundred	lead	blue	

Appendix B

English function words

Obtained from http://myweb.tiscali.co.uk/wordscape/museum/funcword.html, numbers 1–10, 50, 100 and 100 were added.

1	beside	fifty	i've	or	ten	underneath
2	between	first	isn't	once	tenth	unless
3	beyond	five	it	one	than	until
4	billion	for	its	only	that	up
5	billionth	fortieth	it's	other	that	us
6	both	forty	itself	others	that's	very
7	each	four	just	ought	the	when
8	but	fourteen	last	oughtn't	their	was
9	by	fourteenth	less	our	theirs	wasn't
10	can	fourth	many	ours	them	we
50	can't	hundred	me	ourselves	themselves	we'd
100	could	from	may	out	these	we'll
1000	couldn't	get	might	over	then	were
a	did	gets	million	quite	thence	we're
about	didn't	getting	millionth	rather	there	weren't
above	do	got	mine	round	therefore	we've
after	does	had	more	second	they	what
again	doesn't	hadn't	most	seven	they'd	whence
ago	doing	has	much	seventeen	they'll	where
all	done	hasn't	must	seventeenth	they're	whereas
almost	don't	have	mustn't	seventh	third	which
along	down	haven't	my	seventieth	thirteen	while
already	during	having	myself	seventy	thirteenth	whither
also	eight	he	near	shall	thirtieth	who
although	eighteen	he'd	nearby	shan't	thirty	whom
always	eighteenth	he'll	nearly	she'd	this	whose
am	eighth	hence	neither	she	thither	why
among	eightieth	her	never	she'll	those	will
an	eighty	here	next	she's	though	with
and	either	hers	nine	should	thousand	within
another	eleven	herself	nineteen	shouldn't	thousandth	without
any	eleventh	he's	nineteenth	since	three	won't
anybody	enough	him	ninetieth	six	thrice	would
anything	even	himself	ninety	sixteen	through	wouldn't
anywhere	ever	his	ninth	sixteenth	thus	yes
are	every	hither	no	sixth	till	yesterday
aren't	everybody	how	nobody	sixtieth	to	yet
around	everyone	however	none	sixty	towards	you
as	everything	near	noone	so	today	your
at	everywhere	hundredth	nothing	some	tomorrow	you'd
back	except	i	nor	somebody	too	you'll
else	far	i'd	not	someone	twelfth	you're
be	few	if	now	something	twelve	yours
been	fewer	i'll	nowhere	sometimes	twentieth	yourself
before	fifteen	i'm	of	somewhere	twenty	yourselves
being	fifteenth	in	off	soon	twice	you've
below	fifth	into	often	still	two	
beneath	fiftieth	is	on	such	under	

Appendix C

Graph visualisation using GraphViz

Each GraphViz file starts with this heading, which sets styles for nodes, edges and the graph layout.

```
graph [overlap=false,
       outputorder=edgesfirst]; # Render nodes after edges
edge  [color=gray,
       arrowsize=0.3,
       penwidth=0.8];
node  [shape=box,
       margin=0.03,
       height=0.02,
       width=0.01,
       fontsize=9,
       fontname="Helvetica",
       style="filled,rounded",
       fillcolor=black, # This is redefined at each node
       fontcolor=white,
       penwidth=0.5 ];
```

These files are then compiled into pdf images by invoking:

```
dot -Kneato          \  # Use the neato layout
    -Tps2            \  # Export to ps2 (nicer than direct pdf
        export)
    <graph.gv>       \
    -o <output.ps>   \
    | ps2pdf output.ps  # Convert ps to pdf
```

Appendix D

Essays used for system evaluation

Essay 1
The Computer Laboratory is an academic department within the University of Cambridge that encompasses Computer Science, along with many aspects of Engineering, Technology and Mathematics. CL consists of 41 academic staff, 29 support staff, 5 research fellows, 81 post-doctoral research workers and 119 PhD students. CL has over 300 undergraduates studying for Part I, II and III of the Computer Science Tripos and 36 graduate students studying for the MPhil in Advanced Computer Science.

Essay 2
Isaac Newton was an English physicist and mathematician. He was born on in 1642. He is famous for his work on the laws of motion, optics, gravity, and calculus. In 1687, Newton published a book called the Philosophiae Naturalis Principia Mathematica. There he presents his theory of universal gravitation and three laws of motion.

Newton built the first practical reflecting telescope in 1668. He also developed a theory of light based on the observation that a prism decomposes white light into the colours of the rainbow. Newton also shares credit with Gottfried Leibniz for the development of calculus.

Newton's ideas on light, motion, and gravity dominated physics for the next three centuries. Newton died on 31 March 1727.

Essay 3
Sir Isaac Newton was an English physicist and mathematician (described in his own day as a "natural philosopher") who is widely recognised as one of the most influential scientists of all time and as a key figure in the scientific revolution. His book Philosophia Naturalis Principia Mathematica ("Mathematical Principles of Natural Philosophy"), first published in 1687, laid the foundations for classical mechanics. Newton also made seminal contributions to optics and shares credit with Gottfried Leibniz for the development of calculus.

Newton's Principia formulated the laws of motion and universal gravitation, which dominated scientists' view of the physical universe for the next three centuries. By deriving Kepler's laws of planetary motion from his mathematical description of gravity, and then using the same principles to account for the trajectories of comets, the tides, the precession of the equinoxes, and other phenomena, Newton removed the last doubts about the validity of the heliocentric model of the cosmos. This work also demonstrated that the motion of objects on Earth and of celestial bodies could be described by the same principles. His prediction that the Earth should be shaped as an oblate spheroid was later vindicated by the measurements of Maupertuis, La

Condamine, and others, which helped convince most Continental European scientists of the superiority of Newtonian mechanics over the earlier system of Descartes.

Newton also built the first practical reflecting telescope and developed a theory of colour based on the observation that a prism decomposes white light into the many colours of the visible spectrum. He formulated an empirical law of cooling, studied the speed of sound, and introduced the notion of a Newtonian fluid. In addition to his work on calculus, as a mathematician Newton contributed to the study of power series, generalised the binomial theorem to non-integer exponents, developed Newton's method for approximating the roots of a function, and classified most of the cubic plane curves.

Newton was a fellow of Trinity College and the second Lucasian Professor of Mathematics at the University of Cambridge. He was a devout but unorthodox Christian and, unusually for a member of the Cambridge faculty of the day, he refused to take holy orders in the Church of England, perhaps because he privately rejected the doctrine of the Trinity. Beyond his work on the mathematical sciences, Newton dedicated much of his time to the study of biblical chronology and alchemy, but most of his work in those areas remained unpublished until long after his death. In his later life, Newton became president of the Royal Society. He also served the British government as Warden and Master of the Royal Mint.

Essay 4

Arthur Dent seems to be having trouble with his lifestyle.

He went to Eaton House Prep, where he encountered Blisters Smyth, a bullying head boy who filled Arthur's slippers with tapioca pudding.

Arthur's university yearbook referred to him as "most likely to end up living in a hole in the Scottish highlands with only the chip on his shoulder for company."

After leaving university, Arthur got into radio. He lived in London for a while but moved to his house in the West Country after city life made him nervous and irritable.

One morning, he woke up with a hangover to discover his house being demolished. As Arthur lay in front of the bulldozers, his friend Ford Prefect took him to the pub and revealed that he was an alien.

Shortly after this, his native planet was destroyed to make way for a hyperspace bypass.

Things got really confusing when Arthur discovered a girl he once met at a party had absconded with a two headed alien on a stolen spaceship.

This was just a prelude to a string of very strange events, throughout which Arthur would be utterly unable to get a cup of tea.

Essay 5

Winston Smith works as a clerk in the Records Department of the Ministry of Truth. He has to rewrite historical documents so they match the constantly changing current party line. Because of his proximity to the mechanics of rewriting history, Winston Smith nurses doubts about the Party and its monopoly on truth. Whenever Winston appears in front of a telescreen, he is referred to as 6079 Smith W.

Winston meets a mysterious woman named Julia, a fellow member of the Outer Party who also bears resentment toward the party's ways; the two become lovers. Winston soon gets in touch with O'Brien, a member of the Inner Party whom Winston believes is secretly a member of The Brotherhood, a resistance organisation dedicated to overthrowing the Party's

dictatorship. Winston and Julia join the Brotherhood, believing they have met a kindred spirit.

However, O'Brien is really an agent of the Thought Police, which has had Winston under surveillance for seven years. Winston and Julia are soon captured. Winston remains defiant when he is captured, and endures several months of extreme torture at O'Brien's hands. However, his spirit finally breaks when he is taken into Room 101 and confronted by his own worst fear: the unspeakable horror of slowly being eaten alive by rats.

Winston is terrified by the realisation that this threat will come true if he continues to resist. He denounces Julia and pledges his loyalty to the Party. Winston is then forced to accept the assertion $2 + 2 = 5$, a phrase which has entered the lexicon to represent obedience to ideology over rational truth or fact. By the end, O'Brien's torture has reverted Winston to his previous status as an obedient, unquestioning slave who genuinely loves Big Brother. Beyond his total capitulation and submission to the party, Winston's fate is left unresolved in the novel. As Winston realises that he loves Big Brother, he dreams of a public trial and an execution; however the novel itself ends with Winston, presumably still in the Chestnut Tree Café, contemplating the face of Big Brother.

Essay 6

EDSAC was an early British computer. It was constructed by Maurice Wilkes and his team at the University of Cambridge Mathematical Laboratory. EDSAC was the second electronic digital stored-program computer to go into regular service. EDSAC ran its first programs on 6 May 1949, when it calculated a table of squares and a list of prime numbers. EDSAC was finally shut down in 1958. It was capable of 650 instructions per second. It had 1024 17-bit words of memory in mercury ultrasonic delay lines. It had Paper tape input and teleprinter output at 6 2/3 characters per second. It had 3000 valves, 12 kW power consumption, needed a room 5m by 4m.

Essay 7

Pink Floyd may have been a revolutionary band from the late sixties to today, but you truly can not appreciate the band until you know of its members.

First off is, David Gilmour. He was born on March sixth, nineteen forty-six. He has eight kids, four with a past wife, Ginger, and three with his current wife, Charlie (Charlie had a kid from a previous marriage). David was raised by easy going parents, and was given his own guitar at thirteen. As a young teenager he was brought to the U.S. because of his father's job. He started playing at military bases in a band named Jokers Wild, and working odd jobs including modeling. In his spare time he enjoys flying and collecting vintage aircrafts.

Next is Nicholas Mason the band's drummer. Born on January twenty-seventh, nineteen forty-four, Nick has three kids with his wife, Annette. He was born to a rich family and from childhood accompanied his father to car shows. Nick attended the Frensham Heights boarding school at which he is still remembered as a mischief maker. Today he races and collects cars such as Ferraris, Bugattis, and Macerates.

Another important player in this awesome band was Richard Wright. Born on July twenty-eighth, nineteen forty-five, and has three kids: Gala, Jamie, and Benjamin. Met Nick and Roger at he Regent Street Poly. He enjoys taking his sixty-five foot yacht out when he is presented the opportunity.

The second most important person in the band was George Waters or Roger As he preferred. Born on September sixth, nineteen forty-three. Roger has two kids and has recently divorced

for the third time. Roger's farther was killed in the war and he never had a chance to know him. He tried the navy and quit soon after joining.

The single most important person that truly got the band persona started was Roger "Syd" Barrett. Born on January sixth, nineteen forty-six. Barrett had an above average childhood with supportive parents with a fair amount of money. He was a good student with a lot of friends. His parents encouraged his music with providing the supplies necessary for it. Syd was the founder of Pink Floyd and gave the band its own personality.

Since the sixties the band had pushed their music to the limit, mixing pop, classical, and rock they had a ground breaking formula to success. Most fans know of only the hits that the band had in seventies, and of none of the controversy and pit falls the band suffered throughout their career. So here is the biography of the band Pink Floyd.

The band was originally formed sometime in the mid sixties, and stared with a totally different type of music than the more popular seventies. True Pink Floyd was still a psychedelic rock band but there was something different with Syd writing the music. For example, the band was much more conventional focusing on making rock and R&B songs. Soon they started trying new things and added instrumental solos and mixed in new effects like feedback, and reverb. Also it was said that Barrett's lyrics "viewed the world with a since of poet, child-like wonder".

Essay 8

A mindmap is a diagram used to visually organize information. A mindmap is often created around a single concept, drawn as an image in the center of a blank landscape page, to which associated representations of ideas such as images, words and parts of words are added. Major ideas are connected directly to the central concept, and other ideas branch out from those.

Mindmap can be drawn by hand, either as "rough notes" during a lecture, meeting or planning session, for example, or as higher quality pictures when more time is available.

Mindmap is considered to be a type of spider diagram. Mindmap was first popularized by British popular psychology author and television personality Tony Buzan. Mindmap can be used to generate, visualize, structure, and classify ideas, and as an aid to studying and organizing information, solving problems, making decisions, medicine and writing.

Appendix E

Instructions for human participants

The following instructions were sent via email to the human participants (only the relevant part of the email is included):

[...]
What is my project about? I am creating a system that makes mind maps (see http://en.wikipedia.org/wiki/Mind_map) from the given plain text. In order to test my system I need to show that the mind maps generated by the system are statistically indistinguishable from humans.

I would need your help with creating these mind maps for the 8 essays which are in the attached files. I think it should not take more than 45 minutes and you shall be given a chocolate as a reward!

Please create mind maps for essays in files `essay0.txt` to `essay7.txt`. Hence your output will be 8 mind maps written in English. I recommend using https://www.text2mindmap.com/. It is very easy-to-use. You don't need to send me the image, but rather the text you created (i.e. the contents of the input area) which is a tab-indented tree (mind map) in a plain text format. Save this in a text file called `mind_map<essay number>.txt`.

Attached files:

- Essays to be processed.
- Mind map created by me for `essay0.txt` for you inspiration. However, feel free to use you own format and style.

Some formalities:

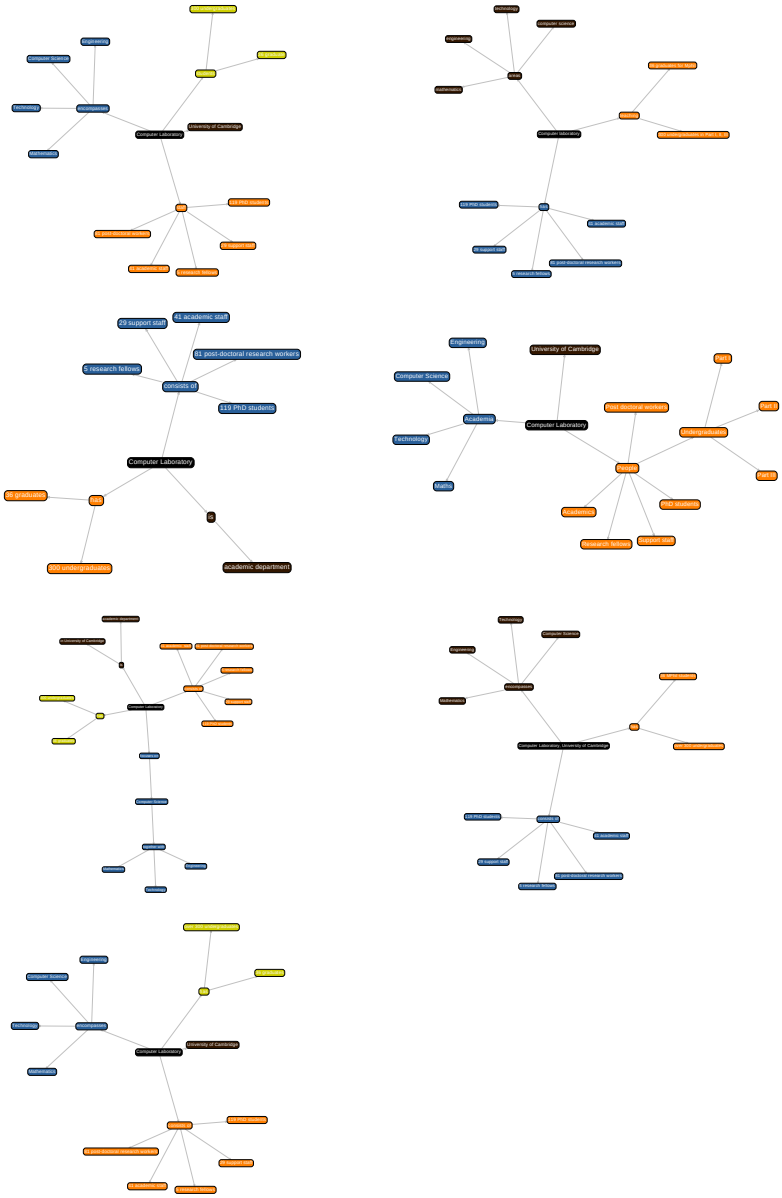
- You can withdraw from the experiment at any time.
- All collected data will be anonymised.
- I won't collect nor store any personal data.
- The experiment doesn't contain any stressful situations.
- Your skills, knowledge or performance won't be measured, rated or compared.
- The experiment is unpaid (you will get only a chocolate).
- The experiment should take you approximately about 45 minutes.

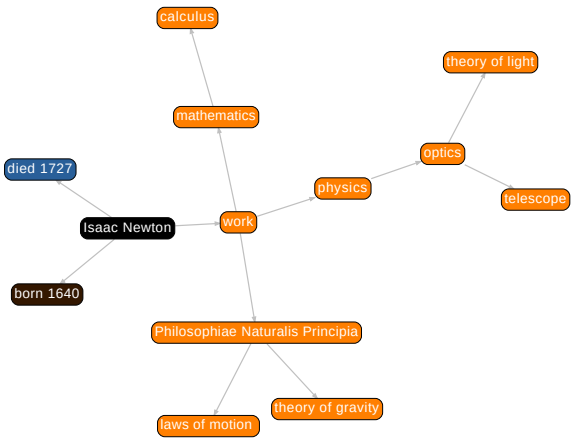
[...]

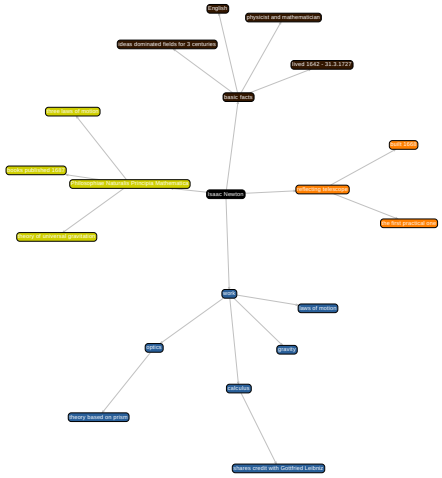
Appendix F

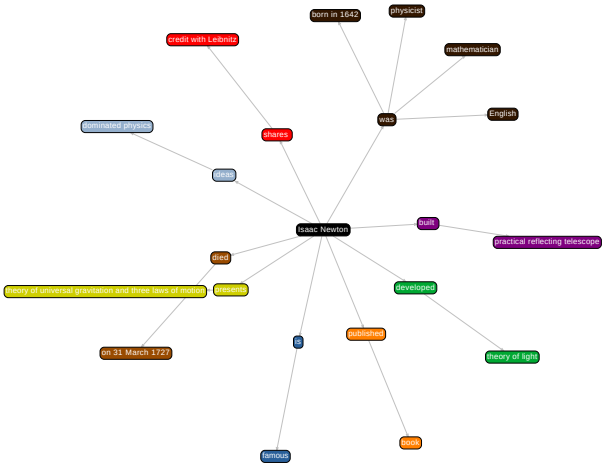
Human participants' mind maps

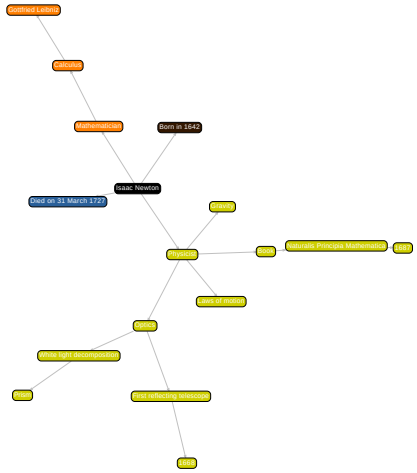
All the mind maps are included here for completeness. The purpose is to demonstrate how the variations in the structure from participant to participant.

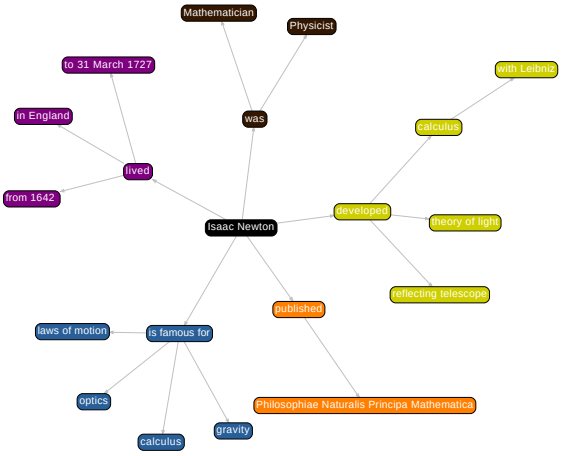


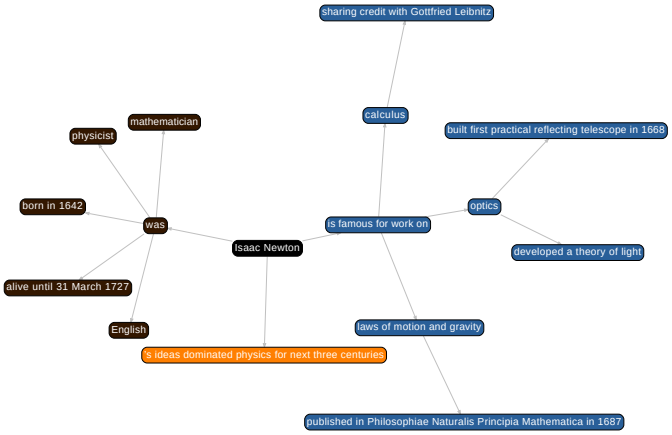

calculus
theory of light
mathematics
optics
died 1727
physics
telescope
Isaac Newton
work
born 1640
Philosophiae Naturalis Principia
theory of gravity
laws of motion

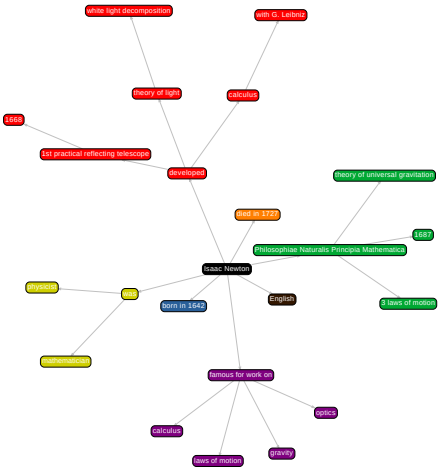

English
physical and mathematician
ideas dominated fields for 3 centuries
lived 1642 - 31.3.1727
isaac facts
Isaac Newton
built 1668
reflecting telescope
laws of motion
gravity
calculus
shares credit with Gottfried Leibnitz

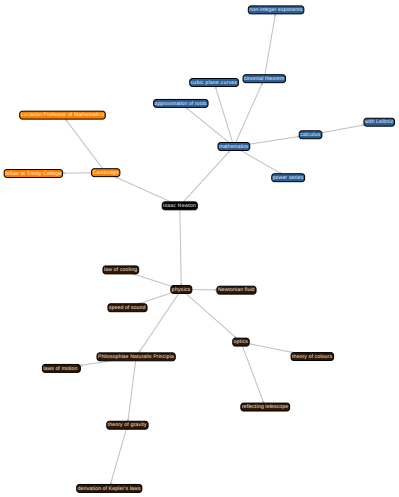

born in 1642
physicist
credit with Leibnitz
mathematician
was
English
shares
Isaac Newton
built
practical reflecting telescope
died
theory of universal gravitation and three laws of motion
developed
published
on 31 March 1727
theory of light
famous
book

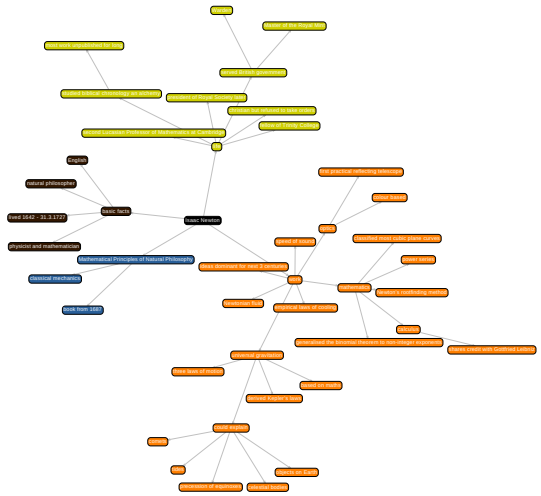

Gottfried Leibnitz
calculus
Mathematician
Born in 1642
Isaac Newton
Died on 31 March 1727
gravity
physics
Naturalis Principia Mathematica
laws of motion
optics
white light decomposition
prism
first reflecting telescope
gold

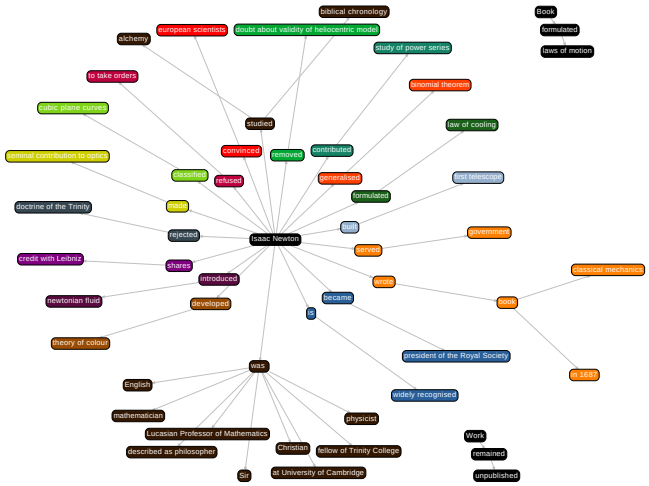

Mathematician
Physicist
to 31 March 1727
with Leibniz
in England
was
calculus
lived
from 1642
developed
Isaac Newton
theory of light
reflecting telescope
published
laws of motion
is famous for
optics
Philosophiae Naturalis Principa Mathematica
gravity
calculus

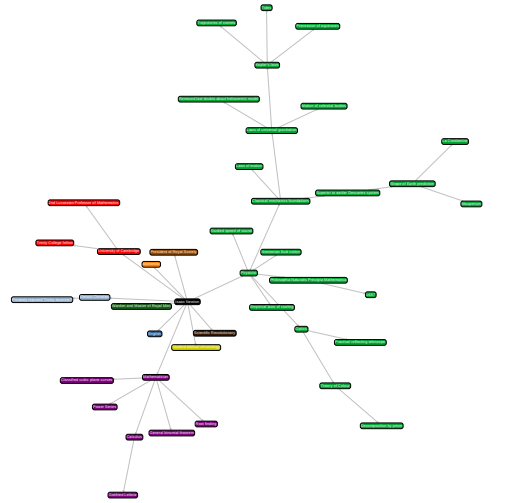

sharing credit with Gottfried Leibnitz
calculus
mathematician
physicist
built first practical reflecting telescope in 1668
born in 1642
optics
was
is famous for work on
Isaac Newton
developed a theory of light
alive until 31 March 1727
English
laws of motion and gravity
's ideas dominated physics for next three centuries
published in Philosophiae Naturalis Principia Mathematica in 1687

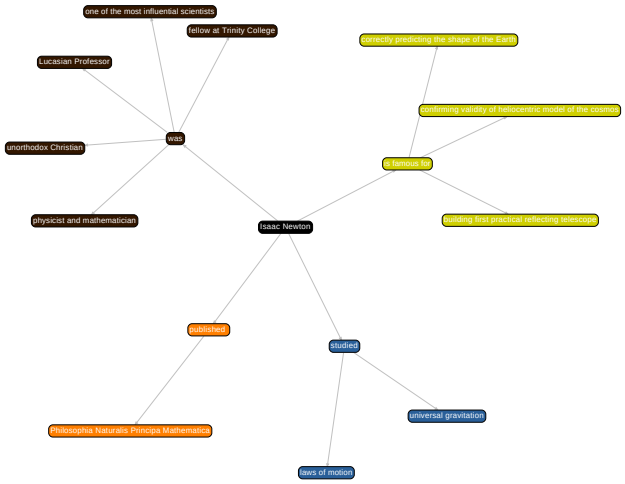

white light decomposition
with G. Leibniz
theory of light
calculus
1668
1st practical reflecting telescope
developed
theory of universal gravitation
died in 1727
1687
Philosophiae Naturalis Principia Mathematica
Isaac Newton
physicist
was
born in 1642
English
3 laws of motion
mathematician
famous for work on
optics
calculus
gravity
laws of motion

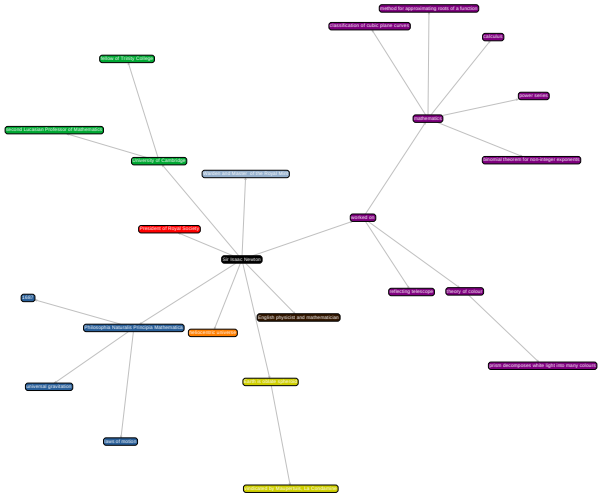

non-integer exponents
cubic plane curves
binomial theorem
approximation of roots
Lucasian Professor of Mathematics
with Leibniz
calculus
mathematics
power series
fellow in Trinity College
mathematics
Isaac Newton
law of cooling
physics
Newtonian fluid
speed of sound
optics
theory of colour
Philosophiae Naturalis Principia
laws of motion
reflecting telescope
theory of gravity
derivation of Kepler's laws

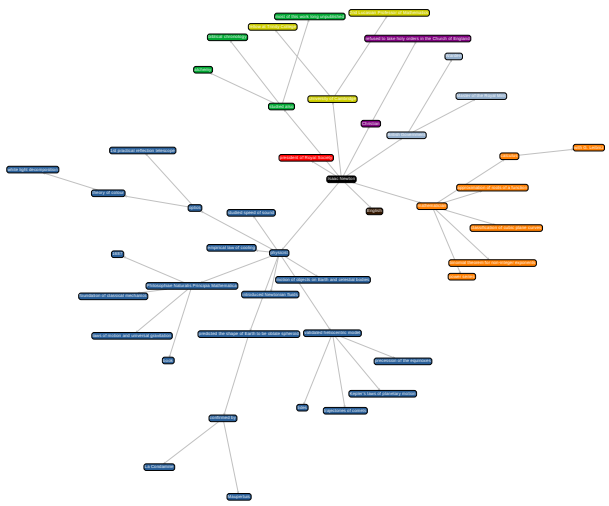

English
natural philosophy
basic facts
Isaac Newton
born 1642 - 31.3.1727
physicist and mathematician
Mathematical Principles of Natural Philosophy
classical mechanics
book from 1687
speed of sound
law of cooling
derivation of Kepler's laws
theory of colour
theory of gravity
derivation of Kepler's laws
derivation of Newton's law

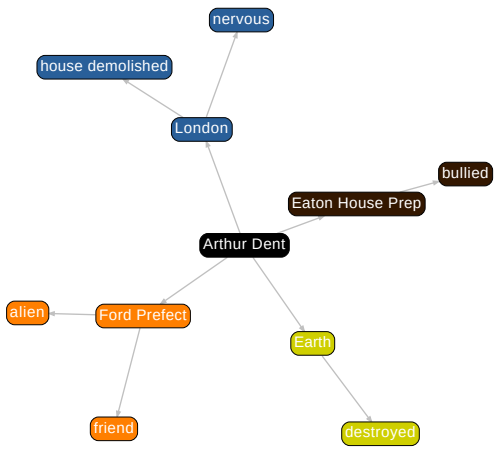

biblical chronology
Book
formulated
laws of motion
alchemy
european scientists
doubt about validity of heliocentric model
study of power series
to take orders
binomial theorem
cubic plane curves
studied
law of cooling
seminal contribution to optics
convinced
removed
contributed
classified
refused
generalised
first telescope
formulated
doctrine of the Trinity
made
rejected
Isaac Newton
government
credit with Leibniz
shares
introduced
classical mechanics
newtonian fluid
developed
became
book
theory of colour
president of the Royal Society
in 1687
was
English
widely recognised
mathematician
physicist
Lucasian Professor of Mathematics
Work
described as philosopher
Christian
fellow of Trinity College
remained
Sir
at University of Cambridge
unpublished

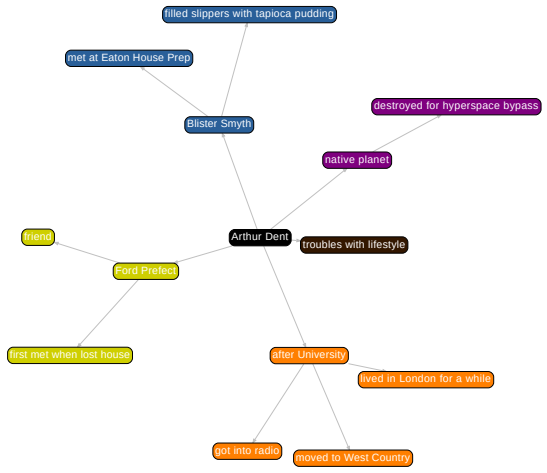


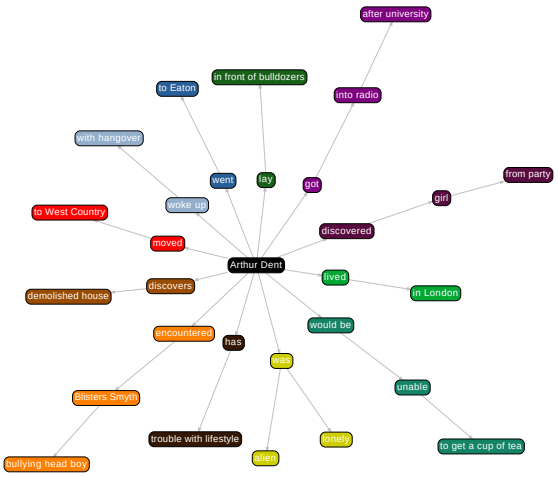

one of the most influential scientists
fellow at Trinity College
correctly predicting the shape of the Earth
Lucasian Professor
confirming validity of heliocentric model of the cosmos
was
unorthodox Christian
a famous fib
physicist and mathematician
Isaac Newton
building first practical reflecting telescope
published
studied
universal gravitation
Philosophia Naturalis Principa Mathematica
laws of motion

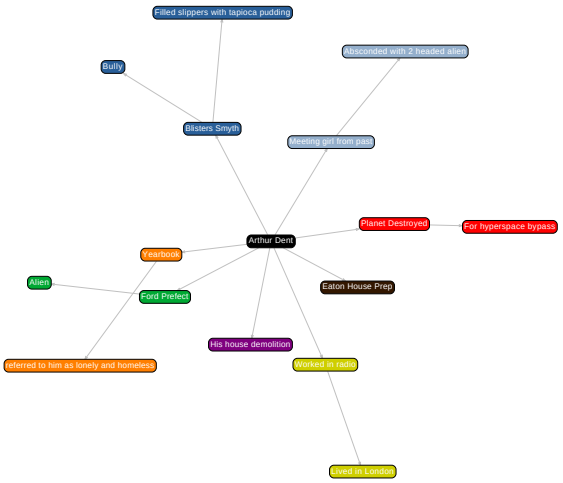

method for approximating roots of a function
classification of cubic plane curves
calculus
fellow of Trinity College
power series
mathematics
second Lucasian Professor of Mathematics
university of Cambridge
seminal theorem for non-integer exponents
spectral all
president of Royal Society
Sir Isaac Newton
reflecting telescope
theory of colour
English physicist and mathematician
Philosophia Naturalis Principia Mathematica
prism decomposes white light into many colours
universal gravitation
laws of motion

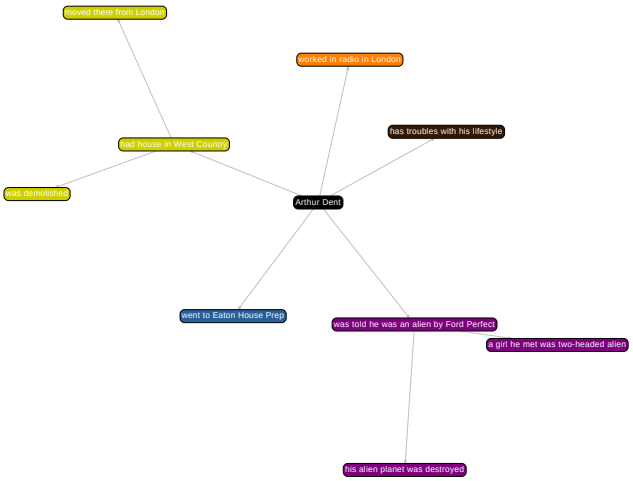


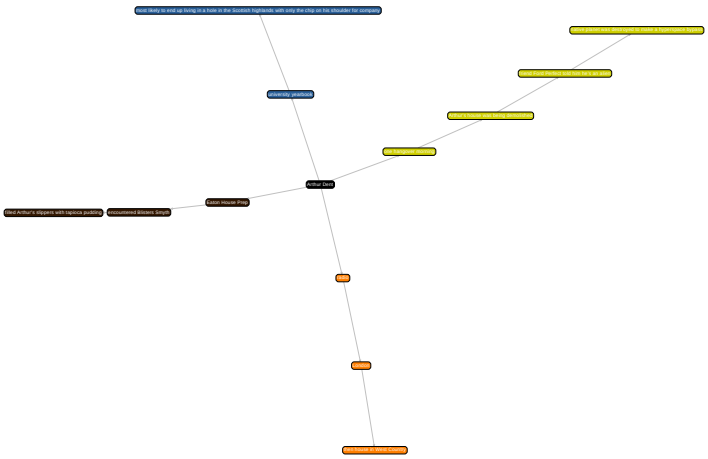

nervous
house demolished
London
bullied
Eaton House Prep
Arthur Dent
alien
Ford Prefect
Earth
friend
destroyed


filled slippers with tapioca pudding
met at Eaton House Prep
Blister Smyth
destroyed for hyperspace bypass
native planet
friend
Arthur Dent
troubles with lifestyle
Ford Prefect
first met when lost house
after University
lived in London for a while
got into radio
moved to West Country

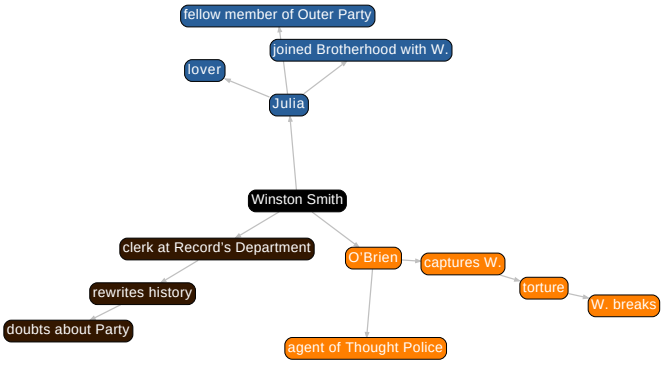

after university
in front of bulldozers
to Eaton
into radio
with hangover
went
lay
got
from party
girl
woke up
to West Country
discovered
moved
Arthur Dent
lived
discovers
in London
demolished house
would be
encountered
has
was
unable
Blisters Smyth
trouble with lifestyle
lonely
to get a cup of tea
alien
bullying head boy

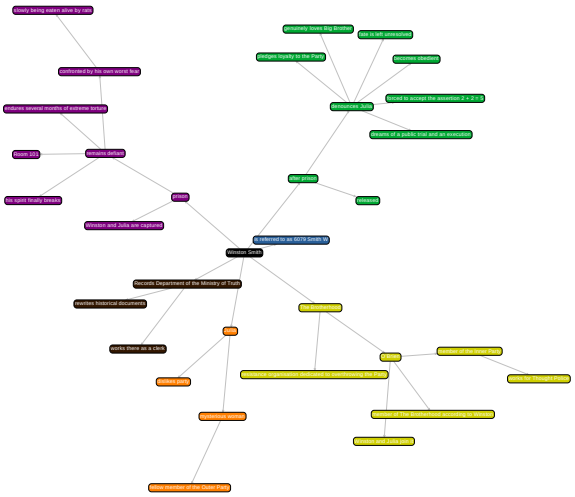

Filled slippers with tapioca pudding
Absconded with 2 headed alien
Bully
Blisters Smyth
Meeting girl from pub
Planet Destroyed
For hyperspace bypass
Arthur Dent
Yearbook
Alien
Ford Prefect
Eaton House Prep
His house demolition
referred to him as lonely and homeless
Worked in radio
Lived in London

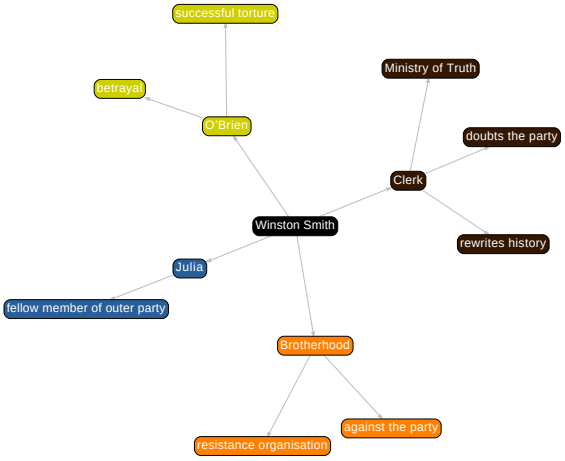

moved there from London
worked in radio in London
has troubles with his lifestyle
old house in West Country
was demolished
Arthur Dent
went to Eaton House Prep
was told he was an alien by Ford Perfect
a girl he met was two-headed alien
his alien planet was destroyed

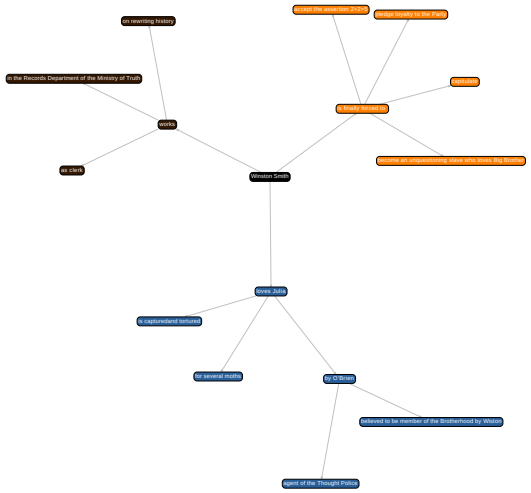


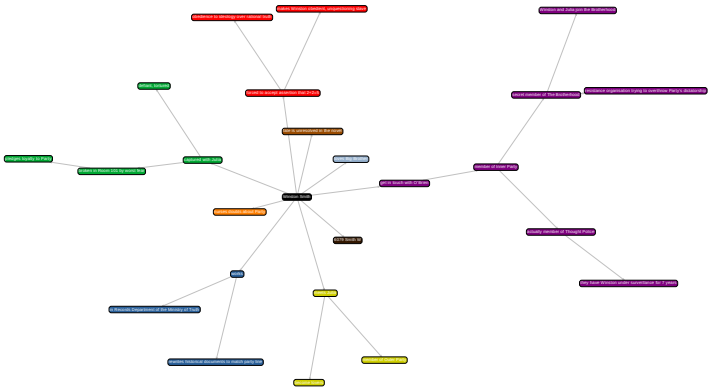

Bully
unable
filled Arthur's slippers with pudding
Blisters Smyth
Eaton House Prep
trouble with his lifestyle
house being demolished
Arthur Dent
alien
Ford Prefect
native planet destroyed
West Country
Absconded
university yearbook
hyperspace bypass
strange events
unable to get a cup of tea
lonely to live
eats only a chip for company
in a hole

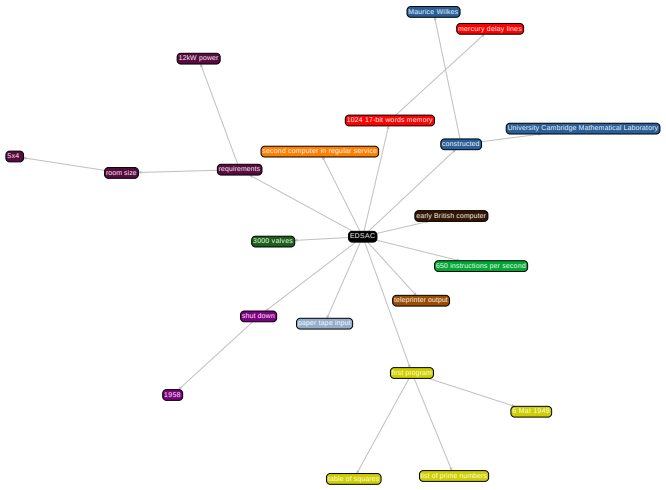

fellow member of Outer Party
joined Brotherhood with W.
lover
Julia
Winston Smith
clerk at Record's Department
O'Brien
captures W.
torture
W. breaks
rewrites history
doubts about Party
agent of Thought Police

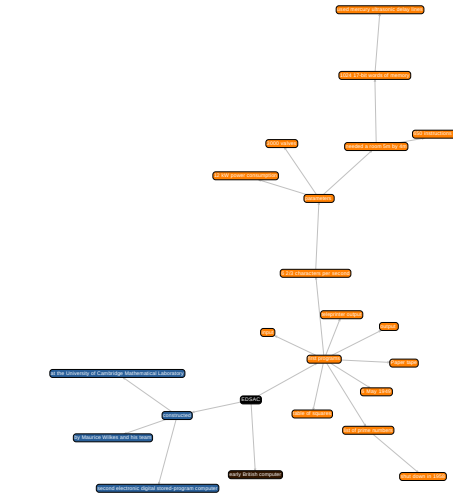



successful torture
Ministry of Truth
betrayal
O'Brien
doubts the party
Clerk
Winston Smith
rewrites history
Julia
fellow member of outer party
Brotherhood
against the party
resistance organisation

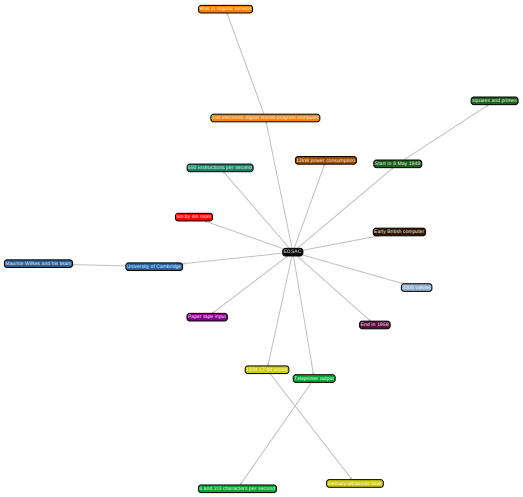


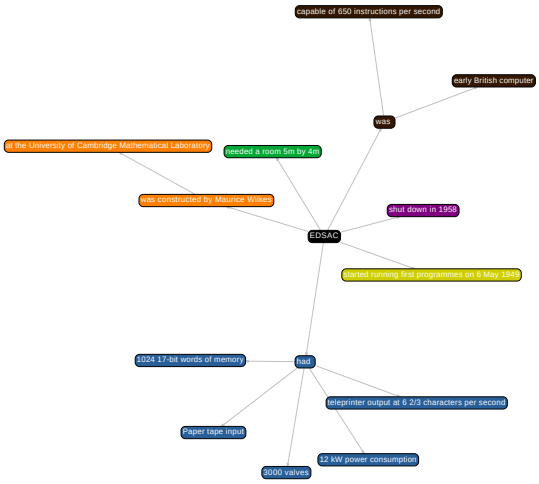


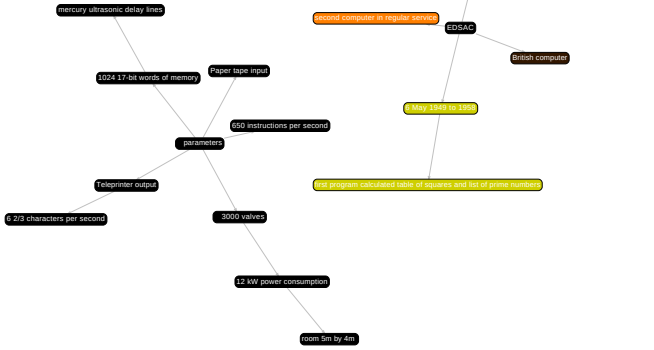

Maurice Wilkes
mercury delay lines
12kW power
1024 17-bit words memory
University Cambridge Mathematical Laboratory
constructed
second computer in regular service
5x4
room size
requirements
early British computer
EDSAC
3000 valves
650 instructions per second
teleprinter output
shut down
paper tape input
1958
list programs
6 May 1949
table of squares
list of prime numbers

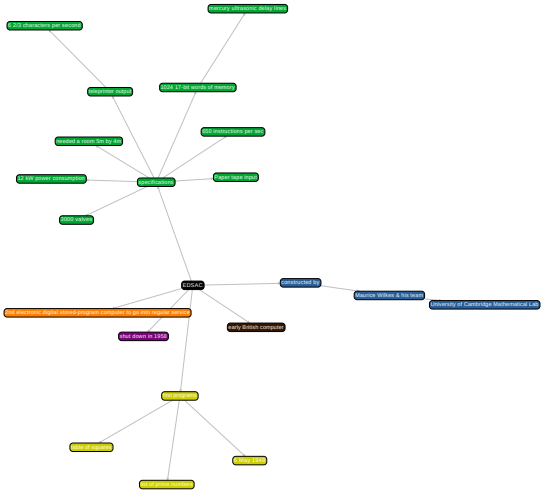

mercury ultrasonic delay line
1024 17-bit words of memory
needed a room 5m by 4m
650 instructions per second
12 kW power consumption
parameters
650 characters per second
teleprinter output
paper tape
at the University of Cambridge Mathematical Laboratory
EDSAC
6 May 1949
constructed
table of squares
list of prime numbers
by Maurice Wilkes and his team
second electronic digital stored-program computer
early British computer
shut down in 1958

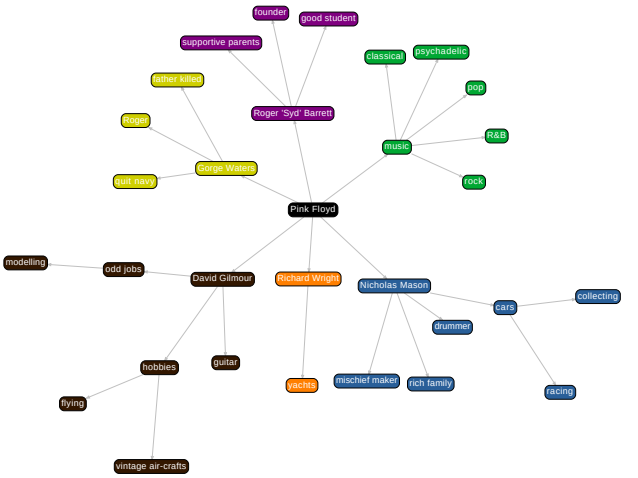

in delay lines
memory
Paper tape input
teleprinter output
12 kW power consumption
list of prime numbers
had
300 valves
table of squares
calculation
to regular service
Edsac
was
British
list programs
at 6 May 1949
by Maurice Wilkes
constructed
needed
capable
second computer
of 650 instructions per second
computer
room
at University of Cambridge
shut down
5m by 4m
in 1958

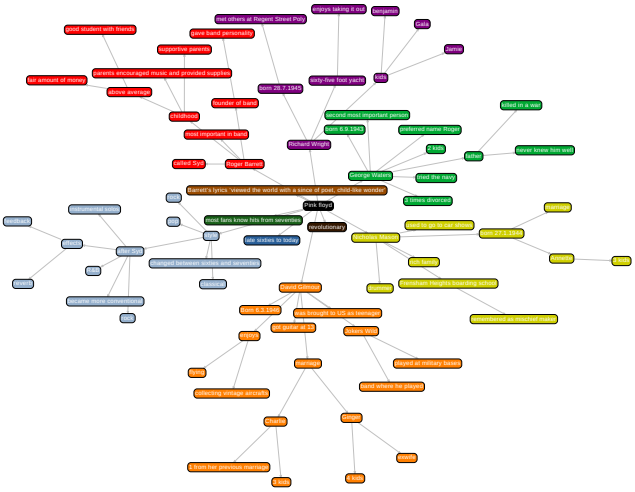

time to regular service
squares and primes
second electronic digital stored-program computer
12kW power consumption
650 instructions per second
shut in 6 May 1949
delay lines used
early British computer
EDSAC
Maurice Wilkes and his team
University of Cambridge
Paper tape input
end in 1958
teleprinter output
6 and 2/3 characters per second

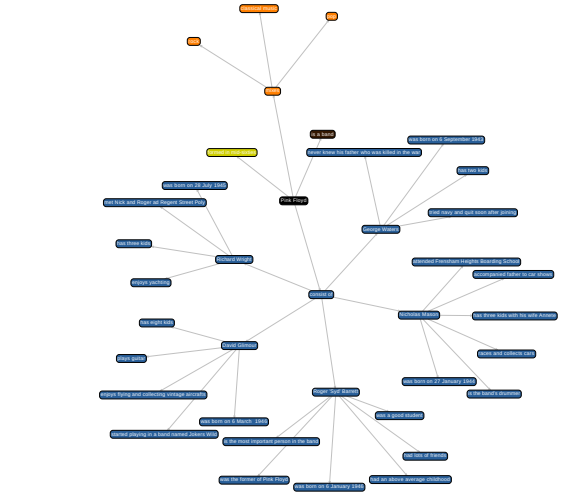

capable of 650 instructions per second
early British computer
was
at the University of Cambridge Mathematical Laboratory
needed a room 5m by 4m
was constructed by Maurice Wilkes
shut down in 1958
EDSAC
started running first programmes on 6 May 1949
1024 17-bit words of memory
had
teleprinter output at 6 2/3 characters per second
Paper tape input
12 kW power consumption
3000 valves

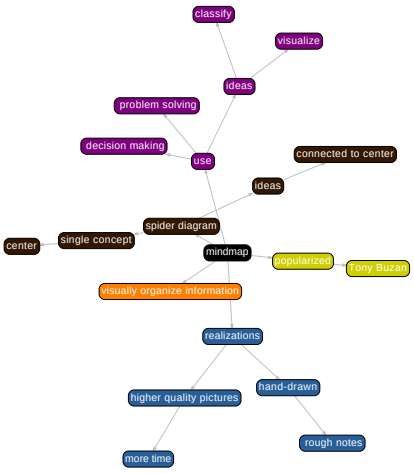

by Maurice Wilkes and team
at the University of Cambridge Mathematical Laboratory
constructed
mercury ultrasonic delay lines
second computer in regular service
EDSAC
British computer
1024 17-bit words of memory
Paper tape input
6 May 1949 to 1958
650 instructions per second
parameters
Teleprinter output
first program calculated table of squares and list of prime numbers
6 2/3 characters per second
3000 valves
12 kW power consumption
room 5m by 4m

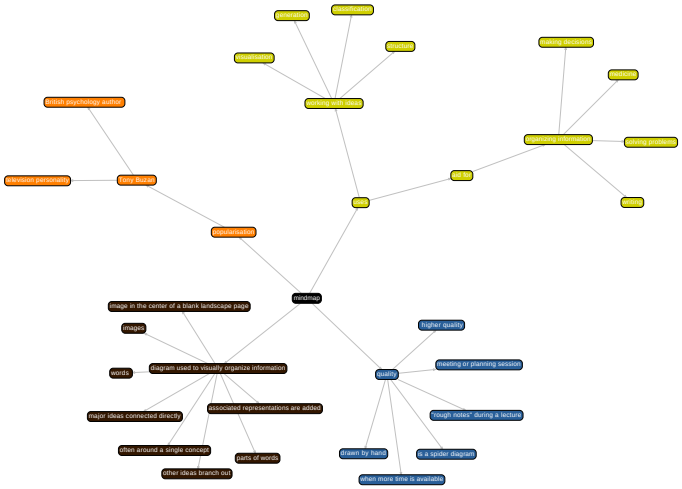

mercury ultrasonic delay lines
6 2/3 characters per second
teleprinter output
1024 17-bit words of memory
needed a room 5m by 4m
650 instructions per sec
12 kW power consumption
specifications
Paper tape input
3000 valves
EDSAC
constructed by
Maurice Wilkes & his team
University of Cambridge Mathematical Lab
first electronic digital stored-program computer to go into regular service
shut down in 1958
early British computer
list programs
table of squares
6 May 1949
list of prime numbers


founder
good student
supportive parents
classical
psychadelic
father killed
pop
Roger 'Syd' Barrett
Roger
music
R&B
George Waters
quit navy
rock
Pink Floyd
modelling
odd jobs
David Gilmour
Richard Wright
Nicholas Mason
cars
collecting
drummer
hobbies
guitar
yachts
mischief maker
rich family
racing
flying
vintage air-crafts

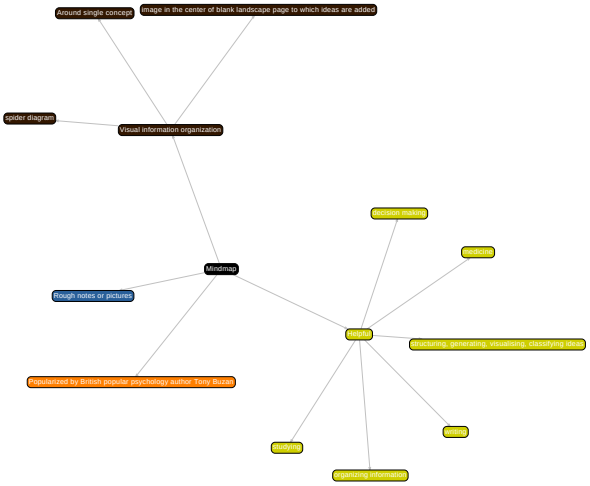


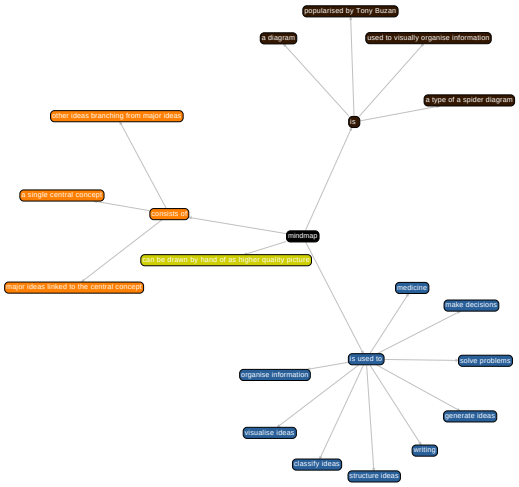


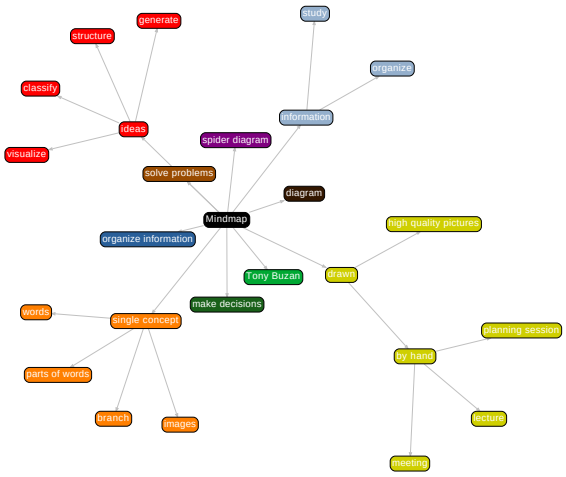

classify
visualize
ideas
problem solving
decision making
use
connected to center
ideas
spider diagram
center
single concept
mindmap
popularized
Tony Buzan
visually organize information
realizations
higher quality pictures
hand-drawn
rough notes
more time


generate
classify
visualize
structure
organizing information
British psychology author
television personality
Tony Buzan
use
popularization
mindmap
image in the center of a blank landscape page
images
words
diagram used to visually organize information
major ideas connected directly
associated representations are added
often around a single concept
parts of words
other ideas branch out
quality
higher quality
meeting or planning session
"rough notes" during a lecture
drawn by hand
is a spider diagram
when more time is available


classify
visualize
generate
structure
or ideas to
by hand
as rough notes
organising information
medicine
of psychology
British
drawn
used
as aid to
studying
author
popular
by Tony Buzan
can be
writing
popularised
solving problems
making decisions
television personality
was
Mindmap
organises
connects
information
major ideas
is
visually
diagram
created
to central concept
drawn
considered
as an image
around single concept
to be spider diagram


Around single concept
image in the center of blank landscape page to which ideas are added
spider diagram
Visual information organization
decision making
medicine
Mindmap
Rough notes or pictures
helpful
structuring, generating, visualising, classifying ideas
popularized by British popular psychology author Tony Buzan
drawn
studying
organizing information


popularised by Tony Buzan
a diagram
used to visually organise information
is
a type of a spider diagram
other ideas branching from major ideas
a single central concept
consists of
mindmap
can be drawn by hand or as higher quality picture
major ideas linked to the central concept
medicine
make decisions
is used to
solve problems
organise information
generate ideas
visualise ideas
writing
classify ideas
structure ideas


generate
study
structure
organize
classify
information
ideas
spider diagram
visualize
solve problems
diagram
Mindmap
high quality pictures
organize information
Tony Buzan
drawn
make decisions
words
single concept
planning session
by hand
parts of words
branch
images
picture
meeting


writing
solving problems
organizing information
studying
classify
generate
medicine
as an aid to
structure
making decisions
to ... ideas
visualize
used
television personality
Tony Buzan
was popularized by
Mindmap
higher quality pictures
British popular psychology author
can be
diagram
picture
type of spider diagram
drawn by hand during
visually organize information
planning session
single central concept
meeting
associated representations of ideas
parts of words
image in the center of blank page
words
other ideas branch out
images
major ideas connected directly

Appendix G

Project Proposal

Augustin Zidek
Gonville and Caius
az317

Diploma in Computer Science Project Proposal

Generation of Mind Maps from Essays

May 7, 2015

Project Originator: Augustin Zidek

Resources Required: See attached Project Resource Form

Project Supervisor: Paula Buttery

Signature:

Director of Studies: Peter Robinson

Signature:

Overseers: Andrew Rice and Timothy Griffin

Signatures:

Introduction and Description of the Work

Mind mapping is a very powerful technique how to organise your information before writing an essay or while learning something. Wikipedia describes mind map as "a diagram used to visually organise information. A mind map is often created around a single concept, drawn as an image in the centre of a blank landscape page, to which associated representations of ideas such as images, words and parts of words are added. Major ideas are connected directly to the central concept, and other ideas branch out from those."

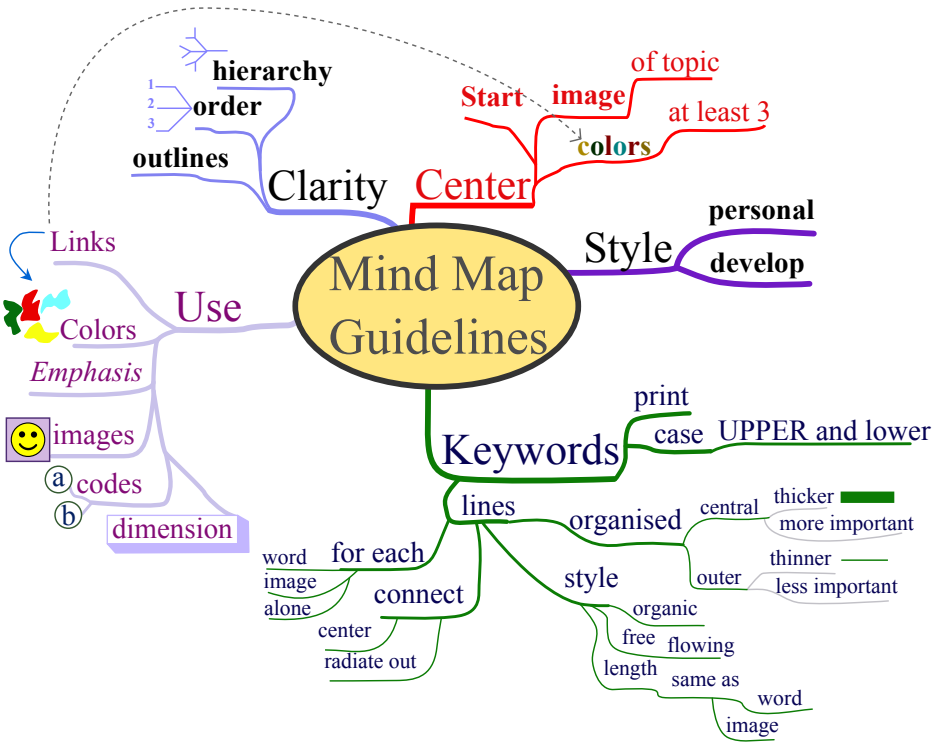
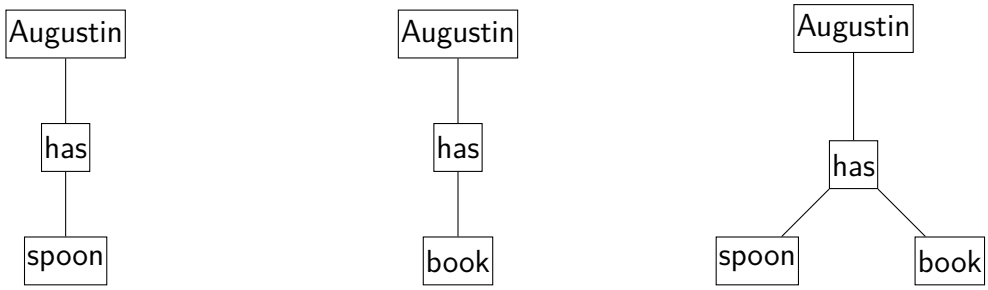


Figure G.1: An example of a mind map, source: Wikipedia

My system should make life easier by completely automating the creation of mind map from a given plain text using techniques of Natural Language Processing.

The task is hence developing a system that takes plain text as an input and then does the following:

1. The text is parsed and irrelevant information is thrown away. Irrelevant information involves for instance most of the adjectives, adverbs, prepositions and conjunctions. For instance the sentence "Augustin is currently writing proposal in his room" should be reduced to "Augustin (subject) writing (verb) proposal (object)".
2. Then the information obtained in step 1 should be adequately grouped together – i.e. common actors and their actions should be found. For instance, if there were two sentences (in this stage stored as a tree) "Augustin has spoon" and "Augustin has book", they should be merged into a graph with main node "Augustin", child node "has" with two child nodes "spoon" and "book", as in the figure G.2.
3. In the final (front-end) step it is necessary to display the generated mind map as a tree, but not in the classical layout with root on the top and its children underneath. The



(a) First tree (b) Second tree (c) Resulting tree

Figure G.2: Merging sentence trees

layout of mind maps dictates the root to be in the middle and its children are placed on a circle around it.

The project offers many opportunities of optimisations which shall be implemented and compared to each other:

- the parsing can be made smarter,
- pronoun to names conversion can be implemented and used to extract more information,
- WordNet and other techniques can be used to group together words which are not necessarily same, but have common meaning (e.g. "owns" and "has").

Resources Required

I have chosen Java to be the language in which I will do the project, as it is the language I am most confident in and all the needed libraries are available in Java.

My laptop (Lenovo ThinkPad Edge E420s) will be used for development and testing. Backup will be done using GitHub and Dropbox on daily basis and using my USB flash drive on weekly basis.

Evaluation will be done with help of human participants – they will be asked to draw word clouds and mind maps which will be then compared to the outputs generated by my system.

Starting Point

I don't have any prior knowledge in Natural Language Processing, only personal interest in linguistics and computational linguistics. I will be using the knowledge obtained in the Part II NLP course.

Substance and Structure of the Project

The aim of the project is to develop a system that takes plain text essay of *at most 500 words* as input and outputs a mind map, that is a tree of the most relevant information in the given text.

Multiple versions/extensions of the system are going to be implemented and compared to each other. **It is probable that only a subset of these will be developed** due to the limited time available:

1. **Naïve** – System which only does parsing and then subject-verb-object extraction.
2. **Named-entity recognition** – System which will try to limit the amount of extracted information only to "interesting" entities, such as names, organisations, places, people, times, amounts, etc.
3. **Relation detection** – System which is also aware of the relationships between the extracted words using WordNet and takes advantage of that when building the mind map tree.
4. **Pronoun resolution** – System will also resolve pronouns to actors. In the text fragment "Shakespeare wrote 38 plays. *He* also wrote 154 sonnets.", *he* shall be replaced by Shakespeare.
5. **Long text** – Creating mind maps of long texts is more challenging. This is because the amount of information increases and it is hence necessary to throw more unimportant pieces of information away. Therefore an algorithm deciding on information importance has to be implemented.
6. **The evaluation** The output that will be evaluated at first will be the word cloud (words with notion of importance) that will be generated together with the mind map. This is much easier to do evaluation on, as it has no structure. Later, if time permits, the whole evaluation system shall be extended to evaluate the actual mind maps.

Evaluation

The project consists of very interesting evaluation part, as it will itself be computationally and algorithmically interesting. The evaluation will be done using a collection of essays. For each essay, each of the 8 human participants will be asked to:

1. Create a word cloud with given number of words.
2. Draw a mind map.

This shall be done early in the project, so that the various systems can be tested against these data.

Implemented algorithms will be then run on the same essays and the generated word clouds for each essay will be compared to the ones created by humans. It will be possible to quantify whether the computer-generated word clouds are statistically indistinguishable from the ones drawn by humans.

The computer-generated and human-generated word clouds can be quantitatively compared by looking at:

1. The generated words (i.e. how many words have the two given lists in common).
2. The importance (order) of the words (i.e. how much does the order of the words differ in the two lists).

As an extension, mind maps can be also compared using a tree distance metric. In order to quantitatively evaluate my algorithm, it needs to be able to:

- Evaluate how many nodes do the trees have in common (i.e. was the relevant information extracted).
- Evaluate the structure of the tree (i.e. how are the pieces of information structured).

The metric that will be used will be some variant of the PARSEVAL metric which is used to measure quality of syntax parsers.

Success Criterion

The following should be achieved:

- Develop systems using techniques described above which are capable of mind map generation.
- Compare these systems against each other by evaluating the word cloud representations corresponding to the generated mind maps.

Timetable and Milestones

According to the suggestions by my DoS, I have decided to divide the work into 10 work packages, each being 2 weeks long. This way I should have the Dissertation ready before beginning of the third term.

Weeks 1 to 2 (26. 10.—09. 11.)

Study various Information Extraction techniques and other relevant NLP material. Get the language parser working. Get familiar with mind map drawing libraries.

Milestones: Able to parse sentences of text. Essays with mind maps from the human participants.

Weeks 3 and 4 (09. 11.—23. 11.)

Further study of the material. Working on the naïve system and named-entity recognition.

Milestones: Working implementation of the naïve system.

Weeks 5 to 6 (23. 11.—07. 12.)

Work on the evaluation system.

Milestones: The tree metric algorithm and tree comparison algorithm should be working so that various implementations can be tested using it.

Weeks 7 and 8 (07. 12.—21. 12.)

Working on implementations of the other versions of the system.

Milestones: Completing more optimisations to the system and evaluating them.

Weeks 9 to 10 (21. 12.—04. 01.)

Working on implementations of the other versions of the system.

Milestones: Completing more optimisations to the system and evaluating them.

Weeks 11 to 12 (04. 01.—18. 01.)

Working on implementations of the other versions of the system.

Milestones: Completing more optimisations to the system and evaluating them.

Weeks 13 to 14 (18. 01.—01. 02.)

Working on possible extensions of the system.

Milestones: Sending the *progress report*.

Weeks 14 to 15 (01. 02.—15. 02.)

Working on possible extensions of the system.

Milestones: Evaluation of extensions done.

Weeks 15 to 16 (15. 02.—01. 03.)

Starting to work on the Dissertation – the Implementation chapter. Finishing the code, perhaps starting work on the extensions.

Milestones: Implementation chapter written. The basis of the system is working.

Weeks 17 to 18 (01. 03.—15.03.)

Further work on the Dissertation.

Weeks 19 to 20 (15. 03—29. 03.)

Working on the Dissertation. System testing. Evaluation of the raw data from the previous evaluations. Write first complete draft of the Dissertation.

Milestones: System works and evaluation complete. Dissertation complete and proof-read by friend and/or Supervisor and possibly other keen people.