Framework for the Optimisation of Full Text Queries on Safety Incident Reports

Matthew Newall

Institute for Railway Research

University of Huddersfield

A thesis submitted in partial fulfilment of the requirements for the degree of

Master of Philosophy

December 2020

Copyright Statement

i The author of this thesis (including any appendices and/or schedules to this thesis) owns any copyright in it (the "Copyright") and s/he has given The University of Huddersfield the right to use such copyright for any administrative, promotional, educational and/or teaching purposes.

ii Copies of this thesis, either in full or in extracts, may be made only in accordance with the regulations of the University Library. Details of these regulations may be obtained from the Librarian. This page must form part of any such copies made.

iii The ownership of any patents, designs, trademarks and any and all other intellectual property rights except for the Copyright (the "Intellectual Property Rights") and any reproductions of copyright works, for example graphs and tables ("Reproductions"), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property Rights and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property Rights and/or Reproductions.

Acknowledgements

I would like first to thank my supervisors, Coen van Gulijk and Violeta Holmes for their invaluable support and guidance during this work. I would additionally to thank Cajetan Chukwulozie, Mark Potter, and the rest of the risk management team at Network Rail for providing access to the Close Call database used in this work, and for their support in this regard. Thanks to Peter Hughes, Paul Allen, and the rest of my colleagues at the Institute of Railway Research for their advice and assistance.

To Rose, Robin, and Blue - thank you for your patience and unwavering support, and for always giving me something to look forward to.

Abstract

The Railway Industry in Great Britain is on the brink of a seed change in the methods used to carry out safety and risk management. Advances in technology over the past few decades have enabled the collection of varied and detailed information regarding the state of track, vehicles, stations, facilities, and personnel at an ever-increasing scale. As more and more industry processes embrace data driven techniques, the growing volume of data being recorded poses a significant challenge in terms of processing and knowledge extraction.

In these chapters, a case study is made of the Close Call system used by railway organisations in Great Britain. Personnel working on the railway network can make reports to this system by phone, email, and in-app, if they see something they consider to have the potential to cause harm or damage. The near ubiquity of mobile devices allows all staff to make free-text reports at any time which has great potential for the completeness of reporting but comes with some drawbacks. The free-text nature of the reports means that basic processing is inadequate for tasks such as categorisation, knowledge extraction, and reporter feedback, requiring ongoing research into appropriate natural language processing techniques. A variety of techniques have shown promise, but the vast and growing quantity of close call reports being made has meant that computing capacity has limited the number of reports that can be processed, and work so far has often been applied only to a small subset of reports.

The aim of this thesis is to demonstrate that it is possible to process or otherwise transform the close call text to allow existing and future analysis techniques pertaining to this text (and data sources like it) to be applied on a larger scale or smaller timeframe. A novel text indexing technique is presented and evaluated, and when compared to other text indexing and pattern matching methods, it is shown that a significant speed increase is possible over the methods previously used.

Versions of the matching technique described in this work have been presented at the European Safety and Reliability Conference in 2017 and 2019.

Contents

List of Figures

Chapter 1

Introduction

Railway safety management has evolved over the years to manage a large number of different risks, and management systems have grown to require a large number of methods, procedures and risk models. In order to keep pace it has been necessary to develop elaborate management structures which are becoming more and more unmanageable. Many models are outdated, and due to the fact that they were often developed in isolation from other risk models there is very little consistency. Different data sources, metrics, timing and scope all compound to result in a labyrinthine mesh of specialised rather than standard definitions, differing suppliers, and even differing experts. For these reasons the case can be made for a more streamlined and dynamic approach to these processes (Swuste et al., 2020), and a key aspect of realising these changes is the efficient use of technology and diverse data sources.

1.1 Why use Computers for Railway Safety?

It is for these reasons, coupled with ever growing avalanche of data collection being carried out on the railways, that the industry is undergoing an IT transformation. Gulijk et al. (2018) outline a number of trends and principles which signpost that the conditions are right for a revolutionary change in the way that data influences the way in which safety management is done:

- **Datafication of the railways**

  Railways around the world have seen a dramatic effort put into IT infrastructures to deliver a myriad of efficient data driven services. some examples include driver-less trains, remote condition monitoring and digital ticketing. These efforts have been bolstered by the IT industry and a healthy academic interest.

- **Serious accidents are not tolerated on the railways**

  Serious accidents cause immense damage, both in terms of the human cost, and to the industry as a whole. The perception of railway safety amongst the public is shaped my coverage of train accidents. Train accidents are generally not at all tolerated.

- **Railway engineering is of high integrity**

  Railway safety is governed by rigorous standards, and a concerted drive for improvement. However, this is not without its costs. Not only does maintaining this level of integrity have high financial costs associated with it, it also leads to an ever growing complexity in safety management as the distance to absolute safety gets ever smaller.

- **Increased use of data driven risk controls in other industries**

  Data driven methods have been making their way into the risk controls of industries worldwide, in a diverse set of applications. For example, asset reliability, mental stress of drivers, and even digital techniques for human reliability

In response to these needs, the digital transformation of safety management is underway. A major tenet in this drive is Big Data Risk Analysis , the aim of which is to develop an enterprise safety management system which provides support for safety and risk management decisions through the following means:

- Extraction of information from a variety of mixed data sources.
- Fast processing to allow inference or relevant safety management information.
- The use of a variety of software and applications to provide sensible interpretation.
- By enabling the connection of the right people at the right time using online interfaces.

1.1.1 The Need for Text Processing

Text reports are an important part of safety and risk management systems across a wide range of sectors. Free form text allows more information to be captured than more structured methods of reporting. Allowing narrative accounts and natural language descriptions allows the capture of new or unanticipated events and trends,

whereas a structured system can only accept information which was known during the design process. Table 1.1 shows some examples of reports submitted to the Close Call system used by railways in Great Britain (the GB railways). A close call is a hazardous situation where the event sequence could have lead to in accident were it not for some intervention (S. Jones, Kirchsteiger, & Bjerke, 1999).

Table 1.1: Examples of Close Call text (Names of people and organisations have been removed)

the tool store on site has been left in an unacceptable condition when staff opened doors tools fell out
subcontract employees refilling generator in tunnels
driver was walking on site with no hi vis orang trousers on
eye protection not worn due to weather conditions

In order to extract information from these reports for safety and risk management purposes, one of the processes currently used is to manually analyse reports to determine what type of hazard the report pertains to from a list of categories. This information is recorded and can then be used to compare the rates of hazard types and identify and locate trends.

Take the first report. From this report we can identify who was involved: "staff", the subject: "tool store" and the hazard: "unacceptable condition" and "tools fell out". This report would be categorised as "Site welfare, site housekeeping" which is a label applied to hazards arising from sites being left untidy or items and equipment being out of place.

The second report is a little more ambiguous in terms of the hazard identified and there are a few labels which might apply. The enclosed space of the tunnels might make "Confined spaces" an appropriate label, or the potential for a build up of fuel vapour could be a "Fire safety" issue. The hazard could be that the fuel was not properly handled, making it a candidate for "Control of chemicals/Hazardous substances"

The last two reports are both examples missing or improper personal protective equipment. The equipment in question being "hi vis orang trousers" and "eye protection. These reports would accordingly be labelled "Personal protective equipment (PPE)"

Processes such as this demonstrate the value in collecting and processing free text reports for safety and risk management. However the volume of reports submitted to the close call system and others like it has reached very high levels, and this volume is increasing year on year. For this reason significant research effort has been placed into developing automatic processes for processing classifying and otherwise extracting information from these text sources.

1.1.2 Prior Work

Hughes, Shipp, Figueres-Esteban, and van Gulijk (2018) developed a method of semi-automated classification of text records from the Close Call system. A close call is a hazardous situation where the event sequence could have lead to in accident were it not for some intervention (S. Jones et al., 1999). The Close Call database contains reports of these kinds of events made by Network Rail employees and specific subcontractors. It contains free form text, which allows expression which would not be possible in a system with structured entry, E.g, selection from a predefined list of hazards. However, this unstructured nature poses a problem for retrieval of information.

Approximately 300000 entries are made per year to the Close Call system, and currently, human intervention is required in order to extract useful information from them, and the volume of entries is set to continue increasing. Hughes et al. (2018) propose an application of Natural Language Processing (NLP) techniques in order to speed up the rate of safety learning from this data.

Hughes and Gulijk (2019) further demonstrated increased close call classification accuracy using an 'interactive learning' process. In this process, a machine learning algorithm is trained using the input of a trained analyst. This method enabled an increase in classification accuracy from the 0-60% of the previous NLP method to up to 98%.

These examples are illustrative of work being carried out in many different sectors. Many methods have been demonstrated be effective in processing, indexing, categorising and otherwise automating the task of learning from text sources. However, the relatively low information density of narrative or free form text in relation to categorical, numerical or other structured data means that text operations are often a bottleneck in these methods. This, coupled with the continuous growth of these reporting systems means there is a mandate for optimisation of this part of the process.

1.1.3 Why Does it have to be fast?

There are a number of potential benefits to improving the performance of text matching processes.

When designing safety or time critical applications, a response is often required within a fixed time frame; faster processing means more information. One major outcome of fast processing is that it allows more to be done in real time. Real time responses allow for interactive processes, and the faster that processing can take place, the more information can be made available to these interactive processes.

Given the expanding size of many incident reporting systems, text processing speed is set to become an increasingly limiting factor for the capacity of the industry to gain full understanding of the content of reports being made.

Another key motivation behind improving the processing speed for text reporting systems is the need for elicitation. As Johnson (2002) explains, text reports can be ambiguous, cursory or otherwise missing information. When the reports pertain to something safety critical, it is often necessary to elicit further information from reporters. Even in a reporting system with a moderate intake volume, unless sufficient staff are available to analyse and respond to reports, there can be a significant time gap between the reports being made and reporters being contacted for further information. As this time gap gets larger it is increasingly likely that key people will forget significant details.

As well as faster processing as a means of keeping pace with growing data volumes, real time responses are necessary for any kind of interactive process utilising such volumes of text. One example already discussed is the interactive learning method proposed by Hughes et al. (2018). For an additional example, consider the concept of interactive reporting. By enabling some interactivity when making reports, the opportunity arises for further enrichment of those reports through suggestions, corrections or additions.

Not only does this potentially improve the quality of incoming reports, it would also allow a large part of the natural language processing burden to be moved out toward the edges of these systems, to the thousands of humans using it to enter reports, reducing the demand for processing after report intake.

1.2 Thesis Statement

Text analysis processes are vital for extracting safety lessons from industry reporting systems. Automatic processes have been shown to be effective in augmenting human analysis but the volume of data in these systems has become so large that it is not possible to process reports in a reasonable time frame. There is a large tool-set available for automated text analysis and processing but what all methods have in common is the need to search, modify or otherwise operate on large and growing volumes of text data. Optimising the way that operations on this text are processed will allow more of this text to be used to inform safety decisions, with a knock on effect to wider railway safety.

To defend this statement, research has been guided by the following questions:

- **How can safety N-gram queries be optimised to deal with large intakes of safety incident reports?**

- **In what ways can unstructured text be matched against known safety risks?**
- **How are N-gram query methods used for text queries, and what are efficient ways to improve the performance?**
- **How can the performance of N-gram queries be optimised in safety applications, and what is needed to demonstrate that the desired optimisation is achieved?**
- **What are the boundaries for this framework and in what ways is it inductive?**

1.3 Optimisation Framework

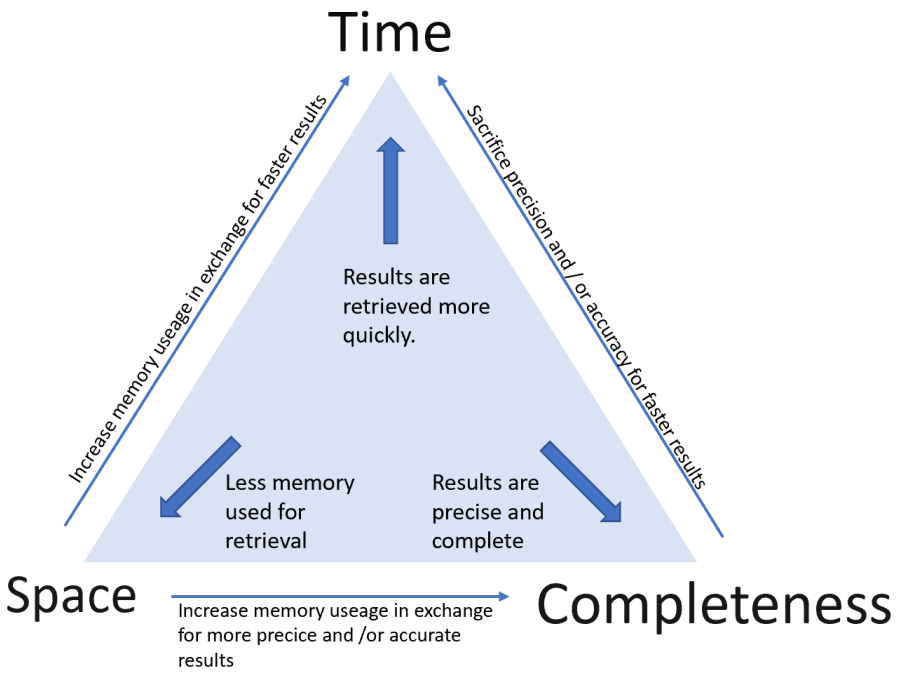

Figure 1.1: Optimisation of matching procedures is a balancing act between, time, space, and completeness

While increased text processing speed is a major factor in improving utilisation of text safety reports it is not the only property which needs to be considered. Russell and Norvig (2002) propose a number of quantities which can be used for comparison of problem solving algorithms, some of which can be applied here: "Time complexity", "Space complexity, and "Completeness"

Time complexity refers to how quickly the task is completed, for example, how long does it take to generate term frequencies for a corpus of incident reports.

Space complexity is the amount of space, either in memory or on disk, that is required in order to complete a task

Finally, completeness refers to the state of results that are returned by the process. Specifically, did all of the tasks get completed. For example, were all instances of a query returned, or were some missed. This also applies to precision. Say for example the location of a term within a corpus is required, does the method tell us exactly where the result is, down to the character, or does it merely indicate which document it was found in.

Taken together, these quantities allow the text optimisation task to be framed as a balance between those three factors, with the specific requirements at any given time dictating which quantities are prioritised. For instance, it might be possible to increase speed by accepting a sacrifice in accuracy or precision. Alternatively, speed could be maintained while achieving high completeness at the cost of increased space complexity.

These are the measures by which the optimisation text queries will be evaluated.

In the following chapter, a review of relevant literature has been carried out, guided by these questions. Several specific examples of automatic information extraction from narrative and free form text sources are presented as well as examples of text reporting systems, from the railway industry and elsewhere. Text matching methods are investigated and compared. In Chapter 3, a novel text matching method for processing of safety related text operations is described and a procedure is established for the comparison of this method and other widely used methods. In Chapter 4 The results of the comparison procedures are presented. Chapter 5 discusses how these methods serve the statement and questions offered here. In the final Chapters, conclusions are offered and some consideration is given to future developments and improvements to the text matching process. Much of the prior work discussed here regarding learning from narrative text sources use various NLP, classification and processing methods. The focus of this document is the optimisation of the underlying structure of the text which these methods operate on. As such while outlines are given, the detailed mechanics behind these methods (aside from details regarding the ways in which they process and access text) are considered outside the scope of this document. Where required, the book "Speech and Language Processing" (Jurafsky & Martin, 2020) provides detailed explanations of the inner workings of the algorithms typically used for these purposes.

Chapter 2

Literature Review

For some time now, Unstructured text sources have become widely accepted as a valuable source of information across a wide range of domains and applications. In particular, the value of narrative incident reports have been found to contain useful safety information that is not captured by other means (Bird and Germain (1987), S. Jones et al. (1999), Dillon and Tinsley (2008), Bliss, Rice et al. 2014). In the following section, a number of applications which attempt to learn from such narrative text sources are reviewed.

2.1 Text Use for Safety Learning

2.1.1 Road

Text analysis methods have been applied to different sources relating to road safety and operations. Pereira, Rodrigues, and Ben-Akiva (2013) present a method of predicting the duration of a road incident using topic modelling on 2 years' worth of free text road accident reports. Features such as day of the week, time of day, and some numerical data were extracted by searching for a list of known terms. This reduces reports down to a smaller dictionary of relevant terms, e.g. locations, distances, and times. Abbreviations are likewise replaced using a known list. Negation is managed by concatenating negating words with the terms they negate, for example 'no injuries' becomes 'noinjuries'. Stop words, or words which are deemed to have no semantic relevance are likewise searched for and removed using a list of known examples. Words are reduced to stems. Stems are root of a word from which all forms of that word are derived. For example, the word injuries, injured and injuring all share the stem 'injur', thus 'serious injuries' when stemmed becomes 'serious injur'. Stemming, negation, abbreviation replacement and locating of known features all require some variation of a search and replace process involving reading through text until a match is located. From each of these processed reports, a bag of words vector

is created from the now reduced vocabulary containing an index of each word and its frequency.

A bag of words vector is a representation of text which ignores the order of words and instead records the frequency of each word. (Jurafsky & Martin, 2020, Chapter 4.1). As illustrated in fig 2.1, the first column of the bag of words vector contains the words, and the second column contains frequencies.

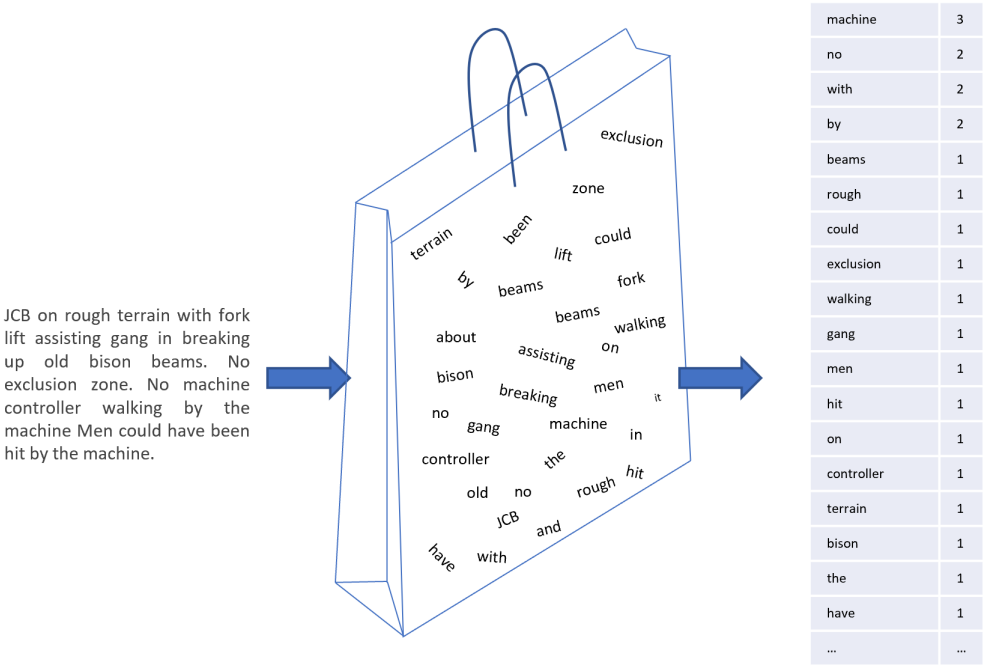

Figure 2.1: Bag of words representation of a railway close call report

The reports in their bag of words representation are then used to perform latent dirichlet allocation (LDA). LDA is a method for inducing sets of related words from text using unsupervised learning (Jurafsky & Martin, 2020, Chapter 6.1). The groups produced by LDA are then manually assigned a topic based on the words they contain. A prediction model is built to identify these terms in new reports in order to assign topics to them. The prediction is done using a suite of regression and neural network methods. It was found that the median error of predictions was decreased by 35% when compared to methods which did not use text analysis.

Similar techniques have been applied in order to identify road traffic events in social media posts. D'Andrea, Ducange, Lazzerini, and Marcelloni (2015) describe a text classification process which works on text submitted to 'Twitter' in real time. The benefit of using a data source such as twitter is that eyewitnesses can make reports

which are distributed in real time as an event unfolds. However, the difficulties in extracting useful information from unstructured text are reiterated, particularly in this case where the content is informal and subject to length restrictions. Abbreviations, misspellings, grammatical errors and brevity are some of the qualities of this text which make event detection difficult. To mitigate these issues it is necessary to apply NLP processes to standardise and clean up text. Pre-processing tasks include, removal of extraneous data using regular expressions, and transformation to lower case. Tokenisation splits the resulting strings in to words, and punctuation and symbols are removed. Stop word tokens are removed at this stage using a publicly available list. Words are reduced to stems. Supervised learning methods are used to determine relevant tokens which are weighted and used for classification. These methods were shown to effective at distinguishing tweets which were related to traffic events from those which were not, as well as determining the existence of any external causal factors relating to particular event.

2.1.2 Medical

There has been increased interest in the past decade or so in using the narrative patient reports contained within electronic patient records as a diagnostic aid. Chase, Mitrani, Lu, and Fulgieri (2017) present a method for early recognition of Multiple Sclerosis prior to a formal diagnosis. In this study a list of 1000 MS related terms were used to query patient records from a sample of patients, with a sub group known to have MS. This list is made up of words and Unified Medical Language System (UMLS) terms. UMLS terms are a controlled vocabulary which describe biomedical concepts and can be words or phrases (N-grams) (Lindberg, Humphreys, & McCray, 1993). N-grams are a contiguous sequence of 'n' items which are explained in more detail in chapter 2.5.1.2. Patient records were queried for these terms and a naïve bayes classification process was used to determine whether a record was MS related or not. Naive Bayes, as described by (Jurafsky & Martin, 2020, Chapter 4.1) is a probabilistic classifier. Naive Bayes makes the assumption that features, in this case tokens from the text, do not interact (hence, naive). Classifiers are trained using the bag of words representation shown in fig 2.1 and for each new document, the most likely class is chosen based on the product of the *prior probability* of the class and the *likelehood* of the document. These probabilities are calculated using the frequency of terms for documents in each class and in the document to be classified.

This process resulted in a high accuracy in identifying patients known to have MS when using records from these patients which were made after diagnosis. However, the process was also able to identify 40% of positive MS cases from patient notes

made up to two years prior to diagnosis, indicating that methods like this could be very useful as a diagnostic aid.

Bejan, Xia, Vanderwende, Wurfel, and Yetisgen-Yildiz (2012) describe a similar process for identifying Pneumonia from patient records. Again, a list of terms is used to identify relevant reports. In this case, relevant terms are determined through a weighting process. Using a subset of reports, a 'training set', all possible word unigrams, word bigrams and UMLS concepts are extracted. Statistical hypothesis testing was used the weight how informative each of these terms is in identifying pneumonia. The result of this is that terms appearing most often in reports known to relate to pneumonia will be given the highest weight. A 'feature vector' is built from the intersection of the ranked terms and the terms that appear in a given record. This feature vector resembles a bag of words vector in that it consists of a list of words and frequencies, but its length is reduced as it only contains the terms from a document that also exist in the ranked term list. The key finding is that reducing the feature space in this way, I.E. only retaining the most relevant terms, resulted in significantly improved accuracy over using the full text.

A method for detecting injuries follwing inpatient falls based on medical records was developed by Toyabe (2012). A variety of different text sources are used, and text sources are processed to extract morphemes (similar to stemming) and tag parts of speech. A rule based process is used to determine whether text refers to a fall incident based on the appearance of a list of fall related terms. As well as the terms themselves, the grammatical relation to other parts of the text are also checked, as often, words earlier in a sentence will change the meaning of those later on. As with Chase et al. (2017) and Bejan et al. (2012) the list of fall related terms is determined by generating a ranking from a training set of labelled reports.

Another application for text in patient reports is phenotyping. Shivade et al. (2014) review a number of methods for identifying groups of patients who all share a phenotype related to serious diseases, for example, cancer, diabetes or heart failure. In this case, methods for using both structured patient data and unstructured narrative text is used to group patients. Text is pre-processed using NLP techniques and domain specific knowledge is used to curate a list of keywords relating to each cohort. These lists are used to identify relevant records. It is noted that by taking a more holistic approach to varied data utilisation, rather than rigid rule based approaches using structured data only, is of benefit to phenotype classification.

2.1.3 Aviation

A primary motivation for using text data in the aviation industry (and many others) is improved safety. Tanguy, Tulechki, Urieli, Hermann, and Raynal (2016) reiterate

some of the difficulties which are usually encountered when utilising free form and unstructured text sources. The use of plain language results in many different ways of saying the same thing. In many industries, narrative reports will contain a high amount of domain specific language or acronyms. Several approaches are outlined below.

Classification of text reports using machine learning is a common approach and is straightforward to implement but is described as poorly suited for constant change and emerging threats. Machine learning can only make useful predictions when new data are similar in content to the training data. Any new trends would therefore not be represented, meaning that these models need to be retrained on a regular basis to account for these trends, and their utility for detecting new threats is low. Advanced NLP techniques are used to extract useful information from text. Tanguy et al. (2016) outline some of the processes common to text utilisation approaches in the aviation industry. Hand written rules are used to replace known variations of the same concepts with a standard form. Words, stems, character bi-grams, tri-grams, and quad-grams, and stem bigrams and trigrams are derived from the text. Vector representations are built from reports representing the relative frequency of the derived terms. A classification process using a group of binary classifiers (one for each group of interest) is described. Each classifier gives a true or false value for each text example indicating whether a record belongs to the associated group. By comparing the classification accuracy when using different types of terms, it was found that the best classifications were made when processing reports in to stems, and bi-grams of stems. Tanguy et al. (2016) also explain an alternative approach using an interactive learning process. In this approach, machine learning tools are used to make efficient use of a domain experts time. In supervised learning, labelled data are required to allow for patterns to be detected and be ascribed to the correct classes. Labelling of data is often a time consuming and expensive task. In the interactive learning process, rather than labelling a full training set, an outline of the desired aspect of an incident by querying for relevant terms, then selecting or rejecting the returned examples. This set is then used as a training set for a supervised learning pass. Using error margin or probabilistic confidence scores, the expert is then shown the borderline cases returned by the supervised learning process for labelling. This process is iterated, this improving accuracy with each pass. This significantly reduces the labour required for obtaining training samples as far fewer examples require manual labelling.

Peladeau and Stovall (2005) produced a report explaining the functionality provided by the "WordStat" suite of text mining tools for statistical content analysis of airline safety reports. This tool provides many of the functions commonly used for information

extraction from text. The tool is used to process flight crew irregularity reports (FCIR) and traffic collision avoidance system reports and extract statistical information from the text. An FCIR is created every time a pilot experiences an “incident or abnormal operation event”. A TCAS report is made by flight crew. for every collision avoidance event. Various pre-processing steps are applied the the text in the reports. Amongst others, categorical information is identified and tagged, spelling variations and mistakes are fixed. Stems and Lemma are identified and a list of stop words is created by manually augmenting a standard list to ensure that domain specific terms and abbreviations are not included. These stop words are then removed from the text. A ‘categorisation dictionary’ is created which contains words, abbreviations, and word N-grams. Text is then queried against this dictionary to collect relative frequencies of terms in context to create a taxonomy which can be used for classification. Some ambiguity resolution is possible by considering words and terms which are co-located with any given term. Co-location can also be used to cluster terms by meaning regarding terms as similar if the list of terms they commonly appear alongside have a high similarity.

2.1.4 Rail

Hughes, Figueres-Esteban, and Gulijk (2016) Describe an automated text mining approach for text records in the close call system. This system is used by GB railways to report hazardous situations with the potential to lead to an accident were it not spotted. An information extraction process is described with the goal being to find reports which have relevance to a number of known risk types with the aim ultimately being to answer high level questions such as ”Do trespasses take place at certain times of the day or do they take place with equal frequency throughout a 24-hour period?”. The text processing steps used are described. Text is first cleaned by searching for instances of specific unwanted text and either removes it or replaces it with the correct information. Before tokenising the text into words, a list of phrases and acronyms known to have specific meanings are located in the text and replaced with single tags so that once tokenisation takes place units of text with known specific meanings are contained in a single token. For example ”British transport police” becomes ”BRITISH-TRANSPORT-POLICE-”. Known synonyms are also located and replaced with a single tag, so that same tag also replaces ”BTP”, BTpolice”, ”B.T.P.”, etc. As is common with free form unstructured text, the close call text contains numerous spelling errors. These were corrected using existing spell check software.

2.2 Incident Reporting

Incident reporting systems are used across a wide range of diverse domains for use in safety and risk management.

2.2.1 Aviation Safety Reporting System

The Aviation Safety Reporting system is one of the oldest incident reporting systems dealing with text data and has been in operation since 1976 NASA (2020). Early reports were written using an imposed writing style which incorporated a specific structure and consisted of standardised terminology and acronyms. In the years since this style has relaxed and reports currently consist of an unstructured narrative description of events alongside structured descriptors created during report intake containing features such as location, time, weather, and the people and equipment involved. The reports are submitted to a categorisation process performed by experts which add further structured information (Tanguy et al., 2016). These structured descriptors, derived from the text are strictly organised into a two level taxonomy describing a number of related entities. For example the *Aircraft* entity would contain information relating to the aircraft in question, for example, the model, operator, flight plan, etc. The *Person* entity denotes the people involved, and so on. Encoding of this information is performed by analysts on reports which they identify as requiring further analysis (NASA, 2020).

ASRS is used by the Federal Aviation Authority in the US, but similar reports are collected across the wider aviation sector. One of the barriers to more widespread sharing of incident data amongst industry stakeholders is a lack of standardisation of data collection and processing pipelines. The European Co-ordination Centre for Accident and Incident Reporting (ECCAIRS) is a co-ordinated effort amongst transport and safety investigation authorities across Europe to standardise reporting processes. Initially focusing on information relating to Aviation, the system has since expanded to include other transport authorities such as maritime and rail. Like ASRS, a taxonomy is used to classify and encode important information to enable further analysis (Tanguy et al., 2016)

2.2.2 MedWatch

MedWatch is a national voluntary reporting system provided by the US Food and Drug Administration. The system allows healthcare staff to make reports regarding adverse effects involving medical devices (Johnson, 2002).

2.2.3 Incident reporting for GB Rail

There are a number of methods and procedures for the reporting of safety related incidents in the railway industry. Some of these, such as TRUST, have been in use for several decades and have remained largely unchanged, while others, such as SMIS see regular improvements and updates as available technology improves and new methods arise for processing and analysis, particularly for text data.

SMIS, the Safety Management Intelligence system, is both a set of safety management utilities, and a database for the recording of safety related events occurring on the railway Network in Great Britain (Brewer, 2017). It is managed by the Railway Safety and Standards Board (RSSB) on behalf of the railway network.

The database exits to record any event or incident which pertains to the safety of the railway network. This would include any fatalities or injuries to staff, passengers, or members of the public; as well as faults or damage to railway equipment, signals passed at danger (SPADs) or derailments (Brewer, 2017).

This system replaced another system, also named SMIS (The Safety Management Information System), with the aim of putting new developments in reporting technology into use.

TRUST, which stands for Train Running Under system TOPS (TOPS itself stands for Total Operations Processing System), is a computer system which records train operational data (Network Rail, 2016). The system tracks the following information about trains on the network (East Coast, 2011):

- Train Activation – Generated when before a train is due to run, and contains information pertaining to that particular train, e.g. Vehicle type, Train ID and Schedule.
- Train Movement – Generated whenever a train Arrives at, Departs From, or passes a location.
- Train Cancellation – Generated when a journey is not, or will not be, completed.
- Train Reinstatement – Generated when previously cancelled trains return into service.
- Change of Origin – Generated when a train will not start from the origin on its schedule.
- Change of Identity – Generated when a freight train when its class changes after being activated. E.g. when its wagons are removed.

The TRUST system can be used by operators for several tasks. One example of a query type possible using TRUST is a TRJA enquiry, which provides a list of all trains currently expected at a given location, as well as the current train running (delays). Another example is a TRJE enquiry, or train delay enquiry. This will show the current running delay recorded by the system between two given locations, giving some indication of train performance, providing that the delay information is up to date. The enquiry can also return all delays in the given location which are attributable to a particular train. The system contains a mix of numerical and text data and is populated both by automatic systems and by manual input. For example, following a delay, the late running would trigger some information to be entered automatically from the TOPS system, only to be overwritten manually later, when more accurate information is available. This design means that the data retrieved from it may not always be representative of a real life situation, as data is collected using two contrasting and potentially contradictory methods.

TRUST is typically used for delay attribution, that is, what were the causes of a particular delay incident. For each incident, as well as the structured fields already described, also included is a free text report describing the incident and any relevant details and involved parties (Board, 2016).

The close call system is a reporting system designed to allow the GB rail industry to record and manage "Close Calls", which are defined by S. Jones et al. (1999) as "a hazardous situation where the event sequence could lead to an accident if it had not been interrupted by a planned intervention or by random event". Similar data collected in other industries such as aviation, healthcare and nuclear power, has proven valuable in the past, and Close Calls are becoming an increasingly accepted source of safety lessons (Bird & Germain, 1987) (Gnoni, Andriulo, Maggio, & Nardone, 2013) (Dillon & Tinsley, 2008).

The data contained in the close call system is described in detail by Hughes et al. 2014. To date, it consists of around one million free form text reports pertaining to scenarios on the railway network in Great Britain, scenarios which could potentially lead to an accident, but did not. The data is a mixture of structured data and unstructured free-form text.

The Confidential Incident Reporting and Analysis system (CIRAS) is a safety reporting and response system which operates independently of the railway network (Baker, 2018). Workers from any member organisation can anonymously report health, safety, security, and environmental concerns or incidents (Baker, 2018). Member organisations are involved in a variety of industries, including rail, across the UK.

The guarantee of anonymity is intended to allow reporting concerns without fear of facing personal consequences for making the report, going some way to removing a barrier to the collection of safety information.

As well as recording incident reports, the service also investigates these reports, with the aim of resolving any concerns or consequences highlighted within. Any safety lessons learned from these reports and from subsequent investigations are made available to member organisations, for use in their own safety management procedures.

As with SMIS and Close Call, the CIRAS database contains vast amounts of safety related reports which currently require a large about of manpower to make sense of.

The Railway Accident and Investigation Branch (RAIB) is and independent body set up by the UK government to investigate railway accidents. While not an incident database itself, the reports it produces often contain valuable safety lessons and recommendations, which inform the railway industry how to operate safely. Reports are produced after railway accidents and incidents and any findings, evidence, or recommendations can be shared with the railway industry.

S. Jones et al. (1999) and Dillon and Tinsley (2008) make the case that there are safety lessons to be learned from analysis of 'near-miss' incidents of the type recorded in the Close Call system. However, systems such as SMIS, Close Call, TRUST and CIRAS still require large amounts of manpower to enable useful action as a result of the collected information, and response time is in the scale of months.

2.3 Data

Extraction of actionable information from large stores of unstructured or uncategorised (or unreliably categorised) text data is not necessarily straightforward. The following two examples of text data, and the methods used to mine them for information, offer some insight into the issues with working with this kind of data.

2.3.1 Event Logs

One potential source of data for extraction of information are event or error logs. These are typically generated by an automatic process and thus the structure and types of information contained are predictable (Fig. 2.2).

The volume of data generated as logs is such that a number of utilities have been developed specifically for managing and analysing event and error logs. Microsoft Event Viewer, a part of the Windows operating system, collects logs from various

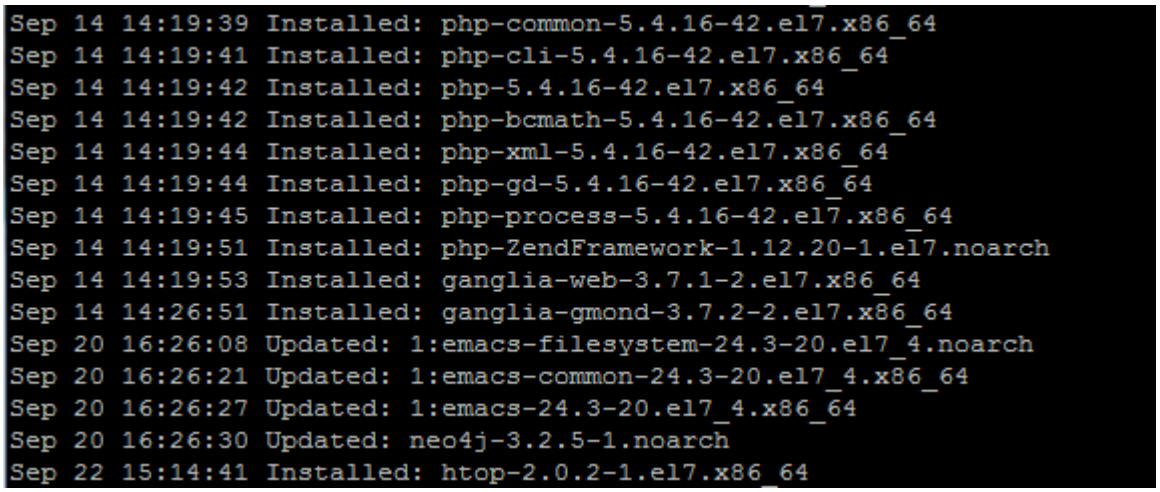

Figure 2.2: An example of an event log from the Yum package manager, showing structured fields

applications and services running on a system and categorise the events contained based on level (Error, Warning, Information), source, number of occurrences, and time. Some utilities, such as Log4j, take the role of both generating, and managing logs (Apache, 2018).

With log files of this volume it can be difficult to locate and asses logs which have particular significance. Fig. 2 is an example of potential security issues on a public facing server, which may have gone unnoticed in very large log files, or amongst a large volume of log sources.

Work into the analysis of event logs has some history. As event logs always report the same event in an identical fashion, it is possible to build analytical processes which exploit this structure, however techniques built this way would not be applicable in the same way to unstructured text where the form of an event report cannot usually be predicted. Lee, Iyer, and Tang (1991) describe a method whereby the reliability of a system can be evaluated from raw event logs by using time domain filtering, probability distributions, and other statistical techniques to determine the frequency and relevance of certain events.

A system able to extract information from logs automatically, with no prior knowledge of the structure of the logs would be particularly useful, given the differing formats of logs from various sources, and the potential large volume of sources. However, being still reliant on these underlying structures limits those systems utility for unstructured text. Xu, Huang, Fox, Patterson, and Jordan (2009) present one such system, capable of the production of more informative messages from a large number of aggregated logs. This system uses machine learning techniques to detect anomalies and suggest a cause or generate a decision tree.

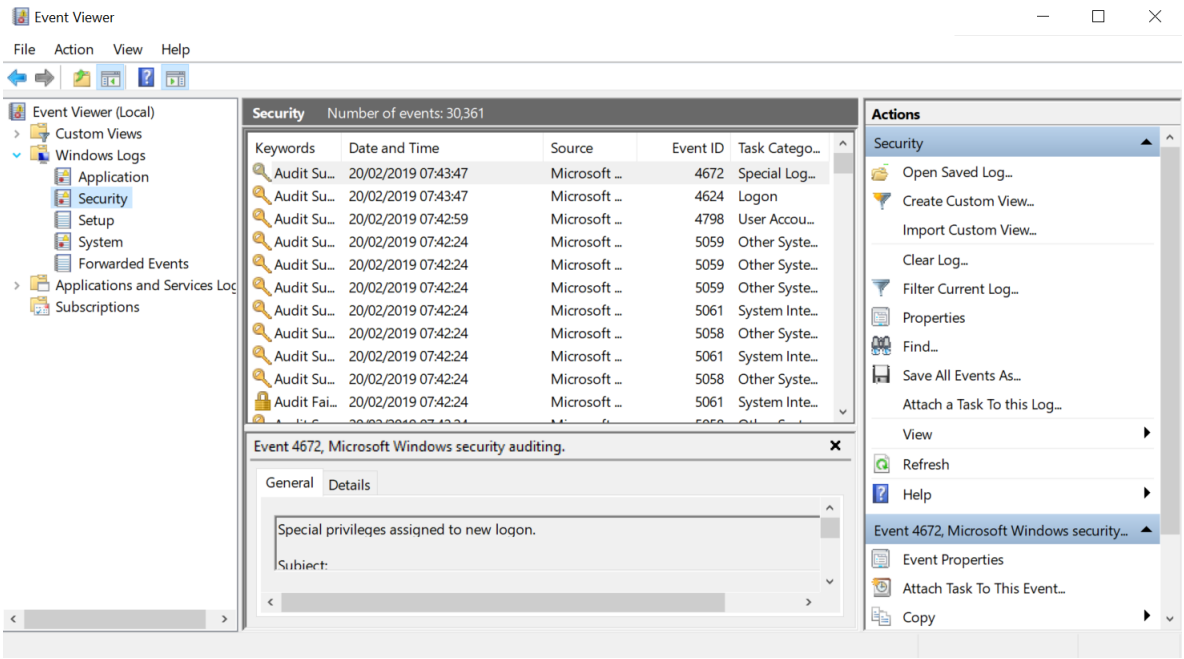

Figure 2.3: An example of aggregated logs in Windows Event Viewer

These studies illustrate that extracting 'buried' information from text based event logs is a non-trivial task. The techniques shown here are of interest in the extraction of information from structured logs, particularly when they occur in large volumes. Additionally, the use of string matching to find relevant adjacent data could be applicable to other kinds of text records.

2.3.2 "Near Miss" Reports and Interpretation Methods

A large form of unstructured safety text data comes in the form of near-miss reports made to the Network Rail Close Call system. Gnoni et al. (2013) define close calls as "a hazardous situation where the event sequence could lead to an accident if it had not been interrupted by a planned intervention or by random event". These reports in particular are collected from staff working on the railway network in Great Britain, in order to document events, scenarios or anomalies that fit the Close Call description.

Many industries, such as Aviation, and Nuclear Power, are turning to close calls to give insight into ways in which safety procedures can be improved. However, the way these Close Calls are reported for the railway network in Britain, I.E. A mixture of structured and unstructured text and images, presents a challenge for analysis. While some headway is being made in this regard, the techniques used require extensive computation time. Hughes, Figures, and Gulijk (2014) describe a natural language

processing (NLP) approach, the aim of which is to extract meaning, and therefore and potential safety lessons, from the unstructured, free-form text portion of the reports. This freeform text would otherwise be to voluminous to be read and understood by human readers who could make use of lessons contained within, so an automated process is desirable. These techniques, however, require time to compute. If the process could be completed closer to real time, this would enable close to real time feedback, perhaps as the report itself is being made, enabling an interactive reporting process.

The data is a mixture of structured data and unstructured free-form text. An example report is shown in Fig. 2.1.

Table 2.1: Structure of a Close Call report

Field	Field Name	Data Type	Example
1	Reported Date	Date and Time	10/02/2015 05:42
2	Describe the Close Call Event	Unstructured Text	reported by A global operative Lap 903 - loc 40 (SWB 084U005) cable damage seen as cable emerge from the trough route SSL/BCM to rectify asap
3	What To Do About It	Unstructured Text	photos taken, reported the SSL SHEQ
8	Risk Ranking	Predefined List	LOW

2.4 Time Cost

While ample discussion can be found regarding the techniques used to extract information form unstructured test sources, processing time is often not considered. Numerous studies, such as those reviewed here, demonstrate the value in mining unstructured text sources for safety related purposes, but it can be noted that these studies often involve small sample sizes relative to the total available data and as such, processing time is often not a main concern. For example, the pneumonia identification method produced by Bejan et al. (2012) and the MS identification method covered by Chase et al. (2017), use samples of 326 and 2999 patients respectively from a database covering millions. Likewise, the close call classification method developed by Hughes et al. (2018) uses a sample of reports from a database of hundreds of thousands.

It has been apparent for some time that the sheer volume of text available for analysis presents a challenge for timely feedback, particularly in the safety domain, where the cost of missed information can be high. Johnson (2002) noted in 2002 that

the numerous reporting systems used across domains such as aviation, medical, and rail had become "victims of their own success" and that the size of these systems was becoming unmanageable. For example, in 2002 the ASRS received around 2600 reports per month and contained around half a million reports in total. Similarly, MedWatch contained over 700000 reports. This volume has continued to increase in the years since. For example, when the ASRS was first started, report intake was around 400 per month, it is now averaging 435 reports per working day. In 2019, the ASRS was receiving close to 9000 reports per month, and in that year alone the report intake was 107,879 (NASA, 2020). MedWatch has gone from containing 700000 reports in total in 2002, to having an intake of over one million per year by 2017 (28). This pattern is mirrored across industries. The close call system used by GB railways had a report intake of around 200000 per year as of 2018 (hughes et al, 2018).

Part of the reason for this is increased participation, but an additional factor has been the consolidation of a number of local reporting processes into larger central structures. These local systems have the potential benefit of allowing contributors to directly observe the impact of their reporting, and systems can be managed by people who are able to directly observe the environment to which the reports pertain to. However, a major drawback to this approach is that lessons learned in one reporting group are not necessarily propagated to others which might encounter the same issues. There is also the danger that the limited scope of a local reporting system might encounter what appear to be isolated incidents when in fact they may be part of a larger pattern which more centralised reporting would make apparent.

Johnson (2002) identifies two main tasks that users of large-scale incident reporting systems wish to achieve. The first is to produce statistics relating to how failures are affected by management actions. The second is to identify trends that need addressing. Some possible solutions to the expanding volume of text based reports, and what is described as the "almost contradictory" requirements for the two tasks. One approach is to use using a reporting paradigm which is inherently structured. A number of industries use relational databases for incident reporting. This is ideal for extracting statistical information as incidents are strongly typed. A major downside of this approach is that these systems are often built in an ad-hoc manner from commercial database products and there is often little regard for standardisation across domains or even within the same organisation, this can result in several different ways of describing the same thing, which can be very difficult to co-ordinate (Lainoff, 1999).

Another drawback of this approach is the complexity of queries. To query relational databases, all but the most basic of queries can run multiple lines of, for example, SQL

and the possibility for error is high. Johnson states that even professional software engineers can fail to retrieve correctly indexed records when complex query languages such as this are required. Another approach is to use the various information retrieval processes that have been developed to retrieve relevant information from massive sources that have arisen from the near ubiquitous use of the internet. As can be seen from some of the studies outlined in earlier sections, the use of full text indexing, pre-processing and categorisation processes are now seeing widespread adoption in applications using free-text sources. There are numerous benefits to this approach. Depending on the methods used, building indexes from the text can improve query response time dramatically, sometimes to speeds rivalling the relational databases. Indexing full text instead of requiring structured input allows for more freedom of reporting while a structured database can only contain the fields which it has been designed for, leaving little flexibility for unanticipated events.

From full text indexes which provide exact text matching, retrieval methods can be built to provide relevant information in response to queries. A successful retrieval system would need to return reports based on the concepts in the query, not just the exact words. Say for example a user wanted to retrieve reports regarding "vegetation obscuring a signal", a report containing "Tree has grown in front of W172" would be a relevant candidate, as even though it shares no words with the query, the concepts are the same. There are a number of ways to do this. One, as used in the accident duration predictor developed by Pereira et al. (2013) and for classifying reports as described by Tanguy et al. (2016), is topic modelling. Topic modelling uses a vector space representation of reports and terms are weighted based on their relative frequency within a document. These weights can the be used either to train classifiers using some machine learning approach, or to directly build a relevance score in relation to a given query. Concepts can also be identified based on co-located terms with some success. This approach is one of the tools explored by Peladeau and Stovall (2005) for text mining of aviation safety data. The assumption here is that two words, terms, stems, etc that appear alongside the same other terms often enough then they might refer to the same concept. As an example, the word "vegetation" might often appear alongside the terms like "Leaf, green, growth, vision, obscured", by searching for other words which also often appear alongside these ones perhaps the word "Tree" appears often enough that "Tree" and "vegetation" can be assumed to refer to the same concept. Other studies, such as the aforementioned MS detection method (Chase et al., 2017), phenotype classification (Shivade et al., 2014), incident duration prediction (Pereira et al., 2013), and many others, use machine learning techniques to classify reports into categories based on their content.

What is common amongst all information retrieval approaches is the need for an indexed representation of the text to enable any of the processes that follow, be they approximate or concept matching, or machine learning processes based on term frequencies and weighting schemes.

2.5 Text Processing

From the various applications of text mining techniques discussed earlier, a number of text processing and NLP tasks can be identified which have been put to use for safety learning across a wide range of sectors. These techniques are summarised in table 2.2.

In this section these techniques, and the ways in which they interact with the source text, are discussed in more detail.

2.5.1 NLP

Natural language processing is a key component in enabling computer assisted understanding of free text reports. All of the studies, developments, and reviews discussed so far have used at least some NLP concepts.

Natural Language Processing as a field of computer science refers to process of modelling text and speech in a way that allows computers to infer meaning and concepts, with he aim of allowing such systems to interact with natural human communication in a way more meaningful than simple analysis Miner et al. (2012).

Early NLP techniques relied on human produced lists of rules, based on grammatical standards and linguistic expertise. According to (Miner et al., 2012, Chapter 1), this has worked well when the text in question sticks to these rules and remains predictable, but unstructured or informal texts often present more of a challenge. The development of machine learning in the past few decades has allowed research into machine generated models based on vast corpora of real world text.

This shift toward machine learning and artificial intelligence has resulted in differing techniques. The methods and and related concepts differ depending on the approach used and, according to (Miner et al., 2012, Chapter 1), machine assisted NLP has lead to the development of powerful text parsers, which spilt text into tokens; and stemming algorithms, which reduce words to their base or roots, and move towards being less domain specific.

Table 2.2: Summary of text processing and NLP methods used by a selection of studies using safety text

Publication	Author(s)	Methods
Early recognition of multiple sclerosis using natural language processing of the electronic health record	Chase et al. (2017)	Tokenisation, Statistical analysis of words and word N-grams, term search, supervised learning
A review of approaches to identifying patient phenotype cohorts using electronic health records	Shivade et al. (2014)	Tokenisation, Statistical Analysis of words and word N-Grams, supervised learning
Detecting inpatient falls by using natural language processing of electronic medical records	Toyabe (2012)	Tokenisation, Tagging, Term Search, Supervised Learning
Pneumonia identification using statistical feature selection	Bejan et al. (2012)	Tokenisation, Statistical Analysis of words and word N-Grams, Term Search
Natural Language Processing for aviation safety reports: from classification to interactive analysis	Tanguy et al. (2016)	Tokenisation, Statistical Analysis of words, N-Grams, stems, and stem N-Grams Stemming, Normalising, Supervised Learning, Interactive Learning, Topic Modelling
Application of Statistical Content Analysis Text Mining to Airline Safety Reports	Peladeau and Stovall (2005)	Tokenisation, Statistical Analysis of words and word N-Grams, Normalising, Tagging
Text analysis in incident duration prediction	Pereira et al. (2013)	Tokenisation, Statistical Analysis of words and word N-Grams, Tagging, Topic Modelling, Supervised Learning
Real-Time Detection of Traffic from Twitter Stream Analysis	D'Andrea et al. (2015)	Tokenisation, Stemming, Statistical Analysis of words and word N-Grams, Supervised Learning
Crime Pattern Analysis through Text Mining	Ananyan (2004)	Tokenisation, Normalisation, Statistical Analysis of words and word N-Grams
From free-text to structured safety management: Introduction of a semi-automated classification method of railway hazard reports to elements on a bow-tie diagram	Hughes et al. (2018)	Tokenisation, Normalisation, Statistical Analysis of words and word N-Grams

2.5.1.1 Tokenisation

NLP techniques typically begin with a number of preprocessing steps. Text must first be delineated into characters, words, and sentences; in order to move from a stream of bits, into something resembling natural text (or speech in the case of audio) with robustly defined linguistic units.

In the case of text it is first necessary to determine the character encoding. In text files this is often included in the file header, but in cases where this is not available or inaccurate, an identification process may be necessary Miner et al. (2012).

It is also usually important to determine the language used in the text, not only to ensure that the proper character encodings have been used, but to ensure that the text is properly delineated into linguistic units. In English, for example, words are delimited by spaces, and sentences are delimited by the full stop or period. The importance of this step becomes clear when considering that other languages do not use this structure, or even the same kinds of linguistic units (Feldman & Sanger, 2006).

Text is typically split into tokens representing words, multi word terms (N-grams), fixed length character N-grams, or word roots either through stemming or lemmatisation.

2.5.1.2 N-grams

An N-gram is a sequence of items from text or speech of length 'n'. N-grams are commonly used in NLP and probabalistic linguistic analysis due to the fact that they include contextual information exceeding that of simple fixed length or fixed word tokens, and are relatively simple to implement. Fig. 2.4 demonstrates how n-grams are formed from an input sentence when 'n' denotes the number of words.

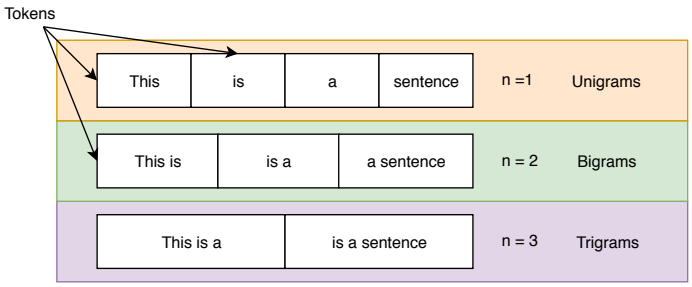

Figure 2.4: Formation of n-grams

Storing n-grams rather than single word tokens has a number of advantages for retrieval and categorisation purposes. By including sequences of words in tokens rather than individual words alone allows only searching tokens representing only those n-grams matching the length of the query term (Newall & Van Gulijk, 2019).

In addition, tokenising n-grams allows the capture of some words which are missed my the tokeniser without implementing complex detection rules. For example, acronyms. If the only tokens captured are single words, and the tokeniser simply splits tokens based on delimiters (such as space and full stop) then acronyms, e.g. T.O.P.S. will not be captured as a single token. This can be mitigated by adding routines in the tokeniser to detect these circumstances, but if n-grams are also captured as tokens, this is unnecessary.

As evident from numerous applications of text analysis, such as the early detection method for MS Chase et al. (2017), and the term extraction processes described by Tanguy et al. (2016), N-Grams, both of characters and of words, have been found to be a useful tool for computer aided learning from text sources. Several studies have been made into their effectiveness as an information retrieval tool. Mayfield and McNamee (1998) explore the relative value of N-Grams compared to words for information retrieval accuracy. Documents were indexed by terms weighted using TF/IDF and Okapi (a metric derived from TF/IDF which ascribes larger weight to longer documents). Queries were made using both words and N-Grams, and the accuracy of the retrieved results was measured. In this case, the text has been pre-processed to drop it into all lower case, and common words such as 'of' and 'with' are replaced with single characters. This means that character N-grams have increased scope in terms of the amount of words they can span. An example given is that the phrase "statue of liberty" might produce the 5-gram: "e_ϕ_l" (spaces are also characters), spanning 3 words, as the word 'of' has been replaced by 'ϕ'. In this particular retrieval task, it was found that 5-Grams were around as effective as words for retrieving relevant documents.

Similarly Miao, Keselj, and Milios (2005) demonstrated that a vector space model using the most common character N-Grams in a document (instead of all the words in it), was more effective than words and terms for document clustering once a suitable distance measure was derived.

Baeza-Yates (2000) demonstrate that N-grams can be efficiently hashed using a rolling hash by exploiting the fact that N-grams overlap, meaning that part of any N-gram was already partially hashed when the previous N-gram was hashed. A key observation from this work is that, while the potential N-gram space for any given corpus can be very large, most of these possible N-grams are very unlikely to appear in real text, so index size can be reduced significantly by only indexing observed N-grams.

2.5.1.3 Tagging and Normalisation

Tagging is a process whereby some tokens are found in the text, and replaced with different ones. Tagging text first involves searching the text, to find all instances of the words to be tagged, and then replacing with the new token as appropriate. There are a number of reasons for doing this. One is to remove tokens which are superfluous for the task at hand, for example, categorical data which has already been extracted into a structure, such as dates and locations. An example of tagging used in this way can be found in the process used by Hughes et al. (2016) to remove words related to locations by searching though text for tokens matching list of location names and replacing all of them with a single token. Likewise, many reports in the close call database contain artefacts as a result if the intake process, such as parts of html script which do not make up part of the report. Where the structure of these artifacts is known they can be removed in a similar fashion. A similar process can be used to replace all instances non standard spellings and mistakes with a single correct one, either by using a list of known mistakes and variants or by using a metric such as edit distance to determine what the most likely intended word was for any cases which fail a dictionary check. Some applications also replace common words such as 'the', 'of', and 'and' with a single character in order to expand the meaning contained in fixed length N-grams, as seen in Mayfield and McNamee (1998). In other cases, these words are simply removed instead as they are commonly assumed not to add meaningful information to a report. Another reason for tagging text is to perform part of speech tagging. Part of speech tagging adds semantic information to text by labelling words as verbs, nouns, noun-phrases etc. This is done based on word definition and the grammatical context (Toyabe, 2012). The presence of multiple words or terms that mean the same thing can introduce additional complexity to text analysis processes. Tagging is used to replace synonyms (or other tokens deemed to mean the same thing, be they words, acronyms, stems, phrases or some other token) with a single token which reflects the same meaning. This reduces some of the dimensionality when performing tasks such as classification, frequency analysis, and topic modelling.

2.5.1.4 Stemming and Lemmatisation

Stemming and lemmatisation are techniques for reducing the vocabulary of text into root forms of words. Both approaches perform what is ostensibly the same task, but differ in approach. Stemming truncates words to remove common suffixes in order to approximate root words. Lemmatisation on the other hand uses replacement rules to swap various word endings with a single representative. For example, a lemmatiser might replace all instances of the suffixes 'sses' and 'ss' with 'ss' Thus "grasses" and

"grass" would both become simply "grass". Sometimes, a lematisation rule might instead call for the removal of a suffix, for example, the plural -s. Replacing parts of text like this can be carried out in much the same way as tagging. Reducing the vocabulary in this way can make classification simpler,

2.5.1.5 Statistical Analysis

Statistical analysis of text involves calculating the frequency of terms in text documents. Most of the examples discussed do this at some stage or other during analysis. Depending on how the text is indexed this task can be optimised significantly. By constructing an inverted index or vector representation of a document, the frequencies of any of the words it contains can be easily obtained through a lookup without the need to iterate over the text itself. These frequency values, once obtained, can be used in a number of ways. The token selection process used by Hughes et al. (2018) demonstrates that term frequencies can be used to aid in the selection of representative terms. In this case, the existence of the selected terms in a report are used to ascribe multiple labels denoting the threat pathways that a given report pertains to. In other cases, term frequencies measured from groups of labelled data are used to build classification rules

2.5.1.6 Supervised Machine Learning

While human defined rules and models have proven very useful in classifying and retrieving strongly typed, well structured data, challenges arise when working with free form, unstructured text, or text with a high level of domain specificity. Advances in machine learning has allowed research into the automatic generation of these models, derived from the texts themselves (Feldman & Sanger, 2006).

Supervised machine learning models aim to classify texts based by identify patterns in features present in labelled data. These features may be manually defined based on a list of terms known to correlate with the required classes, as is the case with the MS detection method produced by Chase et al. (2017) or they may be derived using statistical analysis. In the statistical approach, the terms with the highest frequency in each category within the labelled data are selected as features. This approach is used by, for example, Shivade et al. (2014) and Bejan et al. (2012).

Many studies use some form of supervised machine learning to classify text reports. This can be very useful for identifying known events or threats from incoming reports. But the fact that supervised models are trained on known, labelled data means that their capability to adapt to and identify new kinds of threats is low (Johnson, 2002)

A number of challenges still remain before NLP algorithms are able to extract meaning to the level of a human reader, but when dealing with large volumes of text, NLP techniques are often used to to allow analysis in a reasonable time-frame.

A major drawback of supervised machine learning methods for mining of text sources related to safety is that, once trained, the models produced are static. supervised models are trained to recognise fealtures in relation to knwn lables While this may be effective in identifying known factors in unstructured sources, identification of new and emerging threats and trends is not possible for this kind of machine classifier (Johnson, 2002).

2.6 Speeding it Up

There are a number of methods available a for performing search and matching operations on text data, a task which is essential for any NLP process. A few such are covered here.

2.6.1 Exact String Matching Algorithms

The most straightforward string matching method, at least in concept, is the naïve or "brute force" implementation. This involves checking characters in a sub-string against every character in a corpus in sequence. This is rarely the best option for performance except when working with short strings there are a number.

There are, instead, a few methods which use smarter methods to improve performance. D. Knuth, Morris, and Pratt (1977) developed an algorithm for exact string matching which aims to reduce the number of required comparison by using the information learned during one comparison pass to skip later comparisons. The key to this method is in the computation of a 'failure function'. When using the Knuth-Morris-Prath (KMP) algorithm, the target sub-string is preprocessed to locate repeated suffixes. The purpose of this is that, if it is known where suffixes of a string are located within itself then for any failed match, it is only necessary to resume searching at the latest matched suffix, rather than returning to the start of the sub-string. Using such a method, it is possible to search for a sum-string in a volume of text whilst needing to examine characters in the text at most, a single time.

Similarly, the method described by Boyer and Moore (1977) improves matching performance by reducing the comparisons required, but the manner in which it does so is somewhat different. The target sub-string is again preprocessed, but in this case the preprocessing is to establish 'bad suffixes' and 'good suffixes'. Having this information available during matching allows skipping chunks of text which could not possibly contain the sub-string, in a manner not dissimilar to the KMP method.

In addition, the Boyer-Moore algorithm starts by checking the last character of the sub-string against the character in the corpus which is offset by the length of the sub-string. This allows skipping chunks of the corpus when this last character doesn't match, as it would be impossible for the rest of the sub-string to have appeared before this last character.

2.6.1.1 Fingerprinting

Karp and Rabin (1987) present a method of exact string matching that uses short strings, or 'fingerprints' to represent larger strings of text. This constitutes an efficiency over naïve string matching in that the number of required comparisons is reduced when compared to comparisons of the full-length strings. This method uses a 'rolling hash' to allow fast computation of the fingerprints. For any new query pattern 'm' of length 'm_c' a hash fingerprint is computed, and likewise for the text to be matched against 'n', a hash is generated for the first 'n_c' characters. The text is then iterated over one character at a time, and the rolling hash method allows very fast updates to the fingerprint value by only considering the new character at the end of the current window, and dropping the character at the start. The comparisons continue in this fashion until either the end of the text is reached, or a match is found. In the case that a match is found, the algorithm then falls back to character by character comparison in order to confirm that the match is not a false positive. This step is necessary as the hash function is not sufficient to generate a unique fingerprint for every possible sequence of characters, doing so would require significantly more processing time and would result in fingerprints no shorter than the strings themselves. However, because this method reduces to use of character by character comparison to circumstances where a potential match is located, rather than being necessary for the entire text, there is still potential for improved efficiency.

As with all methods requiring some additional processing aside from character comparison, the overheads involved with hash comparison methods mean that, for short strings, the efficiencies gained by avoiding large amounts of character comparison do not apply. The reason for this is that it is often possible to locate a sub-string in a relatively short text using naïve methods faster than it is possible to perform the initial preprocessing of hashes.

2.6.2 Search Methods

Given a system where tokens in text can be represented by numerical values, some consideration can be given to the methods used to locate those values.

2.6.2.1 Linear Search

Linear, or sequential searching can be described thus: "Begin at the beginning, and go on till you find the right key; then stop." (D. E. Knuth, 1997, Vol 3, Chapter 6.1). This is perhaps the most conceptually simple of the searching methods. Consider an array of data "n", and a target value "m"; First, the target value is compared to the first value in "n". If they do not match, then the next value is checked likewise. This continues until either a match is found, or the end of the data is reached. If at any point in the sequence the values are equal then the target value has been found and the process can either stop here, or in the case that multiple instances must be checked for, the process is repeated starting at the next value. Fig. 2.5 illustrates a number of search scenarios, and demonstrates how the linear search method would perform. When counting occurrences of a target value, then it is always necessary to check every index. Therefore, for an array of length 'n', the number of required comparisons is 'n'. If all that were required were to check for the existence of a value in an array then it is possible to stop looking after finding the required value, so the performance in that scenario is dependent on where the value appears. If the value does not appear, then again, the number of required comparisons is 'n'.

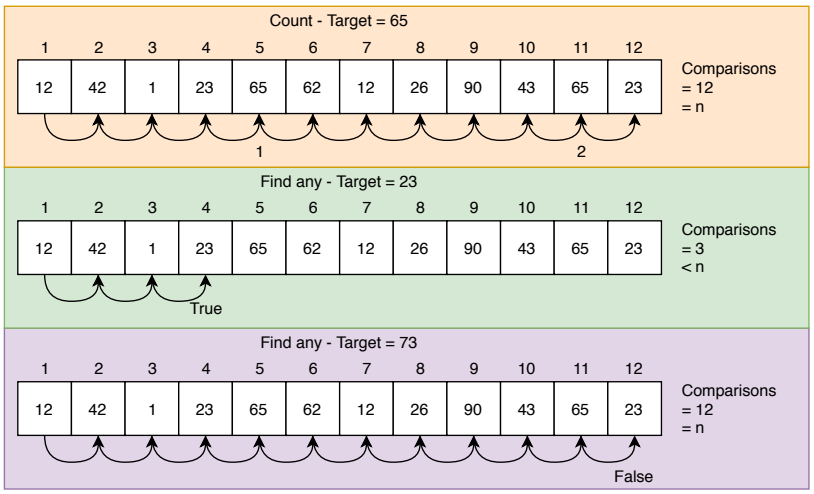

Figure 2.5: Three examples of sequential searching in use, First counting all instances of a target value, Second and Third, Checking for the existence of a value.

The naïve method of string matching functions in this fashion, with the slight modification that, instead of a single value, the target is a sequence of values, or characters. The initial target value is the first character of the target string. The process is carried out in much the same way as before until a match is located. In any instance where a match is found, the target value is then updated to the next value in the target string, which is then compared against the next value in the data array,

and so on. This continues until either: the comparison fails, in which case the target value reverts back to the first value of the target string, and matching resumes at the point after the initial character match; or the end of the target string is reached, indicating a successful match.

2.6.2.2 Binary Search

Binary search allows locating items in sorted data in significantly few operations than is possible with a linear search, particularly for very large data sets. Rather than start at the beginning, the middle index of the array is calculated and a comparison takes place. If there is not a match at this midpoint then, if the value is higher, then a new midpoint in the upper half is calculated. Likewise, if it is lower, a new mid point if calculated for the lower half. This process is repeated until either a match is found, or a midpoint can no longer be calculated, indicating that the sub-string is not found in the text (D. E. Knuth, 1997, Vol 3, Chapter 6.2.1). Incidentally, if no match is present, the position given by the binary search is the location where that value should be inserted should it be necessary Fig. 2.6 demonstrates the behaviour of a binary search when performing the same tasks introduced in Fig. 2.5. Now, rather than requiring up to 'n' comparisons to locate a value in an array or length 'n', now it is possible to complete all search tasks in, at most, log(n).

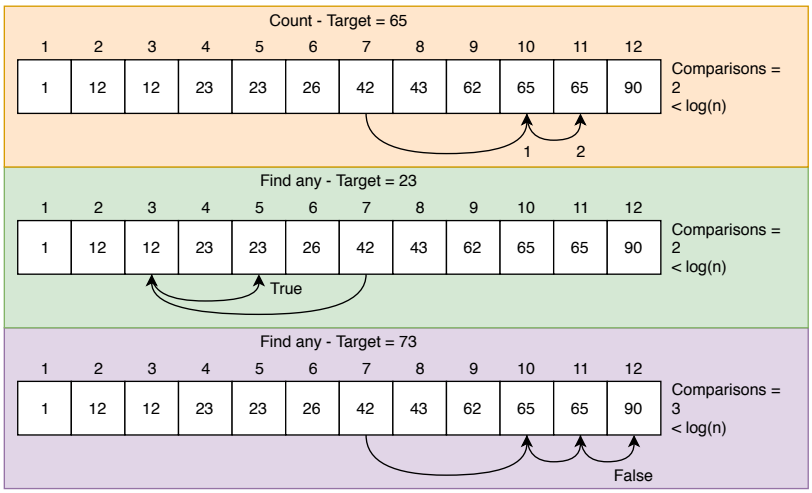

Figure 2.6: Three examples of binary searching in use, First counting all instances of a target value, Second and Third, Checking for the existence of a value

As the data must be sorted to allow the use of binary search algorithms, there is a penalty to insertion speed. Inserting elements requires first performing an binary search to locate the insertion location, then likely requiring moving all the elements after the insertion point to make room.

2.6.2.3 Binary Search Trees

Binary trees are a data storage structure where each item in the tree has up to two children. This structure can be traversed in the same fashion as a binary search, but without the need to calculate midpoints.

Binary search trees mitigate the insertion penalty inherent in sorted data as the index can be maintained separately to the data, i.e. upon insertion of a new element, only pointers need to be updated. This avoids costly memory reallocations. (D. E. Knuth, 1997, Vol 3)

2.6.2.4 Further Efficiencies from Binary Searches

Depending on the data structures in play, it can be possible to use binary search type techniques in a more efficient way. Two such methods for doing so are fractional cascading and exponential search. Both methods improve search efficiency; fractional cascading by reducing the required number of searches required when multiple related lists are used, and exponential search by reducing the range over which the binary search needs to be applied.

Fractional Cascading Fractional Cascading is a method for reducing the number of binary searches required to retrieve a key from multiple related sorted lists. Chazelle and Guibas (1986a) describe a 'fractional cascading structure', a data organisation technique which allows locating corresponding keys in multiple related lists while only requiring a single binary search for an initial match. Once the initial match has been made, the structure contains the necessary information to bridge to the appropriate position in another related list without the need for additional searches. The method is related to an earlier technique, hive-graphs.

A hive-graph(Chazelle, 1986) is a data structuring technique whereby fractional samples of related structures are stored alongside neighbouring ones so that one binary search can return related keys from multiple lists. This however requires a space trade off as a result of storing the fractional samples.

In fractional cascading a refinement is made to the data structure which mitigates the penalty to space complexity.

Fractional cascading has been applied to a number of range finding problems and geometric search problems (Chazelle & Guibas, 1986b), and while not directly applicable to matching text keys in a document (typically a key lookup of this kind is a single search in one list) it may have some merit for locating the same key in multiple documents (E.g. a reporting system full of incident reports) provided the contents could be ordered in such a way that did not preclude it.

Exponential Search Exponential search (Baeza-Yates, Salinger, Mannila, & Orponen, 2010) improves upon regular binary searches by first determining a range within a sorted list where the target resides (or, in the case it does not exist, where it would reside) before then performing a binary search within that range. Instead of finding the midpoint of an entire list and performing a binary search over the entire range, a smaller range is first found by jumping through a list k in increments of 2^j where j is an exponent which increases by 1 for each jump. When the condition $k[2^{(j-1)}] < TargetValue < k[2^j]$ is met, a binary search is performed on the resulting range. Not only does this reduce the range upon which the binary search operates, it also enables the use of binary search on unbounded lists, where it is not possible to determine a midpoint.

2.6.3 Summary

It is clear that unstructured free-text reports can be used to great effect for safety learning. It is also clear that the volume of these reports across a range of industries and domains is very large, and continuing to increase with each year. Most safety related information extraction methods covered in the literature focus on a relatively small sample of available data. Applying these methods at scale requires significant computing resources in order to get results in a timely manner. While efforts have been made to mitigate this through, for example, parallel processing, this issue will only become more pronounced as text databases continue to grow.

By studying various processes that are used for safety text learning across a variety of domains it can be seen that many approaches share a lot of the same basic techniques (as shown in table 2.2). Furthermore, a large proportion of these techniques at some stage must produce a full text index, or perform queries, both of words and multi word phrases (N-Grams) on raw or preprocessed texts. If the way that raw, unstructured texts are pre-processed, indexed, and queried were optomised for speed, the reduction in time complexity could have a positive knock on effect on the multitude of NLP, classification, machine learning, and other processes commonly applied to this problem.

In order to develop and test an efficient text indexing and querying system for safety texts, text from the Close Call system will be used. This text is available in very large quantities (around a million records), and serves as a good example of unstructured text in a domain which has been demonstrated to have value to safety learning. To this end, a system which indexes words and multi word n-grams using integer fingerprints, or 'hashes' has been designed, and binary search techniques have been employed to allow unprecedented matching performance.

Chapter 3

Methodology

Through study of the numerous analysis techniques used in the processing of text reports a number of requirements can be established. Many tasks, such as document vectors, feature extraction and classification require accurate measurement of the frequency of tokens within text documents. Others, such as tagging and certain semantic or grammatical processes require specific locations of tokens within a text. So, while improving the processing and response time of text operations, it is important to also maintain the locations and frequencies of tokens, as they appear in the original text form if the faster method is to be useful in enabling those same techniques to be used.

Referring again to the optimisation triangle (Fig 3.1), it can be identified that the most important factors in optimisation of text operations for safety text sources are speed, and completeness.

In this chapter, a method for fast string indexing and querying which attempts to maximise these two qualities is described.

3.1 Variables for Optimisation

Testing has been designed so that, when measuring time, the only variable is the matching method. Each method receives the same terms for comparison, and these are always compared against a corpus derived from the same text data.

As indicated by the preliminary tests, match times are not necessarily consistent for every search string, so it is not sufficient to compare the speed of a single search. To measure matching speed, it is instead necessary to perform a number of measurements with differing input strings in order to create a complete picture of the performance of the method.

In order to assess the success of this 'hashed n-gram' method, a handful of experiments were devised to compare the speed of this method against not only the

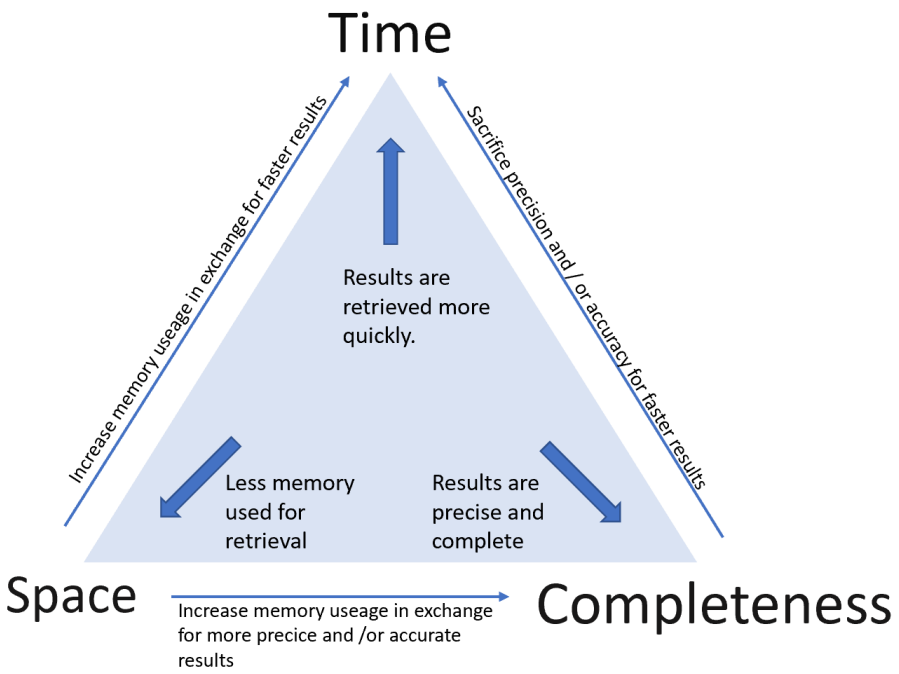

Figure 3.1: Optimisation of matching procedures is a balancing act between, time, space, and completeness

naïve brute force method; but also the Boyer-Moore method, which is often used as a benchmark in literature pertaining to string matching; and the string::find function from the C++ standard library.

3.1.1 Constraints

It is also of great importance to ensure that the new method performs the same function as the existing ones. To this end, the following constraints are set;

- For each measurement, every method must be given the same input. E.g, When comparing matching performance of different string matching methods, the same substrings must be located.
- Each method must operate on identical source data. All experiments will be performed on identical close call reports.
- Each method must return identical results.
- To account for variance due to factors outside of the software such as operating system scheduling and background tasks, Each time measurement procedure was repeated 5 times, and an average was taken of each set of results.

To meet these criteria, matching functions were written which differ only by the matching method. The tests themselves are written around these functions.

When discussing text pattern matching it is useful to refer to a 'Needle in a Haystack' metaphor. In this case the 'Haystack' refers to the text in which we wish to locate one or many 'Needles', which, likewise, refers to the substring which is to be located.

3.1.1.1 Haystack

For all tests, the source text consists of 500000 close call reports, approximately 123MB of text.

For each token in the close call data, the hash of this text is stored, alongside its line number, position, and length.

3.1.1.2 Needles

Needles, or the sub-strings which are to be matched, are treated in exactly the same way as the haystack text, to ensure that comparison of 'like-for-like' takes place.

3.1.2 Measuring Performance

To measure and compare the raw performance of the differing matching methods, a number of tests are to be carried out. Russell and Norvig (2002) propose a number of criteria for comparing the performance of 'Problem-Solving' algorithms:

- Completeness: Does the algorithm produce the expected results (if present)?
- Optimality: Has the optimal solution been found?
- Time complexity: How much time does the algorithm take to complete?
- Space complexity: How much memory is required to complete the search?

These criteria are in reference to searching the solution space for problem solving tasks, and as such one criterion, optimality, does not directly apply to pattern matching performance as no medel fitting is involved. However, the remaining 3 criteria: completeness, time complexity, and space complexity; can all be used to draw comparisons between the methods. The following tests have been designed to assess these criteria.

3.1.2.1 Completeness - Collision Prominence

When searching a corpus for any text matching a given query, the results provided by any searching algorithm can be considered 'complete' when all instances of the given query are returned, and where location is required, the location is precise.

The method used here to generate integers for each word or n-gram (namely, the std::hash function from the C++ standard library) is not a 'Perfect Hash' function, and any sufficiently large dictionary will result in the existence of identical hashes for non-identical sub-strings. In order to evaluate the impact this has on matching accuracy and completeness of results in the example close call text, a full accounting of all terms and corresponding hashes was carried out.

The 64bit data types used to represent the text tokens extracted from close call, can resolve to $2^{63} - 1$ integers. For comparison, recent estimates of the number of words used in the English language stand around 250000, including many words which have fallen out of use (although in specific domains (for example, railway safety), this can be much higher due to domain specific jargon). In practice however, given a hash function that distributes hashes in a uniformly random fashion, collisions will be encountered much sooner than the limit of the 64bit integer. The reason for this can be explained by way of the birthday paradox, or, the phenomenon whereby, given a group of individuals with birthdays uniformly distributed throughout the year, the likelihood of two individuals in a group sharing a birthday is higher than might be expected (Abramson & Moser, 1970). To reach a 50% probability that at least one pair of those individuals share the same birthday, the group need only contain 23 individuals. This, despite the fact that there are 366 possible birthdays. As any text matching solution can potentially be required to account for millions of distinct tokens, one pair of matching hashes out of a million is unacceptable. In reality, as hashes are determined mathematically by the content of the data they refer to, hashing functions, like birthdays, cannot achieve a uniformly random distribution of hash values. The issue of collisions, therefore, becomes even more pressing.

One mitigating factor is that n-grams are indexed separately depending on word count. As n-grams are processed and searched separately based on the size of 'n', it is only necessary to check for collisions against tokens of the same word count. Take for example a term with two words, or bi-gram; and a term with three words, or tri-gram; both terms could safely be allotted the same hash without risking a collision during matching, as any query with two words would only be matched against hashes derived from bi-grams, and likewise, any query with three words would only be matched against hashes derived from tri-grams.

The probability that a collision will occur can be approximated using the following method, as described by Preshing (2011). Assuming a uniform distribution, the

approximate probability that at least two items will compute into the same hash is given by:

$$1-e^{\frac{-k(k-1)}{2N}}$$

Where k is the number of items requiring hashes, and N is the total possible number of unique hashes, in this case, $2^{63}-1$. For reference, there are 744667 unique uni-grams in the 500000 records used for collection of results

It is important to note that in the instance that hash collision is detected, it would be necessary to implement additional checks on resulting matchings to account for false positives (False negatives are not possible, as all instances of a unique word always receive the same hash). This method is employed by most implementations of the Rabin-Karp finger printing method, whereby upon locating a matching hash, the match is then rechecked using a naïve matching method. This is required as the Rabin-Karp method derives much of its performance from an efficient rolling hash algorithm, which, while fast, does not guarantee a uniform distribution of hash values. As this additional check is only carried out on the subset of the haystack that matches the needle, this added complexity is considered acceptable, and indeed, in the case of Rabin-Karp, necessary to maintain exact matching.

The outcome of this test is an accounting for every token in the text, including n-grams, and a count of the number of collisions.

3.1.2.2 Time Complexity

To obtain a measure of the per-match performance of each method, matching queries were made on the close call data using a range of string matching methods (Naïve, Boyer-Moore, string::find (from the C++ standard library), and the 'hashed n-gram' method).The elapsed wall time was recorded while each method was tasked with counting the first 100 unique tokens contained in the close call data, for each size of n-gram up to n=9 (a total of 900 tokens).

To show how the method performs as the size of the data increases, the match performance test was repeated with different numbers of reports. Starting with 1 report, and doubling up to 131072 reports. The same 9000 queries are made as before, for each set of reports.

In addition, time measurements were taken for the time taken to count 100 unique tokens in the same 500000 reports used previously. However for this test, separate measurements were taken for each size of n. In other words: a measurement for the time taken to count 100 unique uni-grams, a measurement for the time taken to count 100 bi-grams, tri-grams, and so on.

3.1.2.3 Space Complexity

The space each method requires to return matches from a given corpus is to be compared by measuring memory useage as corpus size is increased. This serves two purposes. Firstly, a measure of how the memory usage differs depending on the different transformations of the corpus. Secondly, an indication of how memory useage grows to accommodate larger corpora.

3.2 Integer Matching

As a starting point, following the work described by Karp and Rabin (Karp & Rabin, 1987) on 'fingerprinting', integer based matching was investigated as an alternative to brute force matching. Some testing was necessary to determine whether integer pattern matching could offer any performance advantage when matching strings in text reports.

For this initial test, arbitrary strings of 9 character 'words' were generated, and a method for measuring the elapsed wall time for the operation was constructed. A single comparison of two 9 character strings occurs so quickly that the actual time is below the precision of any tested method of measurement. To circumvent this, the operation was performed $2^{31} - 1$ times (The limit of a 32 bit integer in Java), then the recorded time for all these operations divided by $2^{31} - 1$ to produce an average. even performing the comparison this many times, the elapsed time was still small enough that transient changes in CPU load generated by processes other than that being measured would potentially alter the result. To further compensate for any anomalous measurements, this entire process was repeated 10 times for each test, each time a few minutes apart, and an average taken.

The nature of brute force pattern matching dictates that different matches will not always be completed in the same amount of time, As each character must be compared until a non matching character is found. A sub-string which differs from the search pattern at the first character will be rejected in fewer operations than one which does not differ until the final character. In the case of a match, every character must be checked. In order that these situations are included, and to avoid drawing any false conclusions from timings taken from these matches, measurements were also made for strings that were equal up to the last character (For example: Tare and Tarp), as well as measurements for strings where only the first character differs (For example: Hollow and Follow).

The same comparison test was carried out with integers instead of strings. Using the same conditions, 9 digits, once when equal, once when differing by the last digit,

and once when differing by the first. For string comparison the Java string equals() method was used, for integer comparison the equality operator(==) was used.

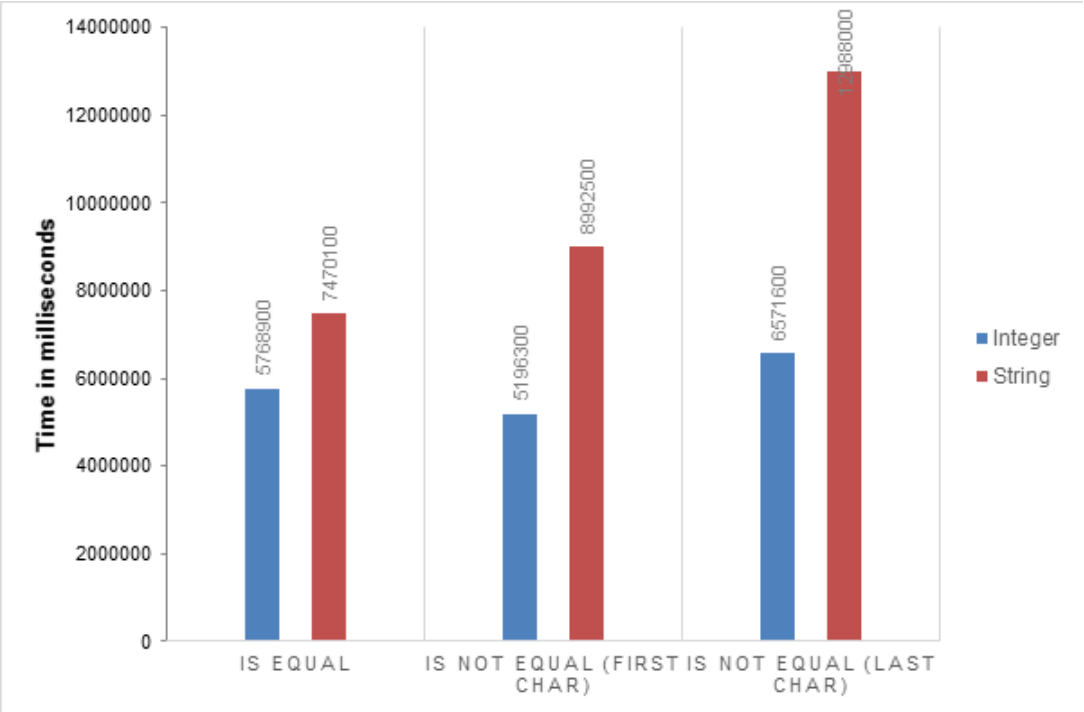

Figure 3.2: Averaged timings for String and integer comparisons

The results of these tests are shown in Figure 3.2. The results appear to show that using this measurement method, integer matching is faster than string matching, as suggested by the literature. However as can be observed in Table 3.1, the timing data before being averaged fluctuates for what should be identical tests. A more thorough benchmarking method was required. Boyer (2008) explains that running time in Java is unpredictable due mainly to a period of 'warm up' where the JVM loads classes as they are required, and the just in time (JIT) compiler optimises performance by compiling patterns which prove common at runtime. To accurately and consistently measure running times, a benchmark that tracks these events and accounts for them in results is required. One such suitable benchmark is described by Boyer, and is used for all further measurements. The benchmark executes the program multiple times until a stable state is identified, before calculating an average running time.

3.2.1 Applying These Methods to Real Data

Since it had been demonstrated that integer matching was likely more efficient, data from the Close Call system was obtained in order to perform a case study.

Table 3.1: A subset of results from batches of comparison tests

	Equal	Not Equal (First Char/Digit)	Not Equal (Last Char/Digit)
String	7136000	10754000	12148000
	7285000	8519000	14306000
	8581000	8388000	12558000
Hash	5327000	6667000	6378000
	6022000	4467000	6180000
	5833000	4450000	7178000

In order that each unique word in the Close Call text has a unique integer value, the Java Hashcode function was used. At this point, collisions were not considered, due to the size of the sample used. As hashing is a 'lossy' process, the original string cannot be retrieved directly. However, by maintaining the structure (i.e. line number and position in line) of the original text, it is possible to do a lookup on the original text once a match is found using the indices of the match in the hashed data.

Consider the following paragraph:

Mary had a little lamb
Little Bo Peep
Bah bah black sheep

Now consider that this text is stored in a table, so each word can be addressed by a pair of co-ordinates:

Table 3.2: A Text Example

	0	1	2	3	4	5
0	Mary	Had	A	Little	Lamb	
1	Little	Bo	Peep	Lost	Her	Sheep
2	Bah	Bah	Black	Sheep		

Now that the dimensions of the paragraph are known, that is, line lengths and line numbers, hashes can be generated and stored in a table with exactly matching dimensions:

Table 3.3: An illustration of the same text after generating hashes

	0	1	2	3	4	5
0	232	784	621	960	720	
1	960	452	915	582	902	572
2	249	249	401	572		

Once these hash values are generated, they can be searched quickly by generating a hash value for the search term (there may be many at once, but for this example and for simplicities sake only one is shown). Say that all occurrences of the word "sheep" were to be located. A hash value is generated and the table is iterated through until a match is found, at which point its location is saved. In this example this yields the result 1,5 and 2,3. As the dimensions of both types of data match, we do not have to use a lookup table to determine the meaning of any hash, as once a hash is located, the original text (if required) can be derived by simply reading the same co-ordinates in the text data. The alternative would be to maintain a list of all unique words alongside the corresponding hash value, which would need to be searched through for every single hash whenever a conversion back to text was required. There are text analysis tasks, such as frequency analysis, for which conversion back to text would not be required.

To test the hashing method on text-based reports, three case studies were devised to represent operations that would be typical for this type of data source, namely pattern search, find and replace (tagging), and generating n-grams, n can be any number but in this case n is equal to 3, so the software generates trigrams. The Java hash-code function is used to calculate integers for each unique word, an example of the result of this process can be seen in Table 3.4, note that the two occurrences of the word "the" share the same hash in the output. not shown in this table is that the line number and position in each line has been preserved in the hashed data.

For each study, three versions were tested, each performing the same task, but differing in implementation. The first performs the operation using the string comparison method included in the Java standard library. The second generates hashes from the text data, keeps it in memory and uses this for indexing. The third loads a pre-generated file with the hash index. This final case is representative of a situation where multiple queries may be made of the data. In such a situation the hashes need only be generated once, and can be queried multiple times without the initial overhead of conversion. Pre-generating a file for this purpose from a set of 150000 close call records takes an average of 2.97 seconds. Figure. 3.3 illustrates these different approaches for the location listing case.

3.3 Using n-grams to Reduce Search Operations

N-grams are a sequence of n items taken from a given (in this case) sequence of text, which retain the order as found in the source. An n of 1 splits the sequence into unigrams - containing one item, n of 2 produces bigrams - containing two items, and so on. the n-grams do not typically follow on from end to end, but rather as

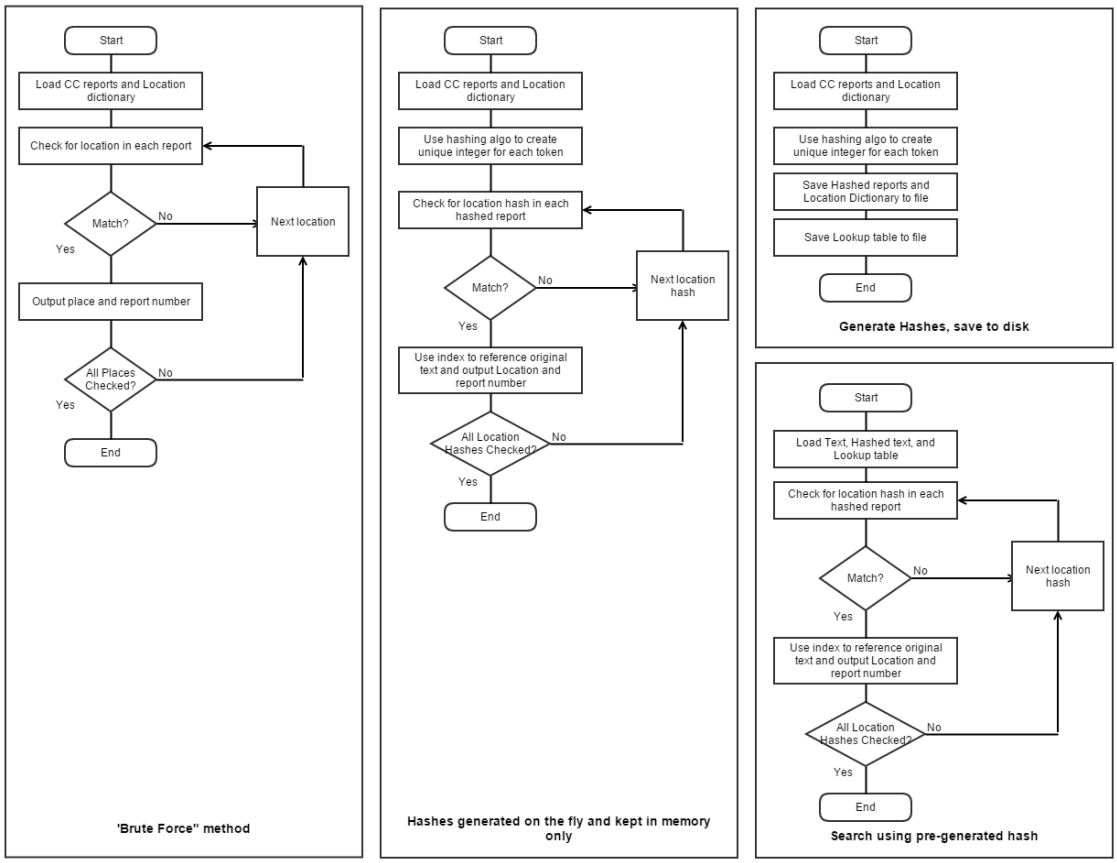

Figure 3.3: Program Flow for each 'place name search' method

a window of size n items, which moves item by item. n-grams were included as a testing example for the hashing method as they are one of the processes currently in use by others working on the close call data. However, the properties of n-grams may provide a method of improving pattern matching efficiency further (Newall & Van Gulijk, 2019).

The worst case for a brute force matching technique, which occurs when no matches are found, requires checking every character of the search string with every character of the text being searched. The number of required operations to complete this search is therefore the total number of characters in the source text multiplied by the total number of characters in the search string. In other words, the maximum required operations is $C_t * C_s$, where C_t is the number of characters in the the source text, and C_s is the number of characters in the source text. In practice a match may be found earlier than this in some cases.

By generating hashes for each unique word, and also each word in the search string, the required operations is reduced somewhat. As whole words are reduced to a single integer, this worst case is reduced to the total number of *words* in the source

Table 3.4: Output of the hashing operation, emphasis shows that both occurrences of the word "the" have matching hashes

Original Text	22/03/2014 12:27,***the*** driver of 1J19 1407 Birmingham International, to Aberystwyth reported that ***the*** crossing gates at Abermule had been left open.
Hashed Text	49089694 46799422 ***114801*** -1323526104 3543 1563201 1511274 -706256204 2064805518 3707 2075440762 -427039533 3558823 ***114801*** 2123306914 98127112 3123 1330114223 103057 3019820 3317767 3417674

text multiplied *words* in the search string. This does not account for the time needed to generate the hashes, but this can be done very quickly, relative to the time taken to find matches. Thus, the maximum required operations required for hashed text is reduced to $W_t * W_s$, where W_t is the number of words in the source text, and W_s is the number of words in the search string.

The implication of this is that, using a naïve matching approach, the number of operations will increase as the number of words in the search string increases. N-grams can be generated from the source text where n is equal to the length of the search string, i.e. $n = w_s$. These n-grams can then in turn be hashed so that each unique n-gram is represented by an integer. The following formula gives the total number of n-grams, produced from a text, $_t$ containing W_t words:

$$n - grams_t = W_t - (n - 1)$$

For example a search string, W_s containing 5 words, compared against a source text of 1000 words from which n-grams of size 5 are produced and hashed (W_S), would require at maximum $W_t - (n - 1)$, or 996 operations. Similarly, a search string of 10 words, compared again against a W_t of 1000, would require 991 operations at maximum.

Compare this to the previous, hash-per-word method; the same two comparisons on prepared text would require up to 5000 and 10000 operations respectively.

A pattern emerges whereby the length of the search term no longer has a multiplicative effect on the required operations for a comparison. Rather, the number of operations for comparisons against the prepared text is in fact reduced by n-1. The text need only be prepared for each size of n once, time taken to generate and hash n-grams of the text is amortised, and becomes less significant as more searches are made. Time complexity is reduced at the cost of of increased space complexity.

In order to investigate the effect this has on search times, a further test has been devised which combines both of the techniques discussed here; Hashing and n-gram

generation. First, the length of the search in words is measured and this becomes 'n'. If none has been prepared preciously, n-grams of size W_s are created for the text, and then hashes created for these n-grams. A single hash is then also created for the search term. The search string hash and the hashed n-grams from the text are then compared for a match. This process is illustrated in the flowchart in figure 3.4

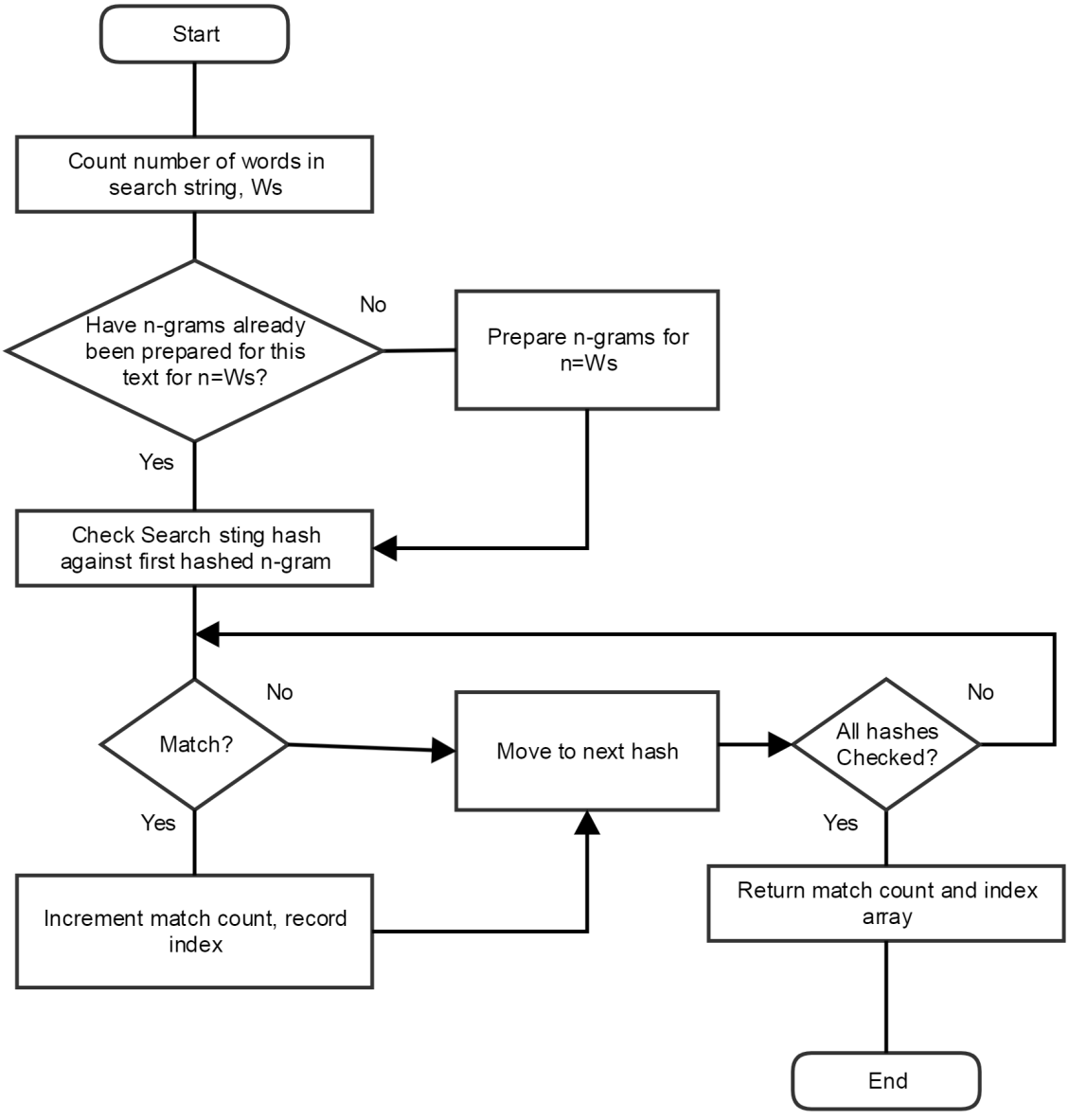

Figure 3.4: Program flow for the n-gram experiment

3.4 Bringing Everything Together

Following the aforementioned experiments, the methods used for each were combined in to a new piece of software. This was written in C++. This language was chosen firstly to remove some of the variability in run time brought about by the Java JIT compiler, and secondly, to reduce runtime, allowing more rapid turnaround from experiments.

3.4.1 Preprocessing

A preprocessing step that formats the text and generates the hashed data is necessary before any matching can take place. Text is loaded, either from a file or other stream. It is then formatted, line by line, to bring all the text into lower case (This is optional, and means that matches are not case sensitive), and to strip away any delimiting characters, e.g. punctuation, and replace them with spaces, so that only full words remain. Each line of the formatted text is then kept in memory. These formatting and tokenisation steps are illustrated with an example in fig 3.5

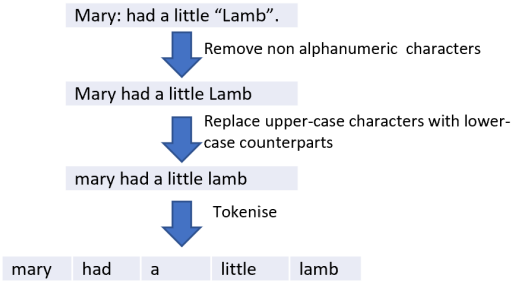

Figure 3.5: A illustration of the preprocessing steps carried out on the source text

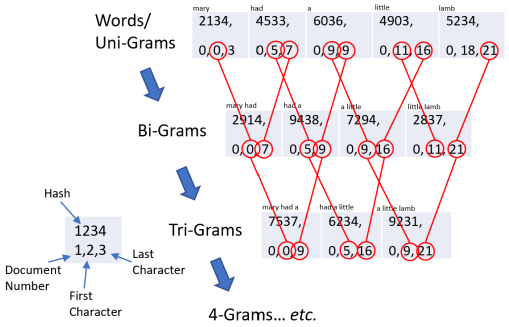

Figure 3.6: Formation of N-grams Using Existing Indices.

This formatted text is then iterated over, word by word, and an integer value is generated for each using the std::hash method from the C++ standard library. This hash method generates an integer mathematically from the content of a given string. Along side the hash for each word, the line number, and position in that line that the word appears are stored to form a tuple of 4 integers in the following form:

```
<hash, line number, start index, end index>
```

By storing text locations alongside the hash, we can build hashes of n-grams more simply than would otherwise be the case. Usually to build an n-gram where n is the number of words, it would be necessary to iterate over each character, to determine

when enough words have been read. This involves checking each character against a list of delimiters to find word boundaries. Instead, once these tuples have been created for the individual words as a result of the tokenisation step, it is possible to build arbitrary length n-grams using the existing indices without the need to iterate over the character data again, In other words, by reading the start index of the first word, and the end index of the word 'n-1' words ahead, the resulting character range can be used to directly reference the text corresponding to that N-Gram, and build a string for hashing. An Illustration of this process can be seen in Fig. 3.6

The size of n-grams generated will differ depending on the requirements. For the Close Call data used in these experiments most queries would not be over 5 words long. The n-gram length can be defined by the user when text is opened, to constrain the amount of required preprocessing. Alternatively, if a search is made which is longer than the maximum size that has currently been pre-generated, those n-grams can then be generated for the search. This pre-processing step is illustrated in Fig. 3.9.

3.4.2 Matching

An additional benefit of storing an index alongside each hash is that the order in which these hashes are stored in memory need not be constant. By sorting these tuples into ascending order, using the hash as the key, it is no longer necessary to iterate over the list of tuples in a linear fashion. Instead, a binary search pattern can be used. This operates as follows;

- A hash is generated for the substring to be located.
- this hash is compared to the item residing at the midpoint of the hashes processed from the text to be searched.
- If the hash for the search term is greater than the hash at the current midpoint, repeat the process for the upper half of the preprocessed data. Likewise, if it is lower, repeat for the lower half.
- Continue comparisons in this manner until either a match is found, or no further subdivision can take place.

'Midpoints' are calculated by adding the current upper and lower bounds together, dividing by two, then rounding down to the next whole integer. In each iteration of the binary search, upper and lower bounds are moved in such a way that, should there be no match, they will eventually pass each other - providing an exit condition.

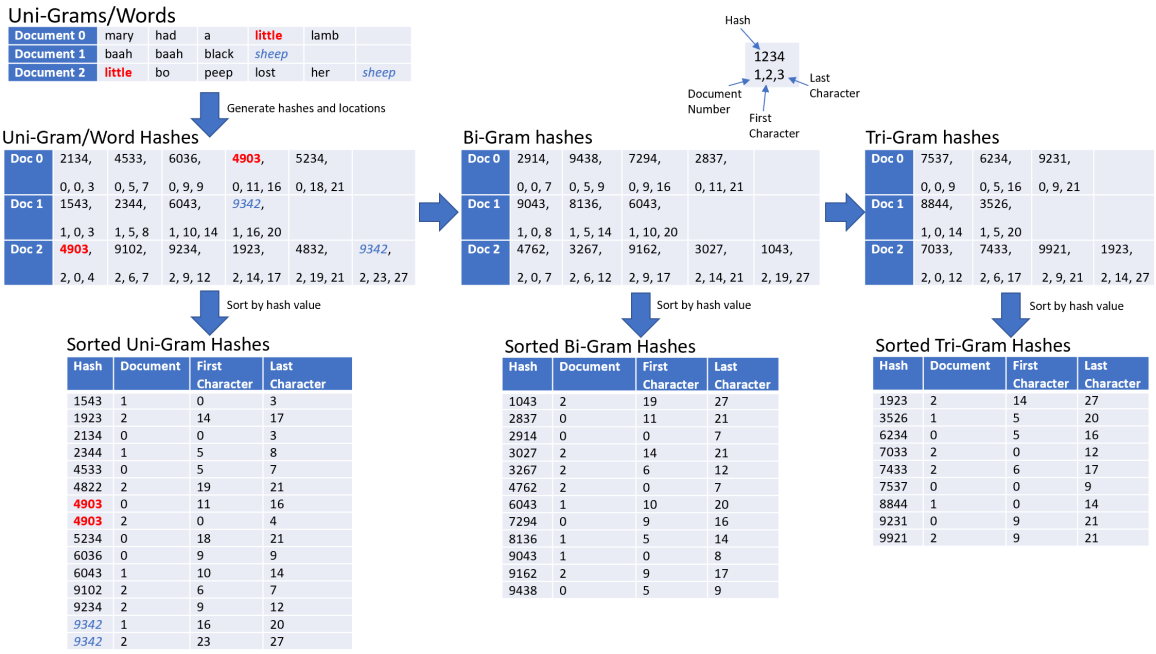

Figure 3.7: Generating inverted indexes for hashed N-Grams

This process takes at most $\text{Log}^2(\text{n})$ repeats, where n is the number items which must be compared to. In other words, in the worst case, doubling the size of the text to be searched, increases the maximum required number of comparisons by only 1. For comparison, when using a naïve matching method, in the worst case, doubling the size of the text also doubles the number of required comparisons.

Finding matches in text which has been processed in this way is, in the worst case, carried out in $\mathcal{O}(m + \log n)$. This compares favourably to the naïve method, which in the worst case performs in $\mathcal{O}(mn)$.

Fig 3.7 shows the path from tokenised text documents to indexed n-gram hashes.

An example of the matching process is illustrated in fig 3.8. A single hash is generated regardless of the number of words in a query, and the appropriate n-gram table is searched depending on the query word count. In this table, a binary search is carried out to locate the the query and return the count and/or its location depending on the users requirements.

To keep parity in the results of each method, matches are rejected if they occur within another word, As the requirements of pattern matching for Close Call are such that a search for e.g. “rain”, should not match within the word “train”. This rejection is not required for the hashed n-gram method, as only whole words (or whole word sequences in the case of n-grams) are captured.

The structure of this software is shown in fig 3.9.

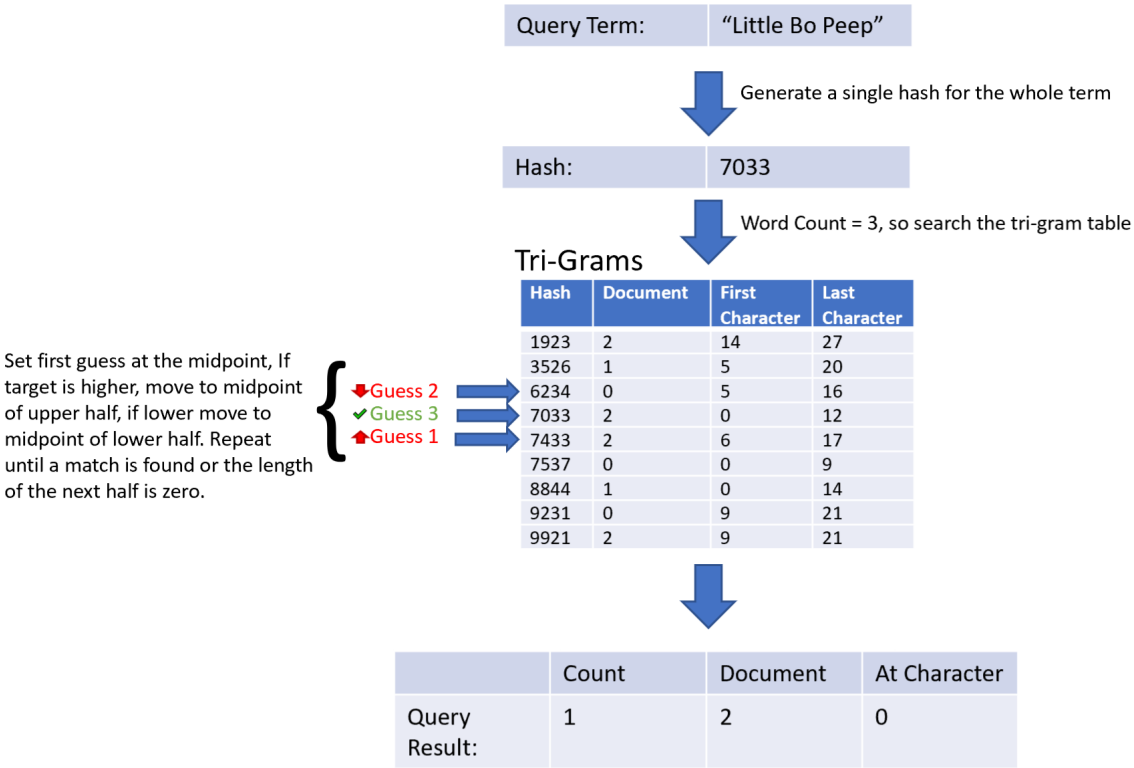

Figure 3.8: An example of the matching process for a query

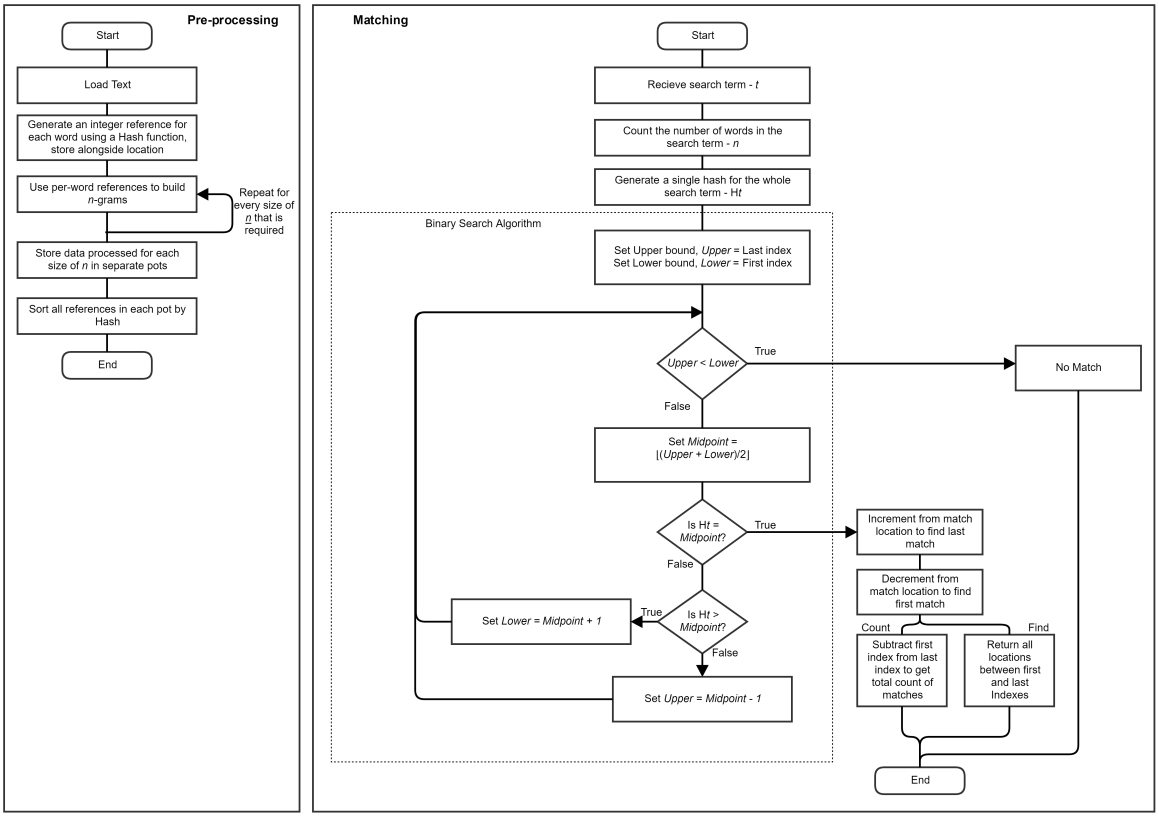

Figure 3.9: Program flow for final hashed n-gram software

Chapter 4

Results

4.1 Completeness - Collisions

Table 4.1: The total number of tokens for each size of 'n' after processing 500000 close call records, alongside approximate probability of a collision in sets of this size

n	Total Unique Tokens	Approximate Probability of collision	Total actual collisions
1	744667	3.0061×10^{-08}	0
2	3389335	6.22743×10^{-07}	0
3	8475642	3.89426×10^{-06}	0
4	12343491	8.25951×10^{-06}	0
5	14184880	1.09076×10^{-05}	0
6	14935413	1.20924×10^{-05}	0
7	15170115	1.24754×10^{-05}	0
8	15129520	1.24087×10^{-05}	0
9	14617801	1.15836×10^{-05}	0

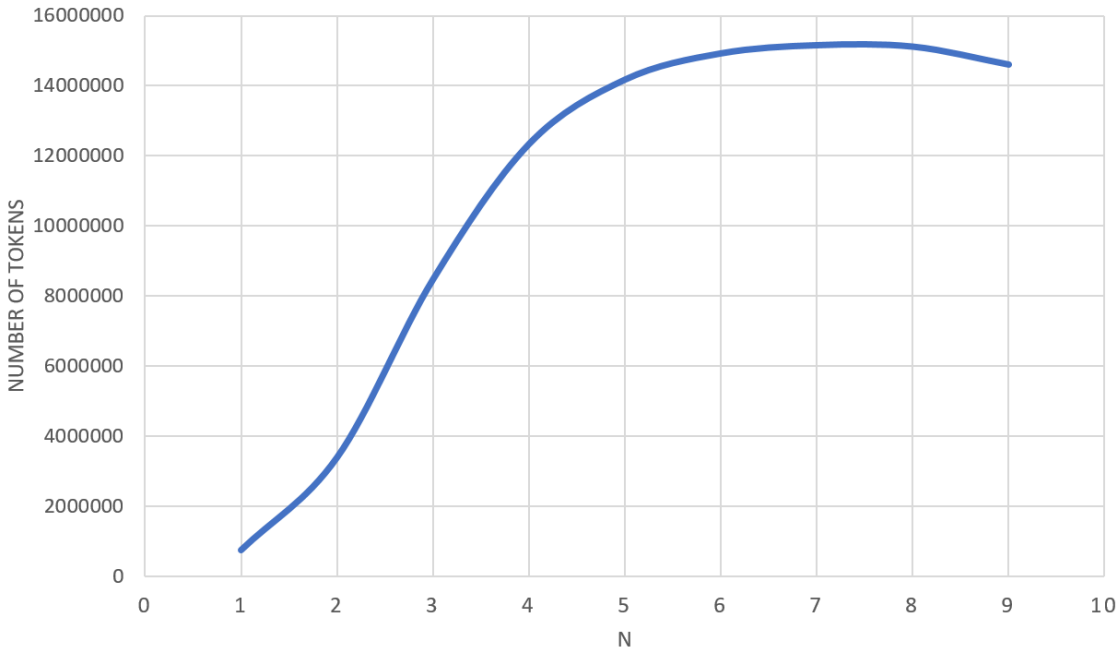

Figure 4.1: The total number of unique tokens for each 'n' after processing 500000 close call records

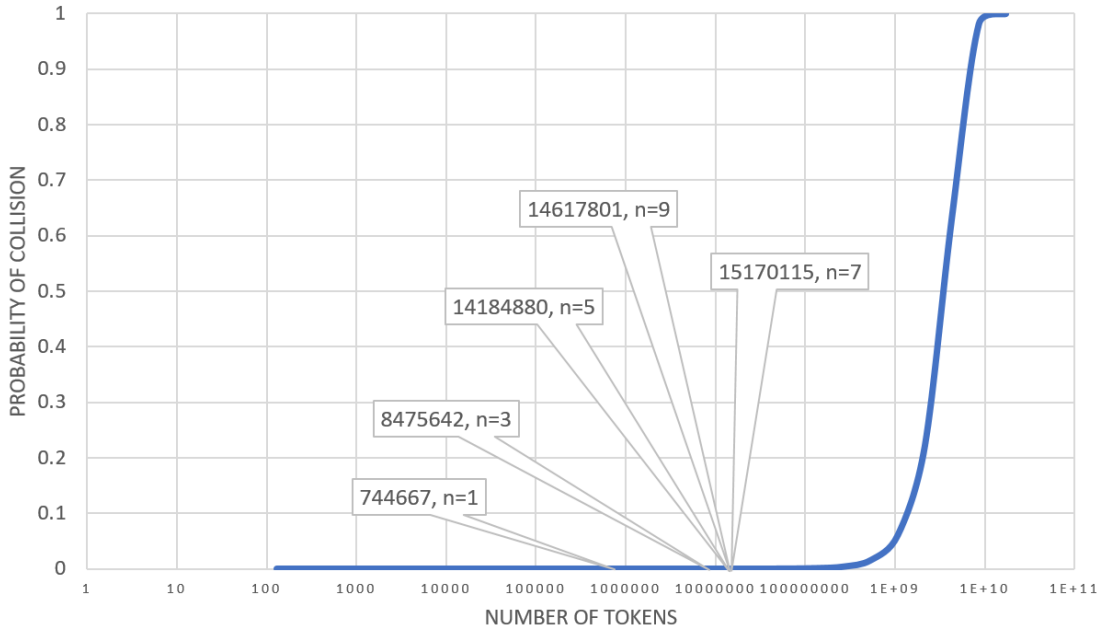

Figure 4.2: The approximate probability of a collision in the hashed tokens. Call-outs show where on the curve a selection of the token count values from Table 4.1 lie. Horizontal axis is logarithmically scaled

4.2 Timings

Figure 4.3: Time taken to count occurrences of 900 unique terms in 500000 close call records

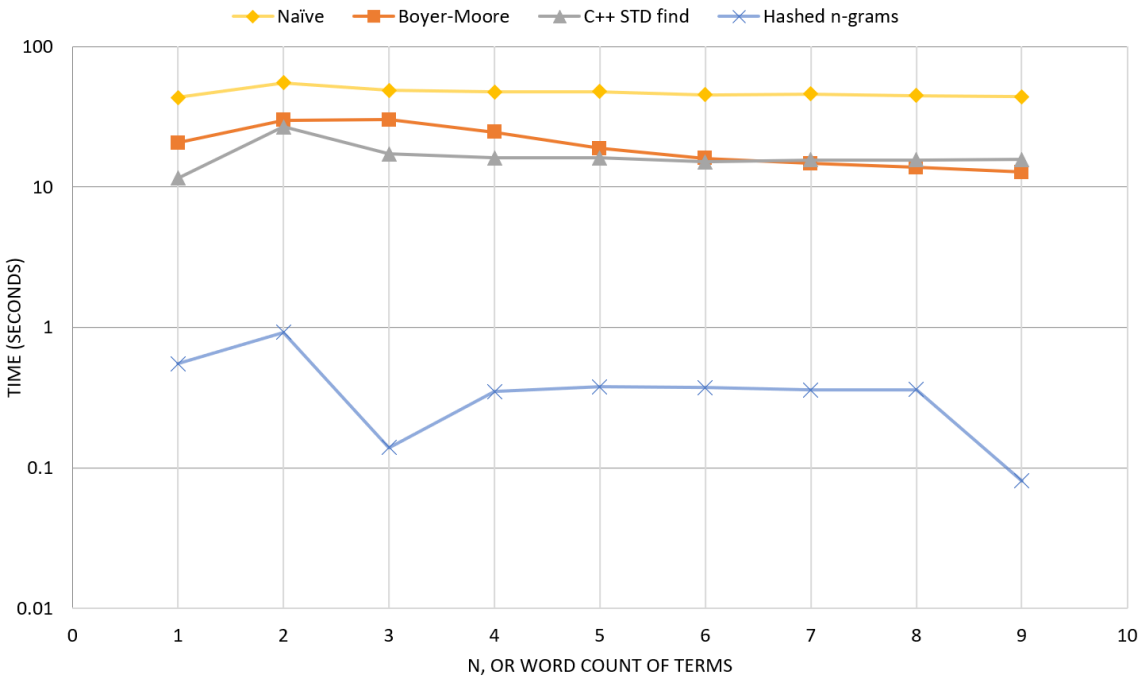

Figure 4.4: Time taken to count occurrences of 100 unique n-grams as n increases. Vertical axis is logarithmically scaled

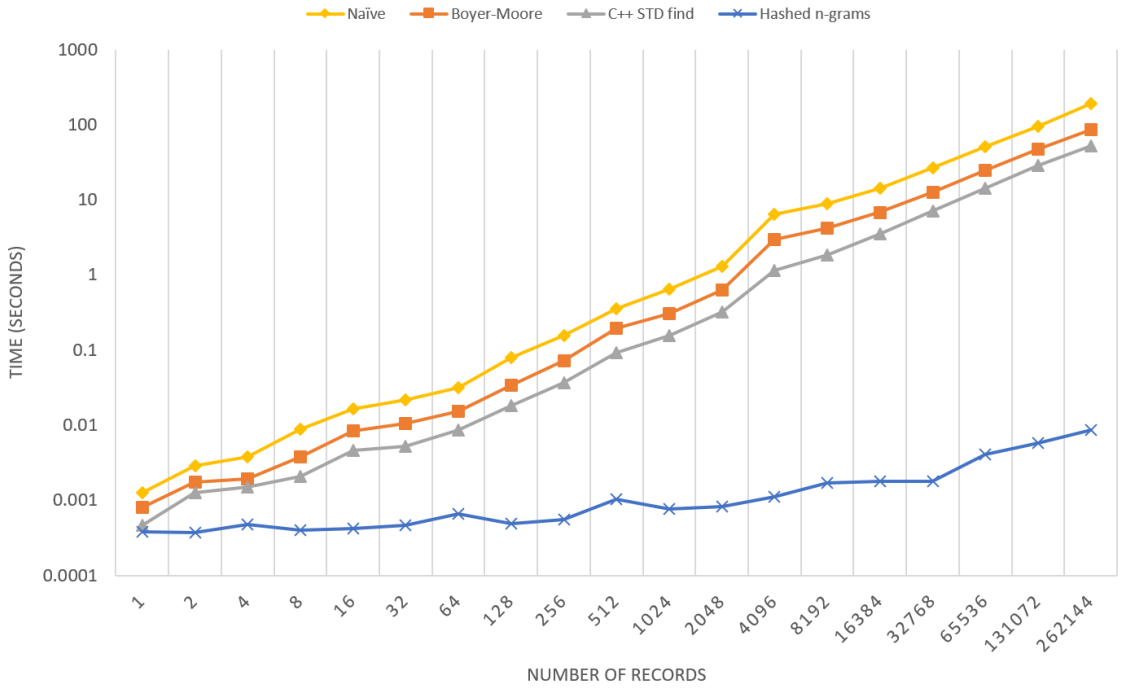

Figure 4.5: Time taken to count occurrences of 100 unique words in corpora of increasing size. Both axes are logarithmically scaled

4.3 Space Utilisation

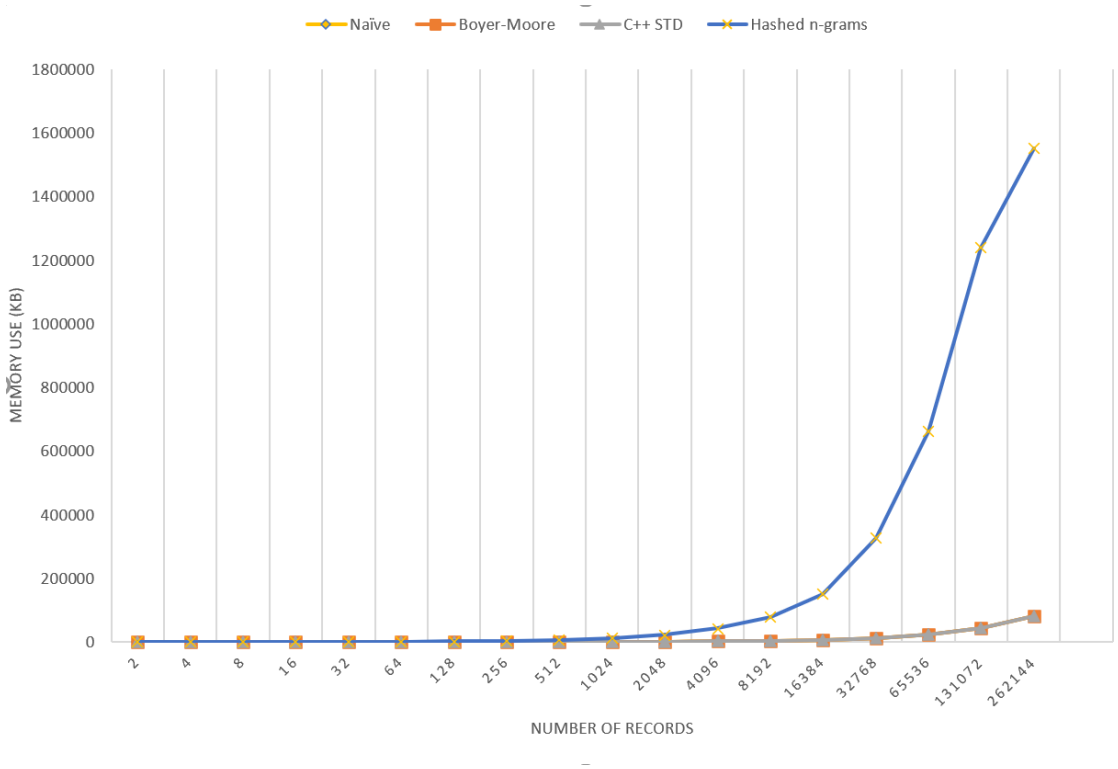

Figure 4.6: Space used by each method as the corpus increases in size. Horizontal axis is logarithmically scaled

Chapter 5

Discussion

These results clearly demonstrate that the hashed n-gram method is significantly faster at locating strings of text in a large corpus. Text analysis techniques are used in multiple fields to support the use of massive amounts of text data to inform safety and risk management. Many of these techniques operate on top of string matching processes to make use of the text. In order to evaluate how well these findings satisfy the stated aims of this work, these results must be considered in the context of the research questions that were posed.

5.1 How can safety N-gram queries be optimised to deal with large intakes of safety incident reports?

The simplistic answer to this question is: by responding to queries more quickly. However, evaluation of the various ways in which text is used in applications related to safety and risk, shows that this is not the only factor. To enable text analysis, completeness must be maintained for any text operation. In practice this means that locations of terms must be as precise as the other methods, and frequency information, In other words, aside from being fast enough to keep pace with the increasing volumes of text, the answer to: 'how may tokens are there and where are they' must remain consistent with expected values. The measurements shown in chapter 4 illustrate the effectiveness of the hashed N-gram method in achieving this.

In order to put these results into context, it is helpful to refer back to the summary of text processes used in various studies in chapter 2, reproduced here in table 5.1.

A task performed in all cases is tokenisation. In the process of building indexes for this method, tokens for words and arbitrary length word sequences or N-grams are produced, so tokens of this dimension are available as an inherent nature of the

Table 5.1: Summary of text processing and NLP methods used by a selection of studies using safety text

Publication	Author(s)	Methods
Early recognition of multiple sclerosis using natural language processing of the electronic health record	Chase et al. (2017)	Tokenisation, Statistical analysis of words and word N-grams, term search, supervised learning
A review of approaches to identifying patient phenotype cohorts using electronic health records	Shivade et al. (2014)	Tokenisation, Statistical Analysis of words and word N-Grams, supervised learning
Detecting inpatient falls by using natural language processing of electronic medical records	Toyabe (2012)	Tokenisation, Tagging, Term Search, Supervised Learning
Pneumonia identification using statistical feature selection	Bejan et al. (2012)	Tokenisation, Statistical Analysis of words and word N-Grams, Term Search
Natural Language Processing for aviation safety reports: from classification to interactive analysis	Tanguy et al. (2016)	Tokenisation, Statistical Analysis of words, N-Grams, stems, and stem N-Grams Stemming, Normalising, Supervised Learning, Interactive Learning, Topic Modelling
Application of Statistical Content Analysis Text Mining to Airline Safety Reports	Peladeau and Stovall (2005)	Tokenisation, Statistical Analysis of words and word N-Grams, Normalising, Tagging
Text analysis in incident duration prediction	Pereira et al. (2013)	Tokenisation, Statistical Analysis of words and word N-Grams, Tagging, Topic Modelling, Supervised Learning
Real-Time Detection of Traffic from Twitter Stream Analysis	D'Andrea et al. (2015)	Tokenisation, Stemming, Statistical Analysis of words and word N-Grams, Supervised Learning
Crime Pattern Analysis through Text Mining	Ananyan (2004)	Tokenisation, Normalisation, Statistical Analysis of words and word N-Grams
From free-text to structured safety management: Introduction of a semi-automated classification method of railway hazard reports to elements on a bow-tie diagram	Hughes et al. (2018)	Tokenisation, Normalisation, Statistical Analysis of words and word N-Grams

indexing method. N-grams are produced in an efficient manner, once the locations of word tokens is known, N-grams of any higher length are produced without the need to iterate over the character data again.

Any task which utilises term counts and frequencies such as Statistical analysis, feature extraction, term weighting, classification, and various machine learning methods will find that these quantities can be determined much more quickly than when using other string matching methods, as demonstrated in fig 4.1. Topic modelling using LDA for example, takes as its input vectors representing the words in a document alongside their frequencies (Jurafsky & Martin, 2020, Chapter 4).

Likewise, the speed of term searches is also improved significantly, again in fig 4.1 it can be seen that when indexing using hashed n-grams, the time taken to locate a list of 900 terms in a corpus of 500000 documents is reduced by a factor of almost 50.

This performance is consistent for queries of any length, as shown in fig 4.4. The number of operations needed to find a word is the same as what is needed to find a 5 word term, rather than increasing with the length of the query as is the case for plain string matching.

A benefit of using fixed length hashes is that tokens can be replaed without the need for memory reallocation, which is computationally expensive when data structures are large. When text data is inserted or replaced, as is the case when tagging parts of the text, if the inserted data differs in size from the data it replaced, all data following that insertion must be moved in memory to accommodate the change. As all hashes are the same size, regardless of the tokens they represent, these can be swapped without requiring any costly reallocation.

A main tenet of this work is to assist text analysis of safety incident reports as the size of these databases continues to increase. The measurements shown in 4.5 are important in this regard. As the total number of records increases, the response time of matching processes increases in a linear fashion. The rate of increase in the hashed N-gram method is far below that of the other string matching methods. As the volume of reports increases, the difference in response time becomes increasingly pronounced.

There is, however a penalty suffered by this method which is not present in the others tested. Because a sorted list of hash values must be maintained in order for a binary search to be possible, insertion of new text into a corpus incurs a preprocessing overhead. Insertion can be implemented either by appending the new hashes to the list, then re-sorting, or, performing a binary search for the new hashes, which will give an insertion point (if the hashes don't already exist), allowing the hashes to be inserted in the correct place. Both methods incur overheads: sorting the array is approximately $\mathcal{O}(n \log n)$ in complexity; while inserting the values into the correct

place in the presorted data which is $\mathcal{O}(m + \log n)$ and potentially a further $\mathcal{O}(n)$ (Amortised) for the chunk of data residing after the insertion point. This is only a hindrance in cases where the corpus is updated more frequently than it is queried, as in other cases the additional preprocessing time is dwarfed by the additional time another matching method would have consumed. For the intended use case, enabling advanced text analysis techniques on close call reports, the performance benefit stands.

- **In what ways can unstructured text be matched against known safety risks?**

Several different ways of doing this are demonstrated across the various studies covered in chapter 2. In the naive Bayes approach, and other supervised learning processes, reports are manually labelled with the safety risks they contain, be they slip, trip and fall events (Hughes et al., 2016) (Toyabe, 2012), or traffic related (Pereira et al., 2013) (D'Andrea et al., 2015). These labelled reports are then used to create a list of terms and their associated frequencies in relation to the labels on any given report. This information is then used to identify known risks in new documents depending on which of those terms is contained within. Another common approach is to produce a term list using domain specific knowledge to associate terms with risks, and classify reports appropriately based on the presence of these terms.

- **How can the performance of N-gram queries be optimised in safety applications, and what is needed to demonstrate that the desired optimisation is achieved?**

The approach used here was to improve the speed of N-gram queries while maintaining completeness of results. This was achieved by creating an index of the text where tokens, both words and N-grams were replaced with fixed length numerical representations, 'Hashes', stored alongside values indicating their location in the text, 'Hashes'. By transforming the data into numerical information

The end result of this is a representation of the text which can be used in the same way as the raw text at the word and N-gram level, but stored in such a way that operations upon it are completed much more quickly than the same operations on plain text.

The speed of this method has already been discussed, but to achieve this while maintaining the same levels of accuracy and precision ('Completeness') as would be found when performing the same operations on raw text, the possibility of hash collisions must be considered. Hashes were used that were sufficiently large enough

to represent every unique token of words and N-grams while keeping the collision probability low enough that it can be reasonably expected that no collisions will take place. In table 3.1.2.1 it can be seen that the probability of a hash collision at the scales used here is 0 even when representing tens of millions of unique tokens. In 4.1 it can be seen that the number of unique tokens in fact starts to decrease when tokens are sufficiently long enough, indicating that there may be an upper bound to the number of hashes required for most applications of this kind. The cause for the increase in this case can be explained by the fact that as a string of words gets longer, the likelihood that that same string will be repeated elsewhere is reduced, so more and more of the tokens are unique as the word count of the tokens increases. In the same way that a very large number or words can be constructed from a 26 character alphabet, a much larger quantity of unique n-grams can be constructed from a smaller number of words. It can also be noted, however, that past a certain point, this increase reaches a plateau (seen around n=8), and begins to fall once more. This happens because many reports are not long enough to contain any n-grams of this length, so the number of unique terms begins to fall.

Fig 4.2 shows that all unique terms in the close call text can comfortably be hashed while maintaining a very low collision probability.

- **What are the boundaries for this framework and in what ways is it inductive?**

Figure 5.1 illustrates the framework used throughout to explain the various qualities which must be balanced when optimising N-gram queries. This is effective in this case as it summarises the the constraints that apply to tasks like this and the ways in which they relate.

As discussed already, completeness is maintained in order to match the behaviour required by the numerous text analysis processes used for safety texts. With that constraint, the implementation was built to maximise speed as much as possible, by combining efficient searching techniques, optimised for tokens token sizes often used for text analysis on text of this type, namely words and N-grams.

The penalty for this, accordingly, is increased consumption consumption of memory. There are a number of reasons for this. Firstly, in order to avoid collisions while allowing for very large corpora of text, it is necessary to use 64bit integers as hash values. Each of these integers is 8 bytes in size, therefore, using the numbers from Table 4.1 as an example, the unique hashes alone account for approximately 792 Megabytes, compared to the original source text, which is 123 Megabytes in size. On top of this, because a hash is stored for every occurrence of the token it pertains to,

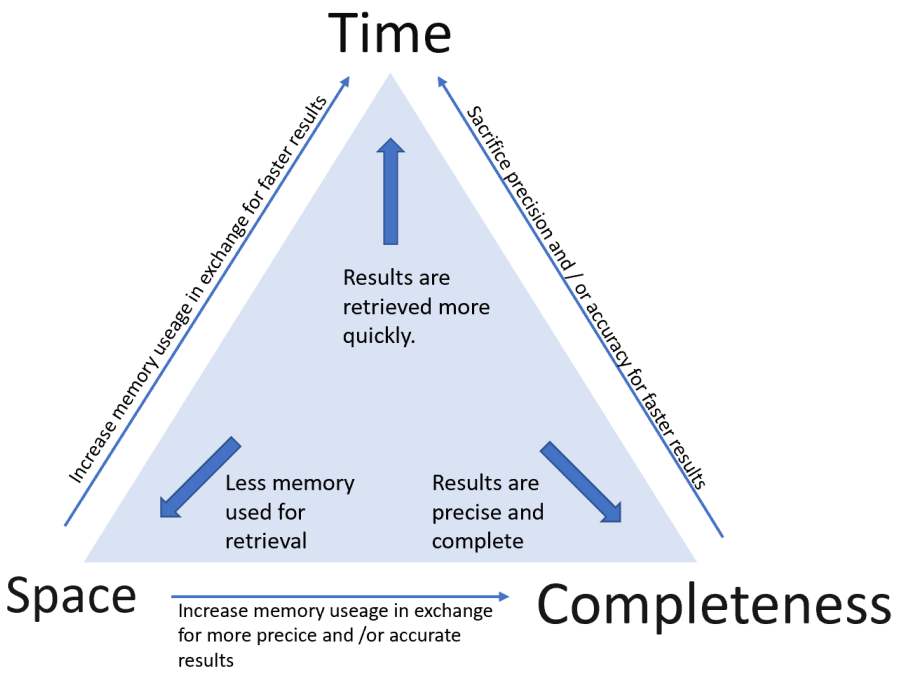

Figure 5.1: Optimisation of matching procedures is a balancing act between, time, space, and completeness

alongside three further integers referring to its location in the text, this amount is duplicated somewhat to around 4 Gigabytes during run-time.

Fig. 4.6 illustrates how the memory usage increases for each doubling of the corpus size. Naive, Boyer-Moore, and C++ STD methods all require virtually the same amount of space, which is to be expected as all of these methods work on the same raw text. The Hashed n-gram method uses considerably more memory to store the processed text. Table. 5.2 shows first a sample of the raw results (in kB) from this test; and second, the same results expressed as a proportion of the result from a corpus half its size. This shows that in almost all cases a doubling of the record count in the corpus size corresponds to less than a doubling in memory usage, so the space complexity is at least proportional to the corpus size.

However, memory usage was not a primary concern in the development of this method, rather, the speed of matching was the priority. In this regard, as demonstrated by the results, the method is successful at performing text matches faster than other frequently used algorithms, whilst also providing some structural benefits (E.g. term frequencies, n-grams) for text analysis proper.

Were the requirements different however, there are ways to tune this implementation to prioritise other factors. Space complexity could be reduced by reducing the hash size. Using a 32 bit hash instead of a 64 bit one halves the required space, and this

Table 5.2: *Top:* A sample of measurements of memory usage (in kB) for doubling sizes of corpora. *Bottom:* The same results expressed as a proportion of the memory useage for a corpus half the size, i.e. the factor by which memory consumption increased when the size of text data was doubled

	8192	***16384***	***32768***	***65536***	***131072***	***262144***
Naïve	3344	5728	11636	22840	43264	80860
Boyer-Moore	3344	5728	11652	22800	43232	80868
C++ STD	3344	5726	11634	22780	43000	80900
Hashed n-grams	77016	151204	327264	660628	1238368	1550852

	8192	***16384***	***32768***	***65536***	***131072***	***262144***
Naïve	1.745303	1.712919	2.031425	1.962874	1.894221	1.86899
Boyer-Moore	1.748954	1.712919	2.034218	1.956746	1.89614	1.870559
C++ STD	1.748954	1.712321	2.031785	1.958054	1.887621	1.881395
Hashed n-grams	1.827275	1.96328	2.164387	2.018639	1.874532	1.252335

can cost either a reduction in completeness, or a reduction in speed. To maintain speed, completeness is reduced as hash collusions introduce incorrect. To maintain completeness, speed is reduced as extra checks must be carried out on th returned results to exclude those which are returned spuriously.

This framework could feasibly be applied to other problems where these same constraints are present. The triangle formed by the interaction of Time complexity, Space complexity, and completeness encloses an area within which all implementations lie. For any given implementation, an area within the triangle can be drawn to provide a qualitative representation of which factors that implementation optimises for. For example the hashed N-gram method outlined here resides mostly along the right vertex, maximising completeness while optimising for speed as much as possible - both at the cost of increased memory usage.

Chapter 6

Conclusions

The method described here constitutes a significant contribution to text processing, particularly in the domain of analysis of unstructured texts. By enabling very fast counting and location of large numbers of terms against large corpora of text sources, this method can expand the scope of text analysis techniques which are demonstrably useful for learning from safety related incident reports, but which, due to the continuously increasing volume of these reports, are often limited in the scale at which they can be applied.

While there are improvements to be made, implementation of exponential search to improve binary search performance for example, the principle has been demonstrated to be effective.

The the methods which have been developed here show that there is scope for a larger throughput of safety text sources into text analysis processes by way of more efficient use of computing resources

The solution offered by the hashed n-gram method uses a number of well established techniques in a novel combination in order to achieve matching speeds which are several orders of magnitude faster than the other alternative methods in common use. Additionally, the matching time scales sub linearly with the size of the text data, so the larger the text is, the greater the performance benefit over alternative methods. This satisfies the initial research goal of improving the performance of queries on these types of unstructured data sources. Rather than allowing hundreds of matches per second, this new method allows millions. This opens up possibilities for the application of analysis techniques which it has previously been impossible or too time consuming to apply to data sources of this size.

The framework used to qualify this optimisation indicates that the hashed N-gram method is optimised for completeness and time complexity at the cost increased memory usage.

One of the key components of current big data risk analysis research is the use of natural language processing to aid in the extraction of information from free form text records. Many natural language processes use word frequencies to aid in the weighting of certain terms in a document, or as a means to categorise segments of text. With a fast method of text indexing such as this one, it is possible to compute these frequencies much more quickly, to the point where results can be collected in real-time from corpora which are simply too large to facilitate this using other methods.

This opens further opportunities for any system which uses text analytics for real time feedback or interaction. By enabling access to larger volumes of text in a smaller time frame, risk management processes would be able to make use of a much wider range of information, and the possibility of more accurate insights.

As the IT transformation of the GB railways moves forward, as it does in countless other domains, the volume of available data will only continue to increase, all of it with the potential to contain valuable information pertaining to safety, reliability, operations and more. In the face of this ever expanding sea of data, methods such as these can only continue to be of increasing utility and importance.

Chapter 7

Future Developments

7.1 Efficiency

There are a number of potential efficiencies worth investigating when implementing a technique such as this. One example would be the selection of hashing algorithm. In the implementation used for collection of the results presented in earlier chapters, the hash function from the C++ standard library was used, as it is well tested and robust, and sufficed in order to demonstrate the principle. However, like the 'rolling hash' used in the Karp and Rabin (1987) method, there may be merit in investigating other hashing algorithms more specialised to a task such as this.

Additionally, further search optimisations such as fractional cascading and exponential search have to potential to not only squeeze out more performance but may even be able to reduce the space complexity in the processed corpus. Methods like this certainly warrent further investigation within the domain of text reporting.

7.2 Tree Structure for Hash Matches

One of the main causes of the relatively large memory footprint of this hashed n-gram method, is the fact that every word in the text, as well as every n-gram up to, in this case, 9, must have a hash and a location stored in order for lookups to work. This results in a very large number of tokens, when in reality the actual count of 'unique' tokens is much smaller.

An alternative is to use a binary tree structure, which only has nodes for unique tokens. Each node can then refer to a list of locations, thus negating the need to store hashes more than once.

Not only does this result in a reduced memory footprint, but the hashes can still be searched using a binary search method, as it is implicit in the structure of the tree.

The method described in the earlier chapters, uses a flat list of tuples which contain the hash, and position information¿ these are sorted, and in order to count how many times a particular token appears, first a binary search is carried out until a single instance is found, then the process reverts to a linear one in order to count the number of times that hash appears.

In contrast, in a tree structure the same binary search for the hash in question is carried out, but once found, the associated value is simply a pointer to a vector containing all locations of the term refereed to by the hash. Counting can now be performed by simply taking the size of this resulting vector.

Another benefit of storing the preprocessed data in this form is that the penalty for data insertion is now much less of a factor. Whereas in the flat structure, most insertions require some amount of memory reallocation, this is no longer necessary, as adding new nodes to the tree is just a case of updating pointers, and new locations can now be added to the end of the associated vectors, allowing the reallocation for this data to be amortized.

7.2.1 Fast Calculation of TF-IDF Weights

TF-IDF weighting is a method of determining the 'specificity' of any given term as it relates to the document which contains it (K. S. Jones, 1972). This method has found use in a number of natural language processing tasks, for example extracting key words from a document, or removing terms which contribute least to the meaning of the text. In order to compute these weights, it is necessary to tokenise the text, and for each unique term, determine the count of each term, both within a document, and within a greater corpus.

By preproccing text data using a tree based, hashed n-gram method, these counts are now part of the data structure itself, and can be obtained at runtime with a single binary search.

References

Abramson, M., & Moser, W. O. J. (1970). More birthday surprises. *The American Mathematical Monthly*, *77*(8), 856–858.

Ananyan, S. (2004, 01). Crime pattern analysis through text mining. In (p. 236).

Apache. (2018). *Apache log4j 2.* Retrieved from `https://logging.apache.org/log4j/2.0/index.html`

Baeza-Yates, R. (2000, 01). Block addressing indices for approximate text retrieval. *Journal of the American Society for Information Science and Technology*, *51*, 69-82. doi: 10.1002/(SICI)1097-4571(2000)51:1⟨69::AID-ASI10⟩3.0.CO;2-C

Baeza-Yates, R., Salinger, e. T., Alejandro", Mannila, H., & Orponen, P. (2010). Fast intersection algorithms for sorted sequences. In *Algorithms and applications: Essays dedicated to esko ukkonen on the occasion of his 60th birthday* (pp. 45–61). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from `https://doi.org/10.1007/978-3-642-12476-1_3` doi: 10.1007/978-3-642-12476-1_3

Baker, C. (2018). *Confidential reporting (ciras).* Retrieved from `https://www.rssb.co.uk/risk-analysis-and-safety-reporting/reporting-systems/ciras`

Bejan, C. A., Xia, F., Vanderwende, L., Wurfel, M. M., & Yetisgen-Yildiz, M. (2012). Pneumonia identification using statistical feature selection. *J Am Med Inform Assoc*, *19*(5), 817–823.

Bird, F. E., & Germain, G. L. (1987). *Damage control: A new horizon in accident prevention and cost improvement.* The Institute Publishing.

Board, D. A. (2016). *Trust access, security and enquiries.*

Boyer, R. S., & Moore, J. S. (1977, October). A fast string searching algorithm. *Commun. ACM*, *20*(10), 762–772. Retrieved from `http://doi.acm.org/10.1145/359842.359859` doi: 10.1145/359842.359859

Brewer, J. (2017). *What is smis?* Retrieved from `https://www.rssb.co.uk/Pages/risk-analysis-and-safety-reporting/What-is-smis.aspx`

Chase, H. S., Mitrani, L. R., Lu, G. G., & Fulgieri, D. J. (2017, 02). Early recognition of multiple sclerosis using natural language processing of the electronic health record. *BMC Med Inform Decis Mak*, *17*(1), 24.

Chazelle, B. (1986). Filtering search: A new approach to query-answering. *SIAM Journal on Computing*, *15*(3), 703-724. Retrieved from `https://doi.org/10.1137/0215051` doi: 10.1137/0215051

Chazelle, B., & Guibas, L. (1986a, 01). Fractional cascading: I. a data structuring technique. *Algorithmica*, *1*, 133-162. doi: 10.1007/BF01840440

Chazelle, B., & Guibas, L. (1986b, 01). Fractional cascading: Ii. applications. *Algorithmica*, *1*, 133-162. doi: 10.1007/BF01840441

D'Andrea, E., Ducange, P., Lazzerini, B., & Marcelloni, F. (2015). Real-time detection of traffic from twitter stream analysis. *IEEE Transactions on Intelligent Transportation Systems*, *16*(4), 2269-2283. doi: 10.1109/TITS.2015.2404431

Dillon, R. L., & Tinsley, C. H. (2008). How near-misses influence decision making under risk: A missed opportunity for learning. *Management Science*, *54*(8), 1425-1440. doi: 10.1287/mnsc.1080.0869

East Coast. (2011). *Trust user guide.*

Feldman, R., & Sanger, J. (2006). *Text mining handbook: Advanced approaches in analyzing unstructured data.* New York, NY, USA: Cambridge University Press.

Gnoni, M., Andriulo, S., Maggio, G., & Nardone, P. (2013). "lean occupational" safety: An application for a near-miss management system design. *Safety Science*, *53*, 96 - 10. doi: 10.1016/j.ssci.2012.09.012

Gulijk, C. V., Hughes, P., Figueres-Esteban, M., El-Rashidy, R., & Bearfield, G. (2018). The case for it transformation and big data for safety risk management on the gb railways. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, *232*(2), 151-163. doi: 10.1177/1748006X17728210

Hughes, P., Figueres-Esteban, M., & Gulijk, C. V. (2016). Learning from text-based close call data. *Safety and Reliability*, *36*(3), 184-198. doi: 10.1080/09617353.2016.1252083

Hughes, P., Figures, M., & Gulijk, C. V. (2014). *Learning from close call events – preliminary report.*

Hughes, P., & Gulijk, C. (2019, 01). An interactive machine-learning method to obtain safety information from free text. In (p. 46-53). doi: 10.3850/978-981-11-2724-3_0055-cd

Hughes, P., Shipp, D., Figueres-Esteban, M., & van Gulijk, C. (2018, 03). From free-text to structured safety management: Introduction of a semi-automated classification method of railway hazard reports to elements on a bow-tie diagram. *Safety Science*, *110*. doi: 10.1016/j.ssci.2018.03.011

Johnson, C. (2002). Software tools to support incident reporting in safety-critical systems. *Safety Science*, *40*(9), 765-780. Retrieved from `https://www.sciencedirect.com/science/article/pii/S0925753501000856` doi: https://doi.org/10.1016/S0925-7535(01)00085-6

Jones, K. S. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, *28*, 11–21.

Jones, S., Kirchsteiger, C., & Bjerke, W. (1999). The importance of near miss reporting to further improve safety performance. *Journal of Loss Prevention in the Process Industries*, *12*(1), 59 - 67. doi: https://doi.org/10.1016/S0950-4230(98)00038-2

Jurafsky, D., & Martin, J. H. (2020). *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition.* Pearson.

Karp, R. M., & Rabin, M. O. (1987, March). Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, *31*(2), 249-260. doi: 10.1147/rd.312.0249

Knuth, D., Morris, J., Jr., & Pratt, V. (1977). Fast pattern matching in strings. *SIAM Journal on Computing*, *6*(2), 323-350. doi: 10.1137/0206024

Knuth, D. E. (1997). *The art of computer programming volume 3: Sorting and searching* (2nd ed., Vol. 3). Addison-Wesley.

Lainoff, S. (1999). Finding human error evidence in ordinary airline event data. In *Seventh international systems safety conference seattle, wa. international systems safety society, unionville, va, usa.*

Lee, I., Iyer, R., & Tang, D. (1991, 6 1). Error/failure analysis using event logs from fault tolerant systems. In *91 fault-tolerant comput. symp.* (pp. 10–17). Publ by IEEE.

Lindberg, D. A., Humphreys, B. L., & McCray, A. T. (1993). The unified medical language system. *Methods of information in medicine*, *32*(4), 281.

Mayfield, J., & McNamee, P. (1998). Indexing using both n-grams and words. In E. M. Voorhees & D. K. Harman (Eds.), *Proceedings of the seventh text retrieval conference, TREC 1998, gaithersburg, maryland, usa, november 9-11, 1998* (Vol. 500-242, pp. 361–365). National Institute of Standards and Technology (NIST).

Miao, Y., Keselj, V., & Milios, E. (2005, 01). Document clustering using character n-grams: A comparative evaluation with term-based and word-based clustering. In (p. 357-358). doi: 10.1145/1099554.1099665

Miner, G., Elder, J., Hill, T., Nisbet, R., Delen, D., & Fast, A. (2012). *Practical text mining and statistical analysis for non-structured text data applications* (1st ed.). Orlando, FL, USA: Academic Press, Inc.

NASA. (2020, Nov). *Asrs program breifing.* Author.

Network Rail. (2016, 11). *Network rail catalogue of railway code systems.* Retrieved from `https://cdn.networkrail.co.uk/wp-content/uploads/2016/11/Catalogue-of-Railway-Code-Systems.pdf`

Newall, M., & Van Gulijk, C. (2019, September 26). Real-time queries on large volumes of safety text. In M. Beer & E. Zio (Eds.), *Proceedings of 29th european safety and reliability conference* (1st ed., Vol. 1, pp. 1800–1803). Research Publishing Services. (29th European Safety and Reliability Conference, ESREL 2019 ; Conference date: 22-09-2019 Through 26-09-2019)

Peladeau, N., & Stovall, C. (2005).

Pereira, F. C., Rodrigues, F., & Ben-Akiva, M. (2013). Text analysis in incident duration prediction. *Transportation Research Part C: Emerging Technologies*, *37*, 177-192. Retrieved from `https://www.sciencedirect.com/science/article/pii/S0968090X13002088` doi: https://doi.org/10.1016/j.trc.2013.10.002

Preshing, J. (2011, may). *Hash collision probabilities.* Retrieved from `https://preshing.com/20110504/hash-collision-probabilities/`

Russell, S., & Norvig, P. (2002). Artificial intelligence: a modern approach.

Shivade, C., Raghavan, P., Fosler-Lussier, E., Embi, P. J., Elhadad, N., Johnson,

S. B., & Lai, A. M. (2014). A review of approaches to identifying patient phenotype cohorts using electronic health records. *J Am Med Inform Assoc*, *21*(2), 221–230.

Swuste, P., Gulijk, C., Groeneweg, J., Guldenmund, F., Zwaard, W., & Lemkowitz, S. (2020, 01). Occupational safety and safety management between 1988 and 2010. *Safety Science*, *121*, 303-318. doi: 10.1016/j.ssci.2019.08.032

Tanguy, L., Tulechki, N., Urieli, A., Hermann, E., & Raynal, C. (2016). Natural language processing for aviation safety reports: From classification to interactive analysis. *Computers in Industry*, *78*, 80-95. Retrieved from `https://www.sciencedirect.com/science/article/pii/S0166361515300464` (Natural Language Processing and Text Analytics in Industry) doi: https://doi.org/10.1016/j.compind.2015.09.005

Toyabe, S. (2012, Dec). Detecting inpatient falls by using natural language processing of electronic medical records. *BMC Health Serv Res*, *12*, 448.

Xu, W., Huang, L., Fox, A., Patterson, D., & Jordan, M. I. (2009). Detecting large-scale system problems by mining console logs. In *Proceedings of the acm sigops 22nd symposium on operating systems principles* (pp. 117–132). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/1629575.1629587` doi: 10.1145/1629575.1629587