Keyphrase Extraction and Grouping Based on Association Rules

by

Xin Li

A Thesis

presented to

The University of Guelph

In partial fulfillment of requirements

for the degree of

Master of Science

in

Computer Science

Guelph, Ontario, Canada

ABSTRACT

Keyphrase Extraction and Grouping Based on Association Rules

Xin Li

University of Guelph, 2013

Advisor:

Professor Fei Song

Keyphrases are important in capturing the content of a document and thus useful for text representation. Keyphrase extraction is often needed for many natural language processing tasks such as Information Retrieval, Document Classification, and Text Summarization. It aims to extract multi-word terms from a collection of documents that more or less correspond to keyphrases. In this thesis, we propose a new method for keyphrase extraction based on association rule mining. Redundant multi-word terms or synonymous terms inevitably make up a big part of keyphrases extracted. With association rules, we can reduce the redundancy by grouping the related keyphrases that have strong co-occurrence frequencies. We further apply our keyphrase extraction and grouping methods to Information Retrieval. By both distinguishing and grouping keyphrases, we are able to achieve improved performance for Information Retrieval.

Key words:

Keyphrase Extraction, Keyphrase Grouping, Association Rules, Natural Language Processing, Information Retrieval

Acknowledgements

I would like to express my deep-felt gratitude to my advisor, Dr. Fei Song of the School of Computer Science at the University of Guelph, for his patient advising, enduring encouragement, and constant support. As an international student, the language is not easy for me. But, with the help from Dr. Song, I did much better with the thesis' structure and proper expression.

I would also like to thank my advisory committee member, Dr. Fangju Wang, for his encouragement to my study and research. I also have to appreciate for his coaching for my swimming skills, which is a unforgettable good lesson.

During the period of study and research in School of Computer Science, I also received help and encouragement from these folks in my lab and school, William Darling, Haochen Zhou, Adnan Duric, Tarek El Salti, Masood Zamani, Jonathan Beer, Melanie Veltman etc. I learned from them in varying degrees. Their passive attitude to research and their passion to life have been influencing me that is more than what I could imagine. I believe those will be the precious treasure in my future life.

Lastly, l would like to thank my wife, Ran Sun, for her encouragement, and my family and other friends for their concern and support during the completion of my degree.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Keyphrase Extraction and Grouping

With the rapid development of the Internet and the World Wide Web (WWW), countless information has been generated but it also presents challenges for people to search and use it effectively. Natural Language Processing (NLP) is an interdisciplinary field of computer science, linguistics, and artificial intelligence. Its ultimate goal is to parse and understand human languages using computer systems. NLP involves research on a wide range of topics, including but not limited to natural language parsing, part-of-speech tagging, information retrieval, word sense disambiguation, automatic text summarization, machine translation, discourse analysis, named entity recognition, natural language generation, question answering, topic segmentation and recognition, and so on. Google, a successful example of NLP, helps people search for the relevant pages among the huge collection of online information on the Internet and WWW.

Although high-level structures could be used to represent natural language documents, many NLP systems, especially those based on machine learning methods, still use low-level elements, such as keywords and keyphrases for text representation. A keyword is a single word or term that partly specifies the content of a speech or document, sometimes called an important word. A keyphrase is usually a sequence of

keywords that follows a grammatical structure and has a specific meaning. Compared with keywords, keyphrases are usually more meaningful since they help capture unique concepts or entities. For example, "hot dog" and "Apple store" cannot be adequately represented by some of the words contained in them. In a library search system, both keywords and keyphrases are used to sort books into categories, write short summaries about books, and index the books for future searches.

The importance of keywords and keyphrases can be seen in research publications and real life applications. For most research papers, authors need to manually provide some keywords and keyphrases, which may or may not appear in the papers. Although the quality of keywords and keyphrases assigned by the authors is high, the number of them is usually small, which will limit the scope of search for users. For modern online life of more than 2 billion people[1] with the explosive growth of WWW, the large majority of web content, such as online news, web pages, blogs, and reviews, have not been assigned keywords and keyphrases. Thus, there is a need to automatically extract keywords and keyphrases for the huge number of documents on the web.

In addition to the extraction of keywords and keyphrases, there is also a need to group them since some may have similar meanings. For example, the full name of "Twentieth Century Fox" is "20th Century Fox Film Corporation". VW is the abbreviation of Volkswagen in the auto industry. The NBA player Michael Jordan's full name is Michael Jeffrey Jordan and some times he is just called Jordan. Finding a way to group these keywords and keyphrases will help us find as many relevant documents as possible for information retrieval.

In our thesis, we will focus on the automatic extraction and grouping of keyphrases along with keywords.

[1] http://www.internetworldstats.com/stats.htm

1.2 Applications

Keyphrases are useful for a wide variety of natural language processing tasks. Here we illustrate their applications in information retrieval, document classification, text summarization, machine translation, and question answering.

- **Information Retrieval** The importance of phrases has never been doubted decades ago for information retrieval. In 1968, Gerard Salton described the application of phrases in the SMART system, an information retrieval system developed firstly at Harvard University and later at Cornell University [55]. Croft, et al.[12] also found that the employment of phrases for information retrieval enriches the representation of queries and documents and improves retrieval effectiveness, as demonstrated by their experiments. Gutwin et al. [24] also applied phrases for browsing their digital library.

- **Document Classification and Clustering** Trappey et al. [64] developed a patent document classification system with phrase extraction as their first step. Hammouda, et al. [25] implemented a phrase-based document clustering system, called CorePhrase. Gutwin et al. [24] also used the phrases for document clustering in their digital library retrieval system.

- **Text Summarization** Berger et al. [6] summarized web pages with the help of phrases in their system, OCELOT, because phrases are both meaningful and mostly available in web pages. D'Avanzo et al. developed their phrase-based short document summarization system, LAKE, in 2004 [18] and 2005 [17]. Riedhammer et al. [53] implemented their phrase-based meeting summarization algorithm. Wan and Xiao exploited neighbourhood knowledge for phrase extraction and used it in single document summarization [70].

- **Machine Translation and Question Answering** Phrase were also used in machine translation [30] by first expanding phrases using retrieved results from a search engine and then finding highly related bilingual information from retrieved results using search engine the second time. For question answering, Hovy et al. [29] first captured the phrases in questions, which are information-bearing, and then expanded the questions using related phrases in order to match more relevant documents in the corpus.

1.3 The Problems

Keyphrase extraction has been studied for a long time as described in Chapter 2. The current solutions can be divided into linguistic, statistical, graph-based, and hybrid approaches. Linguistic approaches use grammatical patterns at morphological, syntactic, or semantic level to extract phrases. Statistical approaches use different association measures to identify frequent phrases from a corpus. Graph-based approaches use tokens and the co-occurrence relations and/or semantic relations among them to construct a graph for finding the most likely phrases. Hybrid approaches are combinations of the approaches mentioned above.

As pointed out by Rizoiu and Velcin [54], the application of phrase extraction is extensive but it is still hard to compare extraction approaches since there is rarely publicly released dataset that can be used as a standard for evaluation [34]. Most researchers present results based on their own manually assigned phrase lists. Even though there are some publicly available datasets with phrase lists, the number and quality of the phrases are quite dependent on the subjective choices [69].

Another problem with keyphrases is that although they are more meaningful than keywords and help distinguish between different concepts, some of them may share

similar meanings like synonyms. For example, "20th Century Fox Film Corporation", "20th Century Fox", "20th Century Fox Pictures", or simply, "20th", "Fox", can all refer to the same company of "20th Century Fox Film Corporation". Failing to group synonymous keyphrases together can lead to low recall in an information retrieval system since documents with different keyphrases will not be matched with the keyphrases in a query even though they may have similar meanings.

Thus, the major challenges for this thesis are how to handle keyphrases extraction and grouping together in a consistent manner and how to evaluate its performance without the need to manually assign keyphrases in a dataset.

1.4 Proposed Solution

In this thesis, we propose our keyphrase extraction and grouping methods, and apply them to information retrieval. Our solution aims to extract keyphrases without using linguistic knowledge and further group them based on the assumption that documents on the same or similar topics share some common keywords/keyphrases. Our solution contains the following major steps:

(1) *Find word sequences of keyphrase candidates that can be efficiently extracted using the BIDE algorithm.*

In the first step, we develop a frequent sequence extraction method based on BIDE (BI-Directional Extension) [71], which is a state-of-art efficient algorithm for finding closed frequent sequences without candidate maintenance and uses less memory but with faster speed to find sequences than previous solutions. However, BIDE aims to find sequences with gaps within them, which is too general for text processing. As a result, we adapt the BIDE algorithm for sequences of consecutive

words, or frequent word ngrams that have no missing words within them from the text, and try to make it more efficient in extracting frequent ngrams. We treat those frequent ngrams as keyphrase candidates.

(2) *Use LocalMaxs to select high quality keyphrases.*

Although keyphrase are usually frequent ngrams, not all frequent ngrams are keyphrases. Consequently, we need a step to extract high-quality keyphrases from keyphrase candidates. LocalMaxs [14] is an algorithm for contiguous multiword term extraction. It selects high-quality keyphrases without the need for complicated linguistic knowledge and is language-independent. This step is important for our solution because the quality of keyphrases is directly related to the next step, keyphrase grouping. However, LocalMaxs is a bit too aggressive in selecting keyphrases and can eliminate useful word sequences that may be needed for some tasks, such as information retrieval where various keyphrases can be used to query the system.

(3) *Use association rules for keyphrase grouping which keeps synonymous keyphrases together.*

In order to balance the quality and coverage of keyphrases, we propose to use association rules to group frequent ngrams with the related keyphrases selected by LocalMaxs. All frequent ngrams in a group are like synonyms in that they co-occur together in about the same set of documents and can be computed from the co-occurrence graphs.

(4) *Apply keyphrase extraction and grouping to information retrieval and evaluate the performance on a standard dataset.*

To measure the effectiveness of keyphrase extraction and grouping, we apply it to

information retrieval and evaluate the system on a standard dataset. Intuitively, keyphrase extraction helps distinguish concepts, and improve precision, while keyphrase grouping helps connect related concepts, and improve recall. So all together, our solution to keyphrase extraction and grouping should improve the overall performance of an information retrieval system.

1.5 Contributions

The main contribution of our research is that we provide a solution for keyphrase extraction and grouping based on association rules. It enables us to explore the relationships between keyphrases and their possible groups so that concepts can be distinguished clearly and related keyphrases can be connected to each other.

Our experiment is the first that attempts to combine keyphrase extraction and grouping in information retrieval. Even though our implementation is based on the basic information retrieval model, vector-space model, it still demonstrates the potential of our approach for information retrieval and will provide the foundation for future research in this direction.

Furthermore, our application of keyphrase extraction and grouping to information retrieval is tested on the standard datasets by NIST (National Institute of Standards and Technology). Such datasets have been widely used to measure the progress of research and development in information retrieval.

1.6 Overview

Chapter 2 provides a background for keyphrase extraction and grouping. In particular, we survey the existing methods based on linguistic, statistical, graph-based, and hybrid approaches. We also give overviews of association rules and information retrieval.

Chapter 3 explains in details the major steps of our solution for keyphrase extraction and grouping, followed by a description of data preprocessing and the application of our solution to information retrieval.

Chapter 4 introduces the datasets and the performance metrics used for our experiments. After that, we present experimental results along with the discussion.

Chapter 5 summarizes our conclusions and points our some directions for future work.

Chapter 2

Background and Related Work

2.1 Keyphrase Extraction

In this chapter, we clarify terms related to keyphrase extraction and survey the existing methods for keyphrase extraction and grouping. We also provide overviews of association rules and information retrieval so that we can develop our solution using association rules and apply it to information retrieval in later chapters.

2.1.1 What is a Keyphrase?

For text representation, keywords, phrases, keyphrases, and terms are closely related. In certain context, they can be used interchangeably, but in other cases, they may have different meanings. To avoid possible confusions in this thesis, we will follow these definitions using and distinguishing them.

Keyword: In scientific articles, a keyword can also be a phrase, but here we limit it to a single word. After a document is tokenized with certain meaningless tokens removed (e.g., numbers and punctuations marks), the remaining tokens are usually considered as keywords and they roughly correspond to words in a dictionary.

Phrase: In Oxford American Dictionary, a phrase is defined as: *1. A small group*

of words standing together as a conceptual unit, typically forming a component of a clause; 2. An idiomatic or short pithy expression: his favourite phrase is "it's a pleasure". Although intuitive, this definition is not easy to implement from the computing perspective.

A more helpful definition given in the well-known linguistic book by David Crystal [13] is that a phrase is commonly formed by more than one word without "subject-predicate structure typical of clause", but between clause and word from structural hierarchy[1], such as noun phrase(NP), prepositional phrase(PP), adjective phrase(AP), etc.

Keyphrase: Although it often means the same as a phrase, a keyphrase usually denotes an important phrase, as measured by certain criteria such as frequently occurring phrases or noun phrases [5, 67, 70].

Term: When both keywords and keyphrases are used for indexing, they are generally referred to as terms. There are two kinds of indexing [67]: *back-of-the-book indexing* and *search engine indexing*. The back-of-the-book indexing is used by librarians to identify a limited number of important words and phrases for human browsing. Search engine indexing is used in information retrieval [67] to gather all words and/or phrases in order to support search for a wide range of queries.

2.1.2 Keyphrase Assignment vs. Keyphrase Extraction

According to Frank et al. [21], there are two purposes to generate phrases for documents:

Keyphrase Assignment: selects keyphrases from a training dataset or a fixed vocabulary, and assigns them to documents as their tags, as is commonly done in

[1]Government-binding theory is a modular theory of Standard Grammar Theory, in which clause is a special kind of phrase that we will not concern in this thesis

scientific articles.

For keyphrase assignment, phrases in a controlled vocabulary are assigned to a document according to whether they help classify the document [67, 74]. Leung and Ken implement a statistical model for phrase assignment that makes use of the syntactic properties to distinguish documents for assigning phrases [38]. The problem is that phrases are controlled, which means that they are predetermined, usually from the training documents, but not necessarily seen in the test documents. To expand the coverage, the training documents need to be extended regularly. However, manually assigning keywords and phrases is both time consuming and tedious.

Keyphrase Extraction: is usually done automatically to identify meaningful and representative phrases, for a document collection. Keyphrase extraction typically has two phases: candidate selection and candidate refinement. In the selection phase, any possible phrases in new documents are considered as long as they satisfy certain conditions, either heuristic or statistical. In the refinement phase, morphological[2], syntactic[3], semantic[4] or statistical information is employed to help generate high-quality keyphrases.

2.2 Existing Approaches for Keyphrase Extraction

Keyphrase extraction has actively been studied due to its applications to a wide range of natural language processing tasks, especially with the increasing availability of online documents and resources such as the Web, Wikipedia, and WordNet. We can broadly divide the current research into linguistic, statistical, graph-based, and hybrid approaches.

[2]Study of the word structure made of prefixes, stems, and suffixes.
[3]The way in which linguistic elements are put together to form phrases and sentences.
[4]Study of the meaning for words, phrases, sentences, and even symbols and signs.

2.2.1 Linguistic Approaches

Linguistic approaches, as used in early research for keyphrase extraction, employ linguistic structures at POS, morphological, syntactic, or semantic level to extract keyphrases [54].

In English, words can be divided into nine Part-Of-Speech(POS) categories, including Nouns, Pronouns, Verbs, Adverbs, Articles, Adjectives, Prepositions, Conjunctions, and Interjections. Several POS taggers are freely available and can link words with POS tags at a high accuracy of 95% or above. As a result, people often use a POS tagger as a step for many NLP tasks. After POS tagging, certain grammatical patterns such as noun-noun and adjective-noun, can be used to extract keyphrases, as is the case for $LEXTER$ term extraction system by Bourigault [7].

Morphology is a branch of linguistics that studies how words are decomposed into *morphemes*, such as prefixes, roots/stems, and suffixes. For instance, *harm-ful-ness* is made up of one *root* and two *suffixes*. Morphological information helps relate the derivatives of a root to POS categories. For example, *dogs* and *dog* are both nouns. FASTR [33] used both syntactic and morphological information for keyphrase extraction.

Syntax specifies the rules on how words are combined together to form phrases and sentences in a language. Syntactic information helps relate different structures of the same content words to each other in text. For example, given the sequence "technique for phrase extraction measurement", we can also extract the phrase "phrase extraction measurement technique", which can be seen as a variant of "technique for phrase extraction measurement" at the syntactic level.

Systems that use semantic information for keyphrase extraction usually acquire semantic features from a constructed thesaurus for the language. Medelyan and Witten

[49] obtain the connectivity level of candidate keyphrases based on an agriculture thesaurus. Ercan and Cicekli [19] acquire semantic information using lexical chains based on WordNet. A lexical chain connects a set of phrases through semantic relationships.

The advantage of linguistic approaches is that they can generate less noisy output, but the disadvantage is that they are not easily adaptable to a new domain, since the rules are usually constructed manually or created for a specific domain. In addition, web pages usually do not follow a specific style and the structures are varied widely, from well-written to poorly-structured, making it difficult for linguistic approaches. Nevertheless, linguistic approaches are still applicable in restricted areas [54] or used along with other approaches, like hybrid approaches [44].

2.2.2 Statistical Approaches

Statistical approaches can be further divided into three types: simple statistical, Machine Learning, and clustering methods.

Simple statistical methods use measures of ngrams [10], such as term frequencies [45], TF×IDF [56], co-occurrences [47], or term independence [9] to find phrases. Such methods are easy to use and good for general applications. However, the precision is usually low.

We can also automatically learn phrases from a training dataset using machine learning methods such as Naive Bayes [74], Maximum Entropy Modeling [39], and SVM (Support Vector Machine) [80]. Two such systems are GenEx [66] and KEA [74]. This type of approach can extract unknown phrases in a new dataset, but it often suffers from sparse training data and over-learning problems.

Clustering methods involve the construction of multi-classifiers to extract phrases automatically, like the Bagging algorithm [31]. It can increase the precision perfor-

mance, but training multi-classifiers can takes long time to complete.

In the following, we selectively review some representative keyphrase extraction systems based on statistical approaches. We begin with GenEx, which is not publicly available, followed by Kea, and then the simple statistical approach of LocalMaxs.

GenEx was proposed by Peter Turney [66] in 1999. The algorithm includes two components: Genitor and Extractor. The Extractor takes documents as input and produces the weighted phrases with twelve tuned parameters. Genitor, which is a genetic algorithm, is in charge of tuning the parameters. Phrases are evaluated by scores based on a number of features (such as frequency, length, first occurrence of a candidate, if_proper_noun etc.) Those features that are more likely to match the manually labelled phrases in the training documents are kept. GenEx extracts phrases of up to three words without stop words and punctuations in them from the stemmed documents. According to [5], GenEx achieved a good result during a web evaluation against human judgements [5]. However, the content and the size of the test data are not mentioned; nor are the common measures such as precision, recall and f-value, are used. It just uses three categories ("good", "bad" and "no opinion") to show the results.

KEA is a Naive Bayes classifier based phrase extraction system developed by Witten et al. [74]. It is frequently cited work on phrase extraction. KEA uses two features to determine the availability of phrases: $TF \times IDF$, a numerical statistic that measures how important a word is to a document in a corpus, and the first occurrence, the distance from the beginning of a document to the first occurrence of a phrase measured by the number of words in between divided by the number of words in the given document. KEA treats phrase extraction as a classification task and uses Naive Bayes classifier along with a few heuristics. Because of the Naive Bayes classification

algorithm, KEA only takes into account the simple statistical properties of candidate phrases and considers those candidates to be independent of each other.

For each phrase, $TF \times IDF$ is defined as follows:

$$TF \times IDF = \frac{freq(P, D)}{size(D)} \times log_2 \frac{N}{df(P)} \tag{2.1}$$

where $freq(P, D)$ is the frequency of phrase P in the document D; $size(D)$ is the number of words in document D; $df(P)$ is the number of documents in which P occurs in a document collection; N represents the total number of documents in the collection.

For word i in document D, the distance is defined as follows:

$$d_i = \frac{word\_number\_precede\_word(i)}{size(D)} \tag{2.2}$$

Naive Bayes classifier assigns an object to a category using the *maximum a posteriori method*[48]. Given a set of categories $C = (c_1, c_2, ..., c_m)$ and a set of conditions $(b_1, b_2, ..., b_n)$ for an object, the corresponding category C_{max} is determined as follows:

$$C_{max} = argmax_{c_i \in C} p(c_j | b_1, b_2, ..., b_n) \tag{2.3}$$

Applying the Bayes' theorem, we have:

$$C_{max} = argmax_{c_i \in C} \frac{p(c_j)p(b_1, b_2, ..., b_n|c_j)}{p(b_1, b_2, ..., b_n)} = argmax_{c_i \in C} p(c_j)p(b_1, b_2, ..., b_n|c_j) \tag{2.4}$$

since $p(b_1, b_2, \ldots, b_n)$ does not change for different categories C_i.

In practice, the probabilities $p(b_1, b_2, ..., b_n|c_j)$ are hard to estimate but the independent assumption allows us to simplify it to $p(b_1, b_2, ..., b_n|c_j) = \prod_{i=1}^{n} p(b_i|c_j)$. Consequently, we can make a decision using the following simplified formula [41]:

$$C_{max} = argmax_{c_i \in C} p(c_j) \prod_{i=1}^{n} p(b_i|c_j) \tag{2.5}$$

In KEA, Naive Bayes model uses two features, t (for TF×IDF) and d (for distance) and takes the following form:

$$P[yes] = \frac{Y}{Y+N} P_{TF \times IDF}[t|yes] P_{distance}[d|yes] \tag{2.6}$$

where Y is the number of candidate phrases identified by authors in training documents; N is the number of candidate phrases not identified by the authors. Y and N can be replaced by $Y+1$ and $N+1$ in case of zero probabilities. Similar formula can be made for $P[no]$. Then, all phrases will be ranked by equation (2.7):

$$p = \frac{P[yes]}{P[yes] + P[no]} \tag{2.7}$$

As reported in [21], KEA's performance is comparable to GenEx, especially for phrases used frequently in a particular domain. For both GenEx and KEA, only precisions are given, which are 29.0% and 27.0% when five phrases are extracted for each document, and 17.7% and 18.3% when fifteen phrases are extracted for each document with GenEx and KEA, respectively.

LocalMaxs was first proposed by Silva and Dias [15] for contiguous multiword term extraction and later extended for non-contiguous multiword term extraction by Silva et al. [14]. The algorithm makes use of the lexical connection within the multiwords, which are called keyphrases in our thesis. Silva [15] call the connection, "glue value", which is usually strong within true phrases, such as "Twentieth Century Fox Film Corporation" but weak within arbitrary multiword terms such as "if his cats", "I will be". Based on this intuition, Silva et al. proposed their LocalMaxs algorithm as follows

[14]:

Assume W is a n-gram phrase $\{w_1, w_2, \ldots, w_n\}$ and Y is a $(n+1)\_gram$ extended from W by adding another word w_{n+1} within, before or after W to get $\{w_1, w_2, \ldots, w_{n+1}, \ldots, w_n\}$, $\{w_{n+1}, w_1, w_2, \ldots, w_n\}$ or $\{w_1, w_2, \ldots, w_n, w_{n+1}\}$. Similarly X is a $(n-1)$-gram reduced from W by removing a word. Then, W will be chosen as a keyphrase if:

$$length(W) = 2 : g(W) > g(Y)$$

$$length(W) > 2 : g(X) \leq g(W) and g(W) > g(Y)$$

Figure 2.1 shows an example from [14] that clearly illustrates the idea of LocalMaxs.

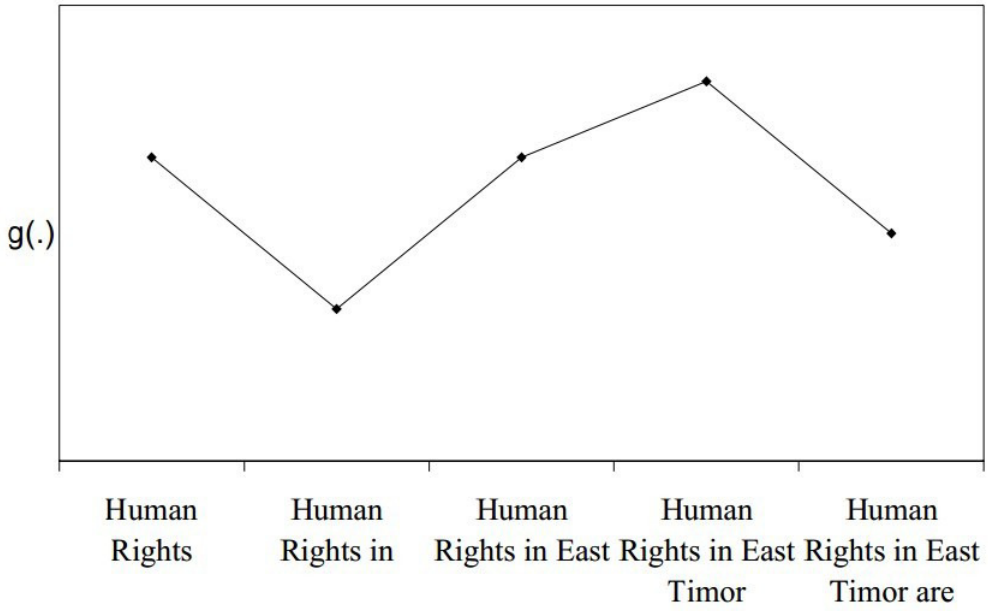

Figure 2.1: Example of "glue" values within a n-gram [14].

We can see that the glue values of *"Human Rights"* and *"Human Rights in East Timor"* are locally higher than their $(n-1)$-grams and $(n+1)$-grams. As a result, they will be extracted as keyphrases. The advantage of LocalMaxs is that it needs neither complex linguistic information nor empirically obtained thresholds for keyphrase

extraction.

2.2.3 Graph-Based Approaches

Graph-based approaches decide the importance of words inside a graph based on the overall information recursively collected from the graph. This approach was firstly applied to keyphrase extraction by Mihalcea and Tarau in 2004 [50]. In a graph-based approach, vertices (or nodes) are tokens (words or phrases) and edges are the co-occurrence relationships between vertices. Given a directed graph $G = (V, E)$, where V represents the set of vertices and E for a set of edges, $In(V_i)$ is the the set of vertices that point to vertex V_i, and $Out(V_i)$ is the set of vertices that V_i points to. Then, the score of a vertex V_i is defined as follows in PageRank [8]:

$$S(V_i) = (1 - d) + d * \sum_{j \in In(V_i)} \frac{1}{|Out(V_j)|} S(V_j) \tag{2.8}$$

where d a parameter that can be set between 0 and 1. For example, this value was set to 0.85 in Brin and Page[8].

In TextRank[50], nouns and adjectives are the only two kinds of candidates used. Some derived algorithms of TextRank try to improve its accuracy [69, 77, 16]. Wan and Xiao (2008) improves it by constructing a graph with highly similar documents [69]. Zesch proposes a lexical semantic graph that represents edges as semantic relatedness between the tokens in a document in 2010 [77]. Semantic relateness measures the relatedness of terms/concepts, which was first introduced by Strube and Ponzetto [60]. Tarau develops a graph-based phrase extraction system using Wikipedia as a semantic resource [50].

Besides the Wikipedia, another semantic resource is WordNet (a lexical database for English). It organizes words into synonym groups, called synsets, and link them by

semantic relations such as IS-A and HAS-PART [59].

2.2.4 Hybrid Approaches

Hulth (2003) developed a supervised machine learning algorithm for phrase extraction with linguistic knowledge added. In her experiments, she parses the corpus and uses Part-of-Speech tags to extract Noun Phrase(NP)-chunks. The quality of parsing directly affects the result of phrase identification. After semantic knowledge is taken into account, better result was achieved in the experiment, resulting in a hybrid system [44].

2.3 Keyphrase Grouping

Grouping documents into natural clusters helps improve text classification and discover how documents are inherently related for a given document collection. It is a popular technique for text mining and the results are useful for data analysis or form an organization for browsing and retrieving documents efficiently.

Similarly, we can also group phrases into clusters so that similar phrases can be treated as synonyms to increase the recall performance of text processing systems.

In text processing, phrase grouping/clustering is based on the fact that similar documents share many phrases in common. As a result, we can use the co-occurrence patterns of the phrases to group related phrases together. Note that phrase grouping does not have to rely on high co-occurrence frequencies; it only assumes that documents on the same or similar topics share many common phrases [76].

Zamir [76] tried to cluster documents based on the phrase-clusters. In the experiments, Zamir used suffix trees to find all maximal phrase clusters and ranked them

by scoring. Those with higher scores will be kept and later merged with other phrases based on the overlaps of their document sets.

Phrase grouping was also used by Latiri et al. for query expansion [36], where the concept iceberg lattice is used to find the maximal term groups that share the same documents so that we can add the terms from the same group to expand queries that do not have enough information. As far as phrase grouping is concerned, this method is similar to that proposed by Zamir [76].

Kuhn et al. [35] used phrase clustering for statistical machine translation. Based on their survey results that phrase-based translation systems are better than word-based systems, they tried to group phrases based on two types of similarity metrics between phrases: count-based metric and edit-based metric. The former relies on the co-occurrence counts of phrases, such as "law" and "court" which often appear together in documents about law. The latter focuses on the make up of the phrases, such as "International Business Machine" and "Business Machine". They showed better result and attributed the improvement to the count-based clustering. However, they found that the result can be unreliable when the size of training data is small.

The phrase grouping/clustering methods are also used in other areas, such as Discriminative Learning [40] and Mandarin Chinese fluent speech [65].

In this thesis, we also use concept iceberg lattice to group phrases [61, 62, 36], but we further build co-occurrence graphs so that more phrases can be grouped as synonyms.

2.4 Association Rules

Association rules are one of the important techniques for data mining, and in this thesis, we intend to use them for keyphrase extraction and grouping.

2.4.1 What are Association Rules?

Association rules capture important relations between variables in large databases. Association rule mining was first proposed by Agrawal, Imielinski, and Swami in 1993 [1]. Given the large number of records in databases, association rule mining focuses on finding the rules among the variables that satisfies the minimum support and minimum confidence constraints specified by users [42]. For example, in a department store, customers who are buying computers, are likely to also buy anti-virus software. If the store knows such information, it can increase the sales by arranging them close to each other on the shelves so that the customers can find them easily.

Following the notations in [27], we let $I = \{i_1, i_2, ..., i_n\}$ be a complete set of items (called itemset) and let $T = \{t_1, t_2, ..., t_m\}$ be a set of transactions, where each $t_k \subseteq I$. Suppose that there are two subsets of I: A and B. The basic concept of association rule follows the form $A \Rightarrow B$, where $A \subseteq I, B \subseteq I$, and $A \cap B = \emptyset$. The percentage of co-occurrences of A and B denoted as $A \cup B$ in T defines **support($A \cup B$)**. The **confidence** of the from $A \Rightarrow B$ is the probability of B given A as shown in the following:

$$confidence(A \Rightarrow B) = P(B|A) = support(A \cup B)/support(A)$$

Support helps find itemsets with a sufficient frequency and confidence is for reliable rules. The higher the confidence, the closer the two sides of a rule. The *strong* rules are defined as the rules that meet both a minimum support threshold and a minimum confidence threshold.

There are two major steps for association rule mining [27]:

(1) **Frequent Itemset Generation**: find all frequent itemsets that meet a minimum support threshold.

(2) **Association Rule Generation**: find all strong association rules for the frequent itemsets generated from step (1) above.

2.4.2 Frequent Itemset Generation and its Extensions

Apriori(T, min_sup)
Input: a set of transactions T, a minimum support threshold min_sup
Output: frequent itemsets L in the database
Notations: L_i: the set of i-itemsets
C_k: the set of k-itemset candidates
C_t: the temporary set of candidates

```
1: procedure
2:     L_1 ← {FindFrequent1 − Itemset(D)};
3:     while L_{k−1} ≠ ∅, k ← 2 do
4:         C_k ← {a ∪ b : a ∈ L_{k−1} ∧ b ∈ L_{k−1} ∧ b ≠ a};
5:         for each transcations t ∈ T do
6:             C_t ← {c|c ∈ C_k ∧ c ⊆ t};
7:             for candidates c ∈ C_t do
8:                 count[c] ← count[c] + 1;
9:             end for
10:        end for
11:        L_k ← {c|c ∈ C_t ∧ count[c] ≥ min_sup};
12:        k ← k + 1;
13:        C_t ← ∅;
14:    end while
15:    return ⋃_k L_k;
16: end procedure
```

Figure 2.2: Apriori Algorithm Adapted from Wikipedia

Apriori is a classical algorithm for finding frequent itemsets in association rule learning, which is firstly proposed by Agrawal et al. [2]. It includes two general steps: join and prune. Apriori first identifies frequent 1-itemsets in a database and extend them by joining each 1-itemset with a single item to gain 2-itemsets. After that, it prunes those 2-itemsets that do not satisfy the minimum support min_sup. Such a process is repeated level-by-level until no frequent k-itemsets can be found. One property for Apriori is that any nonempty subsets of a frequent itemset must be

frequent as well, which also means that any supersets of an infrequent itemset must be not frequent [26]. The pseudo code of the Apriori algorithm is listed in Figure 2.2.

However, Apriori is costly in both memory and time: (1)memory: generating and maintaining a huge number of candidate itemsets. For example, there are 10^4 frequent 1-itemsets, and more than 10^7 2-itemsets generated by Apriori. Let alone the 3 and 4-itemsets; (2) time: repeatedly going through the database and checking the exactly matched candidates will cause many needless scans [26].

Apriori is an example of breadth-first search. Another algorithm FP-growth uses depth-first search and is more efficient in mining frequent itemsets, as reported in [4].

FP-growth algorithm is proposed by Han, Pei, and Yin [28] in 2000, which adopts depth-first search. It follows the divide-and-conquer principle and discovers frequent itemsets without candidate generation. It builds a compact data structure, FP-tree, from which frequent itemsets can be mined directly by calling $FP\_growth(FP\_tree, Null)$.

Each node in FP-tree has four elements: label, frequency, parent(single pointer), and children(multi-pointers). The Head Table is used to store all single items based on the alphabetical order and to find all corresponding items in FP-tree during the mining process. The *node-link* is a linked list that starts from a single item in the Head Table and extends to all occurrences of the same single item in the FP-tree in the order that the tree is recursively built.

The construction of FP-tree is explained as follows.

1. Find frequent single items: read the database once to find the frequent single items. The infrequent ones will be discarded.

2. Sort the frequent single items into the alphabetical order and add them to the Head Table.

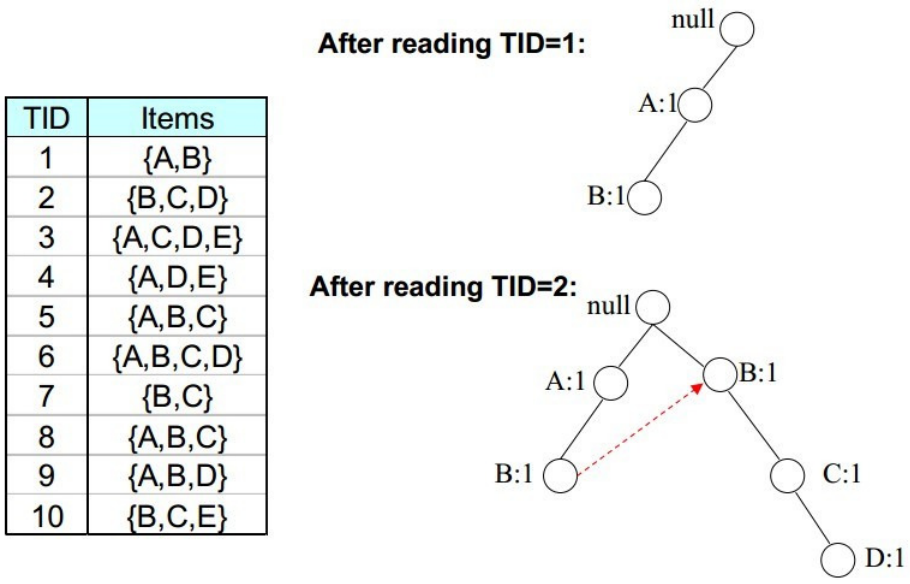

Figure 2.3: Construction of an FP-tree

3. Construct the base of FP-tree: the root node labelled with "null".

4. Build the FP-tree: read each transaction t, and sort its items into the alphabetical order. Let the sorted items be $[p|P]$, where p represents the first item and P, the remaining items. Call the function $insert\_tree([p|P], T)$, where T is the root of a subtree of the FP-tree as the given items are being processed. If T has a child c with $c.name = p.name$, then we increase c's count by 1; otherwise, we create a new child node c' for T with a count of 1. Meanwhile, we extend the node-link for the new child node. After that, we call $insert\_tree(P, T')$ recursively where T' is either c or c' as determined above until P is empty. Please see Figure 2.3 for the processing of the first two transactions in the database.

The FP-tree after all transactions are processed is shown in Figure 2.4. In Figure 2.5, we provide the pseudo code for the FP-growth algorithm. Given a FP-tree, we can use the FP-growth algorithm to generate all frequent itemsets. In Table 2.1, we show the intermediate and final results of the FP-algorithm for the corresponding example.

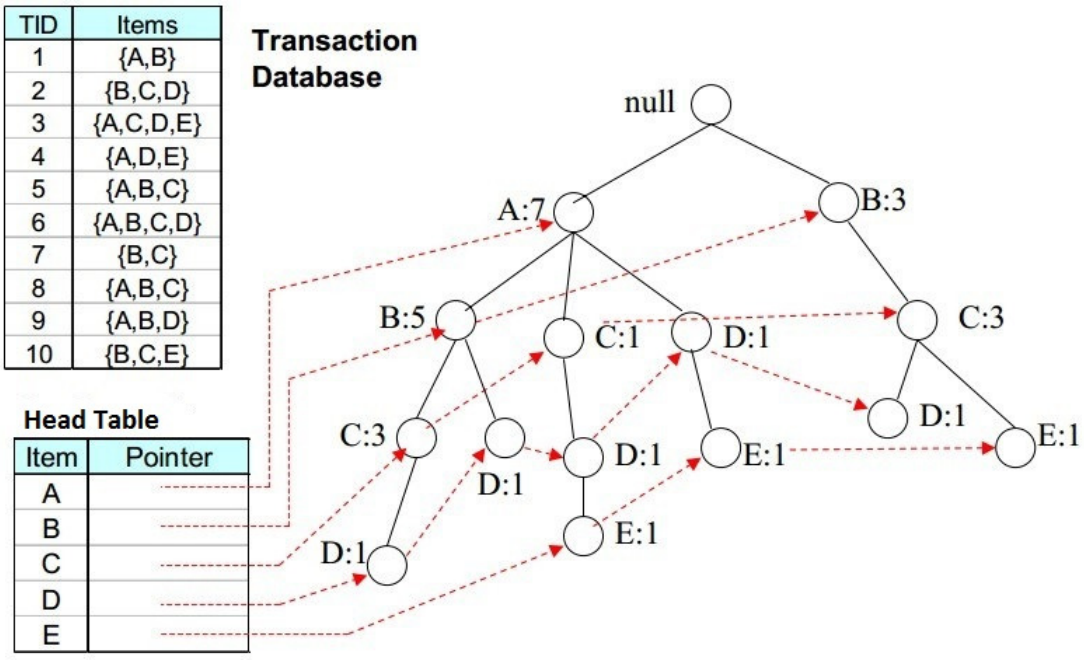

Figure 2.4: Result of Construction of FP-tree

Table 2.1: Mining the FP-tree by Creating Conditional (Sub)-Pattern Bases

Item	**Conditional Pattern Base**	**Conditional FP-tree**
E	{{A,C,D:1},{A,D:1}, {B,C:1}}	(A:2,D:2),(C:2)
D	{{A,B,C:1},{A,B:1}, {A,C:1},{A:1},{B,C:1}}	(A:2, B:2, C:1), (A:2, C:1), (B:1, C:1)
C	{{A,B:3}{A:1}{B:3}}	(A:4,B:3),(B:3)
B	{A:5}	(A:5)
Item	**Frequent Itemsets Generated**	
E	{E},{D,E},{A,D,E},{C,E},{A,E}	
D	{D},{C,D},{B,C,D},{A,C,D},{B,D},{A,B,D},{A,D}	
C	{C},{B,C},{A,B,C},{A,C}	
B	{B},{A,B}	
A	{A}	

The best case of FP-tree is that all of the transactions are the same, resulting in only one branch in FP-tree. The worst case is that each transaction is different from the others and in such case, the size of FP-tree might be at least equal to the size of the given database. The pointers between the nodes and the associated counters will

FP_growth(D, min_sup)
Input: A database D, a minimum support threshold min_sup
Output: The set of frequent itemsets
Notations:
FP: frequent patterns; PB: pattern base
PBs: the set of pattern bases

```
1: procedure
2:     for each item i in Head Table do
3:         for each node n on node-link of i do                ▷ track i's node-link
4:             while node ≠ null do
5:                 (upwardly) locate n's parent p
6:                 PB = PB ∪ p
7:                 node ← p
8:             end while
9:             PBs ← PB;
10:        end for
11:        ConditionalFP_tree ← PBs
12:        FP ← ConditionalFP_tree
13:    end for
14: end procedure
```

Figure 2.5: FP-growth Algorithm for Discoversing Frequent Itemsets without Candidate Generation.

also cause the increase of the required memory.

The pros of FP-growth are that it only scans the database twice without candidate generation and it is much faster than Apriori. The cons are that FP-tree takes large memory and is expensive to build. However, after FP-tree is built, frequent itemsets can be extracted easily and efficiently.

Frequent itemsets are useful for general data mining. For text processing, however, we are interested in extracting keyphrases, which can be seen as ordered itemsets or sequences of words [76].

BIDE is proposed by Wang and Han [71] for frequent sequence generation. To some extent, it solves the problem of large memory overhead as required by Apriori and FP-growth Algorithm by introducing *Pseudo-Projection Databases*. A *pseudo-*

Frequent Sequence Enumeration(FS, SDB)
Input: a sequence database SDB, a minimum support threshold min_sup
Output: the complete set of frequent sequences, FS

1: **procedure** $FrequentSequenceEnumeration(FS, SDB)$
2: $FS = \emptyset$;
3: $F1 = LocallyFrequentItems(SDB, min\_sup)$;
4: **for** each $f1$ in $F1$ **do**
5: $SDB^{f1} = PseudoProjectedDatabase(SDB, empty\_sequence)$;
6: $FrequentSequences(SDB^{f1}, f1, FS, min\_sup)$;
7: **end for**
8: **end procedure**

Frequent Sequences ($S_p\_SDB, S_p, FS, min\_sup$)
Input: a projected sequence database S_p_SDB, a prefix sequence S_p,
a minimum support threshold min_sup
Output: the current set of frequent sequences, FS

9: **procedure** $FrequentSequences(S_p\_SDB, S_p, FS, min\_sup)$
10: **if** $S_p \neq empty$ **then**
11: $FS = FS \cup S_p$;
12: **end if**
13: $LFI = LocallyFrequentItems(S_p\_SDB)$; ▷ LFI: locally frequent item
14: **for** each i in LFI **do**
15: ${S_p}^i\_SDB = PseudoProjectedDatabase(S_p\_SDB, i)$;
16: ${S_p}^i = S_p + i$;
17: $FrequentSequences(SDB^{{S_p}^i}, {S_p}^i, FS, min\_sup)$;
18: **end for**
19: **end procedure**

Figure 2.6: Frequent Sequence Enumeration Algorithm

projection database dramatically reduces the cost of memory consumption by using *pointers* to the original sequences in a database along with the *offsets* that mark the starts of the related subsequences for a new database. Using the transactions in Figure 2.3 as an example, $TID6\{A, B, C, D\}$ and $TID8\{A, B, C\}$ become $TID6\{C, D\}$ and $TID8\{C\}$ after the pattern $\{A, B\}$ is processed. Instead of creating a new database for them, a pseudo-projection database will simply include entries: $(6, 2)$ and $(8, 2)$, indicating that the sequences are from $TID6$ and $TID8$ of the original database, and both sub-sequenes start from the second position in the original sequences.

BIDE is based on FSE(Frequent Sequence Enumeration) algorithm which finds all frequent sequences in a database. FSE is a depth-first algorithm. Within the pseudo code (Figure 2.6), the *locallyFrequentItems(SDB, min_sup)* is used to find the frequent 1-items with supports greater than min_sup from a given sequence database SDB. Another method, *PseudoProjectedDatabase(SDB, seq)* helps create a pseudo projected database given a SDB and a pattern *seq*.

Frequent Closed Sequences

For general association rule mining, it is important to find *frequent closed itemsets*(FCIs) since they can dramatically reduce the number of generated frequent itemsets.

An itemset X is closed if and only if there is no proper super-itemset that has the same support count as X. FCIs are both closed and frequent. With an FCI, we can easily generate all of its sub-itemsets by defining related association rules. Wang and Han [71] points out that mining FCIs is more concise and efficient. Accordingly, they define frequent closed sequences and propose the BIDE algorithm to extract them efficiently.

The full name for BIDE is BI-Directional Extension, which is a novel sequence closure checking scheme. It has two steps: forward-extension checking and backward-extension checking. Forward-extension checking is done by examining a projected database to see if any item or items can be added to a given pattern to form a frequent sequence. If so, add one or more items to the present pattern and repeat this process until no items can be added to the present pattern. Backward-extension checking explores if any item can be added between each of i_n and i_{n+1} ($1 < n < pattern\_size$) words in a given pattern that forms a frequent sequence in the projected database.

As an advanced algorithm for mining frequent closed sequences, BIDE checks

whether a shorter sequence is closed against the original sequences (replaced by pseudo projected databases later on) and decides if there is no hope to grow the sequence to generate any frequent closed sequences. *BackScan*, as an optimizing step in BIDE, stops growing a sequence that is guaranteed to not produce any closed sequences, and thus aggressively prunes the search space and improves the time performance of the BIDE.

Furthermore, another optimizing step, *ScanSkip* is used to stop backward scanning when it finds an empty set in checking i_n and i_{n+1}. For example, if we have a pattern $\{A, D\}$ in Figure 2.3 with the sequence database $TID3(A, C, D, E)$, $TID4(A, D, E)$, $TID6(A, B, C, D)$, and $TID9(A, B, D)$. When the system checks the $TID4$ for i_1 and i_2, it finds nothing between A and D. Thus, the systems will stop checking the remaining sequences. This can be used inside *BackScan* to speed up the search process. More details can be found in Wang and Han [71].

The main contribution of BIDE is that it reduces the memory consumption by using pseudo projected databases. It is also efficient in that it adopts a sequence closure checking mechanism based on BI-Directional Extensions. Through search space pruning and optimization, it can speed up the mining process considerably.

2.4.3 Association Rule Generation

Generating association rules will depend on what kinds of rules to use. Compared with the first step, this step needs much less time and memory space.

Rules are divided into two types: exact association rules and approximate association rules. The exact rules have the confidence equals to 1. The approximate association rules have confidence less than 1. [75, 36]. The challenge is how to find all the strong association rules from a large number of frequent itemsets.

Extra constraints can be added to find association rules more efficiently. For example, user-defined constraints are added in a post-processing step by Liu [81]; Guillet and Hamilton [23] use quality measures to enhance the mining step, like lift(which is another simple measure for the importance of a rule); Galois closure and more advanced techniques are used in mining a limited number of rules by Ganter and Will [22] and Latiri et al [36].

2.4.4 Grouping Ngrams from an Iceberg Lattice

An iceberg lattice can be used to group related itemsets for large databases. It provides an intuitive visualization for concept analysis, since all frequent itemsets can be organized into a hierarchical and dense representation. Tracing back, an iceberg lattice is based on Formal Concept Analysis(FCA), which was introduced in the 1980s for concept formalization. It has been used as a representation for data analysis and knowledge discovery, etc [61, 62].

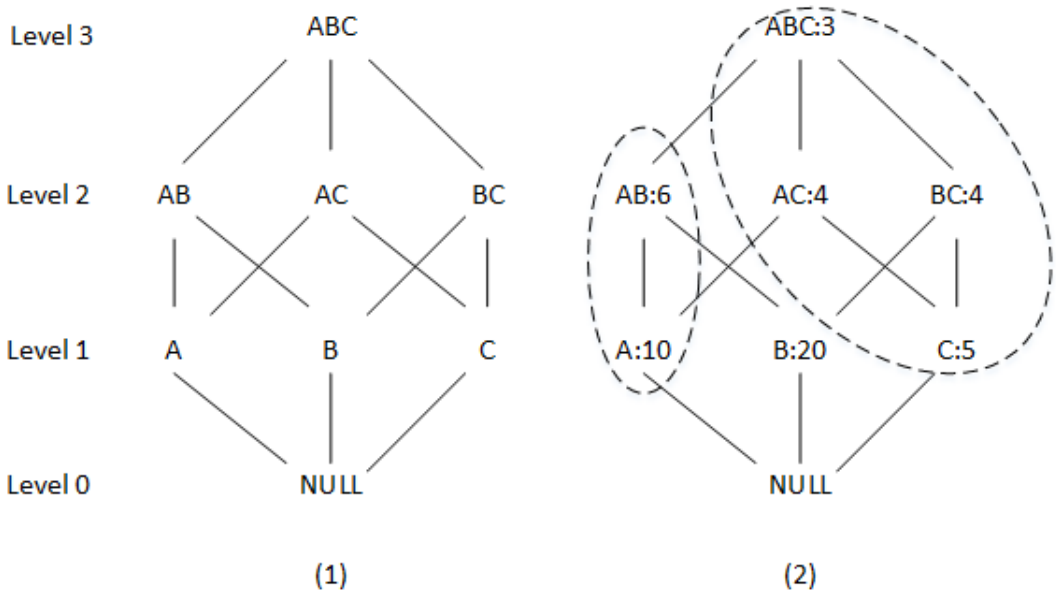

Figure 2.7: Iceberg Lattices, (1)3-D Lattice; (2) 3-D Lattice with Association Rules Mined

Figure 2.7 (1) shows a simple lattice structure. Each node, except the one at level 0, can represent a ngram (or n-dimension sequence). In Figure 2.7 (2), the number next to each n-gram is the frequency of the n-gram in database. For example, $ABC : 3$ means that ABC occurs 3 times in the database. By searching among the nodes in a lattice, we can group the ngrams that co-occur together and satisfy the minimum support and minimum confidence. For example, two groups have been identified in Figure 2.7 (2) with the minimum support of (3/total-transactions) and the minimum confidence of 60%.

2.4.5 Applications of Ngram Grouping

In information retrieval, especially web search, queries tend to be short, usually made of two to three words. As a result, it creates an extremely unbalanced match between short queries and long documents.

Query expansion tries to expand a query by adding related terms in an implicit way so that users can focus on the original query. It has been shown to be helpful in improving both recall and precision of an information retrieval system [37]. The method of grouping ngrams described in the previous subsection can be potentially used for query expansion since it allows us group related ngrams into clusters.

2.5 Information Retrieval

2.5.1 Definition of Information Retrieval

Salton gave the following definition of *Information Retrieval* in his textbook [55] published in 1968:

"Information retrieval is a field concerned with the structure, analysis, organization, storage, searching, and retrieval of information. "

This definition is a classical and accurate description of information retrieval. Although the field has advanced so much over the past decades, it is still appropriate [3]. This idea was popularized by Vannevar Bush in 1945 in his article "As We May Think" as a way to access a large amount of information [57]. The primary applications of information retrieval focus on textual information, such as Web pages, news articles, books, and scholar papers, but with the development of technology, "information" can now refer to a wide range of media such as images, audios and videos. The search engine is a practical application of information retrieval.

Research in information retrieval started in the 60's and it remains active. In 1968, Gerard Salton developed an information retrieval system called SMART(System for the Mechanical Analysis and Retrieval of Text) first at Harvard University and later at Cornell University [12]. Carnfield Test, developed by Cleverdon, provides an evaluation methodology for information retrieval systems. Subsequently, many models of retrieval systems were developed [57]. With the availability of large document collections by Text Retrieval Conference(TREC) [5] since 1992, the field of information retrieval can now measure the scalability and applicability of those systems.

The challenge of information retrieval is how to handle the unstructured documents, compared with traditional structured records in databases. Structured records usually have a specified format. People can easily find the information from the related attributes. For example, a relational database about students may have a few attributes, such as name, age, address, enrolment year, and phone number. Such information is easily searchable. Unlike the structured records, information retrieval systems need

[5]http://trec.nist.gov/

to identify important features from text in the form of words, phrases, sentences and documents. For example, given a stack of personal statements for new graduate students, a staff member may have to check them one by one in order to find one for a specific student. This kind of information is usually easy for people to process, but for a large collection of documents, it is too time consuming and not feasible. As a result, we want to design algorithms that use features to separate documents, analyze them, store them, and search them, especially when the amount of information is out of the range for people to process.

2.5.2 Key Concepts for Information Retrieval

Corpus is a collection of documents, from which users can find the relevant information via information retrieval systems.

Terms are usually words or phrases in the corpus.

Index (Inverted Index) contains terms assigned to documents so that we can find the relevant documents more efficiently. More specifically, it is called the inverted index and is made of two parts: Dictionary is a list of terms in the alphabetical order; and Posting is a list of entries with document IDs where the corresponding terms occur along with the frequencies. For example, in Figure 2.8, "word1" occurs in three documents {1, 3, 4} and the frequencies of this term in these documents are 3, 3, 2, respectively.

Query represents the information need, usually consisting of one or a few terms and used by users to find relevant documents in a corpus.

Ad Hoc Retrieval is a standard task for information retrieval where a query is a set of terms rather than a structured expression of terms with Boolean operators.

Effectiveness is traditionally measured by precision and recall, but recent infor-

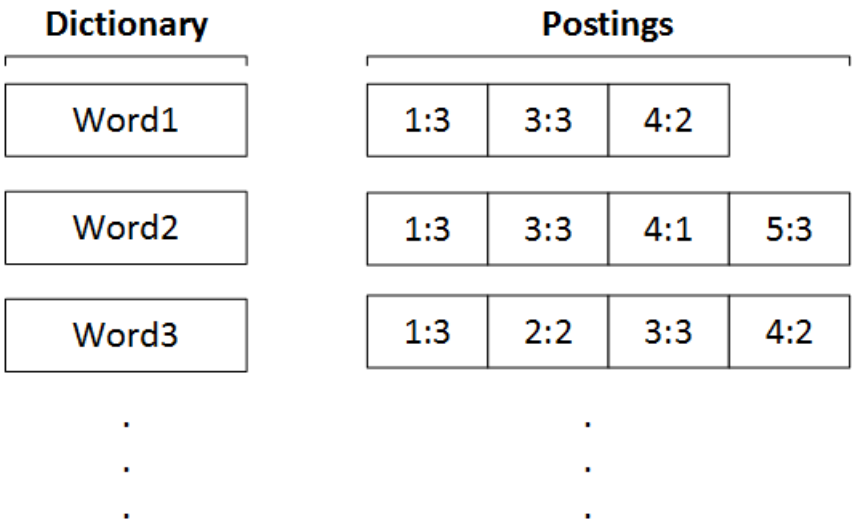

Figure 2.8: Inverted Index Structure

mation retrieval systems tend to use other measures, such as F-measure, MAP(mean average precision) and top5 (or top10) precision.

Precision is the fraction of the retrieved relevant documents over all retrieved documents.

Recall is the fraction of the retrieved relevant documents over all relevant documents in a corpus.

MAP (Mean Average Precision) is a composite measure for ad hoc information retrieval. It sums each query's average precision and averages the sum by the number of queries. **Average Precision (AP)** for a query sums up precisions when a new relevant document is retrieved, which is later divided by the relevant-document-retrieval time (not exactly the same as the number of relevant documents because cognitively, the system can rarely find all of the relevant documents for a query).

$$MAP = \frac{1}{N}\sum_{j=1}^{N} AP(q_j) = \frac{1}{N}\sum_{j=1}^{N}\left(\frac{1}{Q_j}\sum_{i=1}^{Q_j} P(rel = i)\right) \tag{2.9}$$

where Q_j is the number of retrieved relevant documents for query j; N is the number of queries, and $P(rel = i)$ is the precision at ith relevant document.

In this thesis, we will apply our keyphrase extraction and grouping methods to the Ad hoc Retrieval of the standard document collections from TREC [6].

2.5.3 Major Models for Information Retrieval

Boolean Model

A Boolean model is also called the exact-match retrieval because only documents that exactly match the query terms will be retrieved. In Boolean model, the relevant documents are not ranked because the matching of terms is binary (either true or false).

In addition to the three basic Boolean operators: AND, OR, and NOT, some Boolean retrieval systems use additional criteria, such as same sentence, same paragraph, and publish date. For example, a query "york AND U.S. AND NOT business" will retrieve the documents that contain "york" and "U.S.", but not "business" in them.

Advantages of Boolean retrieval are predictable and easy understandable results, efficient processing, and easy addition of new features. However, there are some disadvantages. The AND operator often results in high precision but low recall because the system only retrieves relevant documents where all of query terms are matched. The OR operator can result in high recall but low precision because the system will retrieve all documents with either term1 or term2. As a result, users' knowledge about relevant data is critical in formulating complex queries in order to get better results. Another drawback about such a system is the lack of ranking, which treats all retrieved documents the same, making it difficult to find really important documents for a given

[6] www.trec.nist.gov

query.

Vector Space Model

In the Vector Space Model, both documents and queries are represented by vectors of weights, each of which is associated with a term, corresponding to one dimension in a high dimensional space.

Let document d_i and query q be the vectors of weights: $d_i = (d_{i1}, d_{i2}, ..., d_{ij})$ and $q = (q_1, q_2, ..., q_j)$, where j is the j_{th} term in a dictionary. A corpus of n documents can be represented as a matrix of weights:

$$\begin{array}{cccccc} & Term_1 & Term_2 & \dots & Term_j \\ Document_1 & d_{11} & d_{12} & \dots & d_{1j} \\ Document_2 & d_{21} & d_{22} & \dots & d_{2j} \\ \vdots & \vdots & & & \\ Document_n & d_{n1} & d_{n2} & \dots & d_{nj} \end{array}$$

where each row is a document vector and each column corresponds to the weights assigned to a particular term in all documents. A more intuitive illustration of the vectors is shown in Figure 2.9.

In practical applications, the matrix of weights will be very sparse since many terms in a dictionary will not appear in a document, resulting in zeros for the weights. Thus, the inverted index structure shown in Figure 2.8 help optimize the memory by not storing zero values in the structure.

The term weights are commonly computed by $tf \times idf$, where tf is the frequency of a term in a document and idf is the inverse document frequency for a term in the corpus:

$$idf_t = log\frac{N}{df_t} \tag{2.10}$$

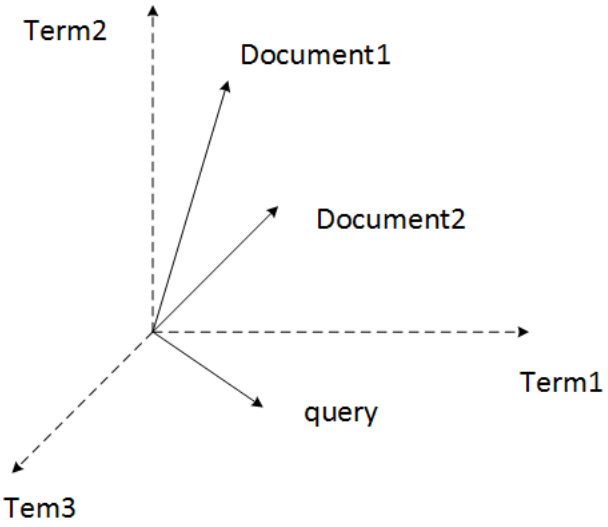

Figure 2.9: A Diagram Illustrating the Vector Space Model

where N represents the number of documents in a corpus and df_t is the number of documents where term t occurs.

From Figure 2.9, we see that documents and queries co-exist in the $j - dimension$ space. Finding the nearest document or documents nearer to a query is the goal of search in the vector space model. Among the measures available, *Cosine* similarity, which measures the cosine between two vectors, is commonly used since it does not require all the dimensions to be independent. For document d_i and query q, the Cosine Similarity between the two is defined as follows:

$$Cosine(d_i, q) = \frac{d_i \cdot q}{\|d_i\| \, \|q\|} = \frac{\sum_{j=1}^{t} d_{ij} \cdot q_j}{\sqrt{\sum_{j=1}^{t} d_{ij}^2 \cdot \sum_{j=1}^{t} q_j^2}} \tag{2.11}$$

Vector space model has both advantages and disadvantages. It has a simple way of measuring relevance based on linear algebra and using it to rank the retrieved results. It also allows the exact match as required by the Boolean model. However, the vector

space model essentially assumes that terms are independent with each other, which in fact is not true. For example, it can have poor similarity values for long documents because a long document has large dimensionality but a small dot product value when compared with a short query. In other words, this model assumes that a query and a document can be treated the same but in fact they are not. In addition, the positions of a term in documents have no weight at all, which does not differentiate the retrievals on specific structures, e.g. standardized scientific articles. Finally, although the computational framework is intuitive, the weighting does not justify how it is related to the relevance ranking formally.

Probabilistic Models

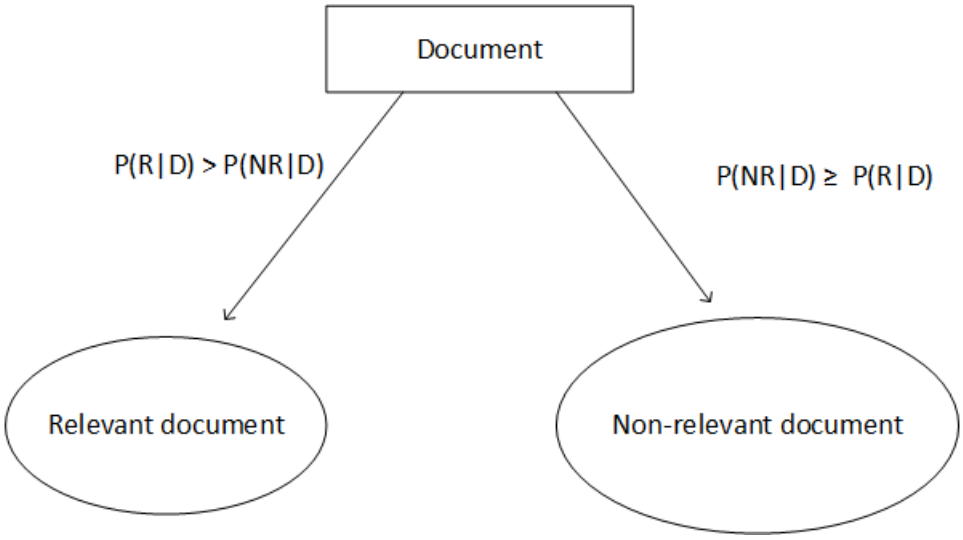

Figure 2.10: Basic Probabilistic Model

A basic probabilistic information retrieval model treats the retrieval process as a classification problem. All documents are divided into two classes: relevant documents (R) and non-relevant class give D, documents (NR), for a given query. The relevant documents will be returned as the search results. Given the probabilities of relevant and non-relevant classes for a document, the class with a higher probability will be assigned

to the document. Let $P(R|D)$ represent the conditional probability of the relevant class given document D, and $P(NR|D)$, the conditional probability of the non-relevant class given D, the documents that satisfy the condition $P(R|D) > P(NR|D)$ will be treated as relevant documents; otherwise, non-relevant documents, as shown in Figure 2.10.

The probabilities of $P(R|D)$ and $P(NR|D)$ are computed by the $Bayes' Rule$:

$$P(R|D) = \frac{P(D|R)P(R)}{P(D)} \tag{2.12}$$

$$P(NR|D) = \frac{P(D|NR)P(NR)}{P(D)} \tag{2.13}$$

where $P(D)$ is a normalizing constant, and $P(R)$ and $P(NR)$ are the prior probabilities of how likely a document is relevant and non-relevant with the sum of $P(R)$ and $P(NR)$ equals 1. If $P(R|D) > P(NR|D)$, then based on 2.12 and 2.13 shown above, we have:

$$\frac{P(D|R)}{P(D|NR)} > \frac{P(NR)}{P(R)} \tag{2.14}$$

indicating that the document will be assigned to the relevant class. For information retrieval, ranking documents is preferred over a true or false decision, and in this case, we can use $P(D|R)/P(D|NR)$ to rank documents.

Support that p_i is the probability of term i occurring in a relevant document set, and s_i is the probability of term i occurring in a non-relevant document set. By the independence assumption of Naive Bayes' classifier that all terms are independent of each other, we have:

$$P(D|R) = \prod_{i=1}^{t} P(d_i|R) = \prod_{i:d_i=1} p_i \cdot \prod_{i:d_i=0} (1 - p_i) \tag{2.15}$$

where $d_i = 1$ means that the term belongs to a relevant document and $d_i = 0$ means

that the term does not belong to the relevant document. Similar expansion applies to $P(D|NR)$. Applying these expansions to (2.14), we get:

$$\frac{P(D|R)}{P(D|NR)} = \prod_{i:d_i=1} \frac{p_i}{s_i} \cdot \prod_{i:d_i=0} \frac{1-p_i}{1-s_i} \tag{2.16}$$

After a series of mathematical transformations, it becomes:

$$\frac{P(D|R)}{P(D|NR)} = \prod_{i:d_i=1} \frac{p_i}{s_i} \cdot (\prod_{i:d_i=1} \frac{1-s_i}{1-p_i} \cdot \prod_{i:d_i=1} \frac{1-p_i}{1-s_i}) \cdot \prod_{i:d_i=0} \frac{1-p_i}{1-s_i} = \prod_{i:d_i=1} \frac{p_i(1-s_i)}{s_i(1-p_i)} \cdot \prod_{i} \frac{1-p_i}{1-s_i} \tag{2.17}$$

Because the second product is the same for all terms, it can be ignored in the ranking process. In addition, since the logarithm preserves both the precision and ranking, we can introduce it into the formula below:

$$argmax_i \frac{P(D|R)}{P(D|NR)} = argmax_i \sum_{i:d_i=1} log \frac{p_i(1-s_i)}{s_i(1-p_i)} \tag{2.18}$$

	$d_i = 1$	$d_i = 0$	Total
Relevant	r_i	$R - r_i$	R
Non-relevant	$n_i - r_i$	$NR - n_i + r_i$	NR
Total	n_i	$N - n_i$	N

Table 2.2: Term Occurrences for a Query

What is left is to figure out how to calculate the values of p_i and s_i. From Figure 2.11, we can find the values of p_i and s_i based on known information. Figure 2.11 (1) shows that there are relevant documents(R) and non-relevant documents(NR) in a document collection for term i. Figure 2.11 (2) shows that the number of documents where term i occurs in N, n_i, and number of relevant documents for i, r_i. Figure 2.11 (3) is a combination of (1) and (2). Table 2.2 presents the term occurrences for a particular query.

From the above information, we can get $p_i = r_i/R$ and $s_i = (n_i - r_i)/NR$. To avoid

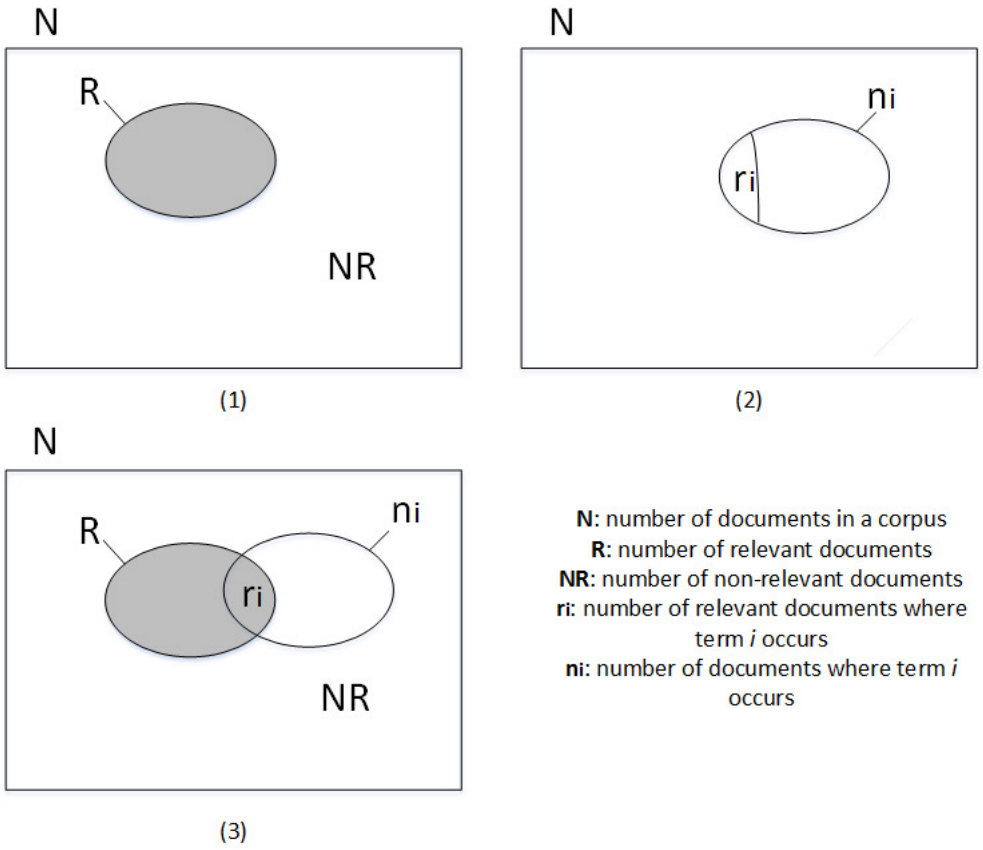

Figure 2.11: Relations of Term i in a Document Collection for a Particular Query

the cases where any numerator or denominator equals to zero, both p_i and s_i are added with 0.5, implying that there might be no relevant documents in a corpus. Then, we get:

$$argmax_i \frac{P(D|R)}{P(D|NR)} = argmax_i \sum_{i:d_i=q_i=1} log \frac{(r_i + 0.5) \cdot (NR - n_i + r_i + 0.5)}{(n_i - r_i + 0.5) \cdot (R - r_i + 0.5)} \quad (2.19)$$

According to the literature, the performance of the basic probabilistic model is often not good. BM25 combines the main part of the basic probabilistic model with additional parameters and has been shown to be an effective ranking algorithm for

information retrieval. The common form of BM25 is given as follows:

$$\sum_{i:d_i=q_i=1} log\frac{(r_i+0.5)\cdot(NR-n_i+r_i+0.5)}{(n_i-r_i+0.5)\cdot(R-r_i+0.5)}\cdot\frac{(k_1+1)f_i}{K+f_i}\cdot\frac{(k2+1)qf_i}{k2+qf_i} \tag{2.20}$$

where f_i is the frequency of term i in a document; qf_i is the frequency of term i in a query; k_1, k_2, and K are the parameters to be assigned empirically since these parameters are dependent on a document collection. Thus, recursive training is often needed for this extended model for information retrieval.

Language Models

Language models are recent but important ways to build information retrieval systems. Ponte and Croft [11] presented a statistical approach in 1998 for information retrieval based on a generative language model. This approach has a big impact to the research field due to the fact that it performs well empirically compared with other retrieval methods.

A statistical language model assigns probabilities to word sequences. The probabilities can be estimated from a corpus of documents so that we can compute the probabilities for sequences of any length. For information retrieval, we estimate language models for each document, and rank the documents based on the likelihoods of a query to the documents. Such a language model for information retrieval is also called the query likelihood model [52].

Let us use D to represent a document and Q, a query, and assume that all document are relevant to the query, the idea of language modelling based on the Bayes' formula is given as follows,

$$p(D|Q)=\frac{p(Q|D)\cdot p(D)}{p(Q)} \tag{2.21}$$

Since $p(Q)$ is the same for all documents, the formula can be simplified to:

$$p(D|Q) \propto p(Q|D) \cdot p(D) \tag{2.22}$$

Here, the first item $p(Q|D)$ is called query likelihood, which captures how well a document matches a particular query and the second item $p(D)$ measures the probability for the document. In most cases, $p(D)$ is assumed to be uniform and is the same for all documents, as is the case in Ponte and Croft [52], Song and Croft [58], and Zhai and Lafferty [78, 79].

The computation of query likelihood, $p(Q|D)$, depends on the language models of the documents. At the unigram level, we have a simpler formula in the following:

$$p(Q|D) = \prod_{q_i \in Q} p(q_i|D) = \prod_{q_i \in Q} \frac{f_{q_i,D}}{|D|} \tag{2.23}$$

where $f_{q_i,D}$ is the frequency of word q_i in document D, and $|D|$ is the number of words in D.

However, this model suffers from the sparse data problem. If there is any word missing in a document, the score for the query will be zero, which is improper. To overcome this problem, smoothing is applied so that the probabilities of the unseen words can be estimated based on the frequencies of the missing words in the whole corpus. Afterwards, $p(q_i|D)$ is replaced by $(1-\alpha)p(q_i|D) + \alpha p(q_i|C)$, where α is usually set to a constant between 0 and 1 for assigning probabilities to unseen words and C is the total number of word occurrences in the corpus. Consequently, $p(Q|D)$ becomes:

$$p(Q|D) = \prod_{q_i \in Q} p(q_i|D) = \prod_{q_i \in Q} ((1-\alpha)\frac{f_{q_i,D}}{|D|} + \alpha\frac{c_{q_i}}{|C|}) \tag{2.24}$$

where c_{q_i} is the number of word q_i appearing in a corpus.

To preserve precisions for the above formula, logarithms are often used as follows:

$$logp(Q|D) = log \prod_{q_i \in Q} p(q_i|D) = \sum_{q_i \in Q} log((1-\alpha)\frac{f_{q_i,D}}{|D|} + \alpha\frac{c_{q_i}}{|C|}) \tag{2.25}$$

Existing experiments show that language modelling is at least as effective as the BM25 ranking model. Extensions to the language model, such as bigrams, trigrams and relevance feedback, can be found in [52, 58, 78, 11].

Chapter 3

Proposed Solution

In this chapter, we describe in detail our proposed solution for keyphrase extraction and grouping along with its application to information retrieval. The content is organized is as follows:

(1) **Generating keyphrase candidates:** finding all frequent ngrams based on document frequencies and treating them as keyphrase candidates.

(2) **Selecting keyphrases:** selecting high-quality keyphrases from the keyphrase candidates using the LocalMaxs method.

(3) **Grouping related keyphrases:** grouping keyphrases to the related synonym groups based on association rules and co-occurrences.

(4) **Data preprocessing**: using the common steps for data preprocessing, including scanning, stop word removal, and stemming.

(5) **Application to information retrieval**: applying keyphrases and their related groups to information retrieval and measuring the effectiveness of our system on a standard dataset for information retrieval.

3.1 Generating Keyphrase Candidates

In Chapter 2, we reviewed the BIDE algorithm for mining frequent closed sequences with gaps within them. It solves the problem of large memory required by prior frequent sequence mining algorithms by introducing *Pseudo-Projection Databases* that use positions and offsets of sequences in a database rather than copying the original database. It also prunes the search space by *BackScan* that stops growing a sequence right after it decides that there is no chance for the present sequence to be a closed one. Finally, another optimization step *ScanSkip* is used to increase the scanning efficiency as described in Chapter 2.

However, general closed sequences with gaps are not suitable for natural language documents since the gaps between words are not very common within phrases. Some verb phrases may have gaps, such as "pick up" for "pick (it) up" but our experiments mainly focus on noun phrases, which rarely have gaps within them. As a result, we limit ourselves to phrases with consecutive words, or ngrams in our experiments in order to increase the efficiency of our systems.

In addition, although closed sequences help reduce the redundancy in text representation, we still need to extract all frequent sequences for information retrieval. This is because a user may use different frequent phrases in the queries and if we only store frequent closed sequences in the index for a document collection, we will not be able to match them with some of the frequent sequences in a query, resulting in low recall performance.

Consequently, we need to consider the extraction of frequent sequences so that we can maximize the possible matches at the phrase level for information retrieval. We could use a direct method to extract frequent sequences. However, there are advantages of using the BIDE-like algorithm as introduced by Wang and Han [71] since it saves

both memory and time in generating frequent sequences. As a result, we choose to adapt the BIDE algorithm for extracting frequent ngrams by reducing the gaps to zero and simplifying the algorithm for improved efficiency.

Frequent ngrams are based on document frequencies, measuring word sequences that occur frequently in a document collection. However, they are not strong enough to be keyphrases, such as "able to find", "fast enough", and "where is it". As a result, we need to further differentiate frequent ngrams by selecting strong keyphrases, which will be discussed in the next section.

3.2 Selecting Keyphrases

In this section, we use the *LocalMaxs* method to select keyphrases made of words strongly associated with each other. Compared with other methods, *LocalMaxs* requires neither linguistic knowledge nor empirically obtained thresholds to select strongly associated multiword terms. Huo [32] applied LocalMaxs for web page summarization so that keyphrases can be extracted and used to produce short summaries for web pages which may be poorly structured with fragmented textual information.

LocalMaxs makes two assumptions: there is a kind of "glue" that sticks these words together in a multi-word term; and the glue is strong for those more meaningful terms, such as "Microsoft Windows" and "hot dog", but weak for those irregular terms, such as "that measures" and "of each". More specifically, LocalMaxs chooses multi-word terms whose glue value *g(.)* satisfy:

$$length(W) = 2 : g(W) > g(Y)$$

$$lenght(W) > 2 : g(X) \leq g(W) \text{ and } g(W) > g(Y)$$

where W is an n-gram phrase $\{w_1, w_2, \ldots, w_n\}$; Y is an (n+1)_gram with another word

w_{n+1} added within, before, or after W to get $\{w_1, w_2, \ldots, w_{n+1}, \ldots, w_n\}$, $\{w_{n+1}, w_1, w_2, \ldots, w_n\}$ or $\{w_1, w_2, \ldots, w_n, w_{n+1}\}$; and X is an (n-1)-gram with one word deleted from W.

In order to define the glue value *g(.)*, Silva, et al. [14] propose one association measure called Symmetric Conditional Probability (SCP), which helps extract keyphrases when used with LocalMaxs method.

A conditional probability is the probability of an event that will occur given that another event has occurred, denoted as $P(x|y)$ and defined as:

$$P(x|y) = \frac{P(x,y)}{P(y)} \tag{3.1}$$

where $p(y)$ and $p(x,y)$ are the probabilities of event y and the joint-event (x,y).

For a bigram, SCP is simply defined as:

$$SCP(x,y) = p(x|y) \cdot p(y|x) = \frac{p(x,y)^2}{p(x) \cdot p(y)} \tag{3.2}$$

To generalize the SCP above to multi-word terms, *Fair Dispersion Point Normalization* is used as the denominator of equation (3.2). Each ngram is seen as a set of "pseudo-bigrams" that break an ngram at different points such as $[w_1 \ldots w_i]$ and $[w_{i+1} \ldots w_n]$. As a result, the Fair Dispersion Point Normalization can be defined as follows:

$$Avp = \frac{1}{n-1} \sum_{i=1}^{i=n-1} p(w_1 \ldots w_i) \cdot p(w_{i+1} \ldots w_n) \tag{3.3}$$

By substituting Avp for the denominator $p(x) \cdot p(y)$ in (3.2), we get the generalized SCP for ngrams with $n \geq 2$:

$$SCP\_f(w_1 \ldots w_n) = \frac{p(w_1 \ldots w_n)^2}{Avp} \tag{3.4}$$

In Huo [32], he also proposes another association measure for extracting phrases with LocalMaxs. The measure is a normalized sequence probability based on the fact

that the probabilities of longer ngrams are always lower than those of their sub-ngrams. As a result, longer ngrams will be at disadvantage when used with $LocalMaxs$. In order to compare ngrams of different lengths fairly, the sequence probability is normalized by its n_{th} root to get the average word-level probability:

$$seq\_p(w_1 \dots w_n) = \sqrt[n]{p(w_1 \dots w_n)} \tag{3.5}$$

For distinguishing these two measures, the method that uses SCP_f is called *LocalMaxs1* and the method that uses $seq\_p(.)$, *LocalMaxs2* in our experiments.

3.3 Grouping Related Keyphrases

3.3.1 Grouping Based on Iceberg Lattice

An Iceberg Lattice is a useful structure that captures the "includes" relationship among the itemsets or sequences in databases. It provides an intuitive visualization for concept analysis in the form of a hierarchical representation for frequent itemsets or sequences.

To group related keyphrases or synonyms together, we construct an iceberg lattice for keyphrases extracted by LocalMaxs. As shown in chapter 4, LocalMaxs can eliminate a large number of frequent ngrams because their glue values are locally weaker. On the other hand, these eliminated ngrams may still appear in documents and queries. If we do not include them in the index, we will fail to match them for information retrieval. However, by grouping these ngrams with the keyphrases selected by LocalMaxs, we can treat the ngrams in each group as synonyms and thus increase the recall performance.

The construction of an iceberg lattice is divided into two steps: (1) bottom up to build the lattice; (2) topdown to compute synonym groups. As illustrated in Figure 3.1, level 1 links NULL(empty phrase) to all unigrams; level 2 links all unigrams to

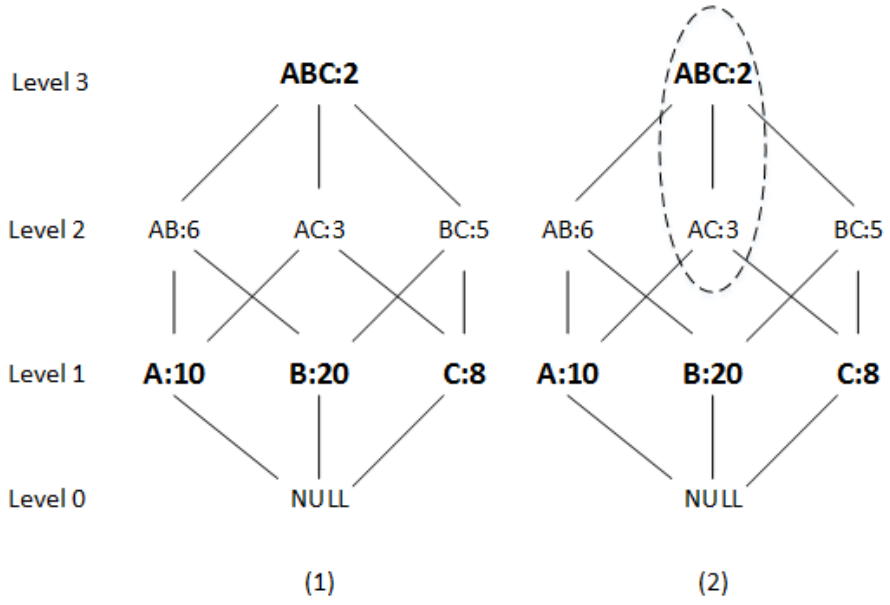

Figure 3.1: Iceberg Lattice, (1)bottom up construction; (2)Topdown search with minimum confidence of 0.6

bigrams, and so on. The topdown step searches the sub-ngrams in the lattice that satisfy the minimum confidence in order to form synonym groups. For example, in Figure 3.1 (2), we assume that keyphrases in boldface are found by LocalMaxs and unigrams are always selected for completeness and the minimum confidence is 0.6. When ABC is processed, we will search its sub-sequences and check their confidence values with ABC. Since AC is the only ngram that satisfies the minimum confidence, it will be put into the same group with ABC. Then, AC will be checked against its sub-sequences to find if more sub-sequences can satisfy the minimum confidence, and so on.

3.3.2 Grouping Based on Co-occurrence Graphs

Alternatively, we can group keyphrases based on their co-occurrences in the dataset. According to Mihalcea, a link in a co-occurrence graph represents the association between two syntactic elements [51]. Also, in Mihalcea and Tarau's TextRank [50], they

declared that "*Any relation that can be defined between two lexical units is a potentially useful connection (edge) that can be added between two such vertices.*". The proposed model, TextRank, is based on the co-occurrence relation between two words with its range restricted to a window of maximum N words within a sentence. The N is a user-defined parameter ranging from 2 to 10 in their experiment.

In other words, if two words co-occur with a sufficient number of times under a parameter value, they will be linked in the graph. Thus, we can also use the "co-occurrence graph" to group phrases that co-occur in a sufficient number documents. Because the TextRank's maximum of N words is very strict, we define a more relaxed condition for the co-occurrence of two terms by expanding the maximum of N words to a document. In other words, any two terms that co-occur in a sufficient number of documents will be connected and put into the same group. Given two words(or phrases) A and B, $F(.)$ as the document frequency of a word or keyphrase, $U(A, B)$ as the number of documents that contain both A and B, R as the user-defined ratio for grouping, A and B will be put into one group iff

$$\frac{min(F(A), F(B))}{F(A) + F(B) - U(A, B)} \geq R \tag{3.6}$$

For the example in Figure 3.2, we construct a lattice based on the LATIMES document set. Each node is a word or keyphrase along its document frequency. The nodes with circles are the keyphrases selected by LocalMaxs, and the ones without circles are not selected. When we use the lattice to do topdown search, the "blood's ability carry" will be added to the keyphrase list based on the confidence of 0.9 but "blood's ability" will not be added.

The advantage of a co-occurrence graph is that by finding the connection between two terms, they become the "recommenders" to each other and thus can help re-

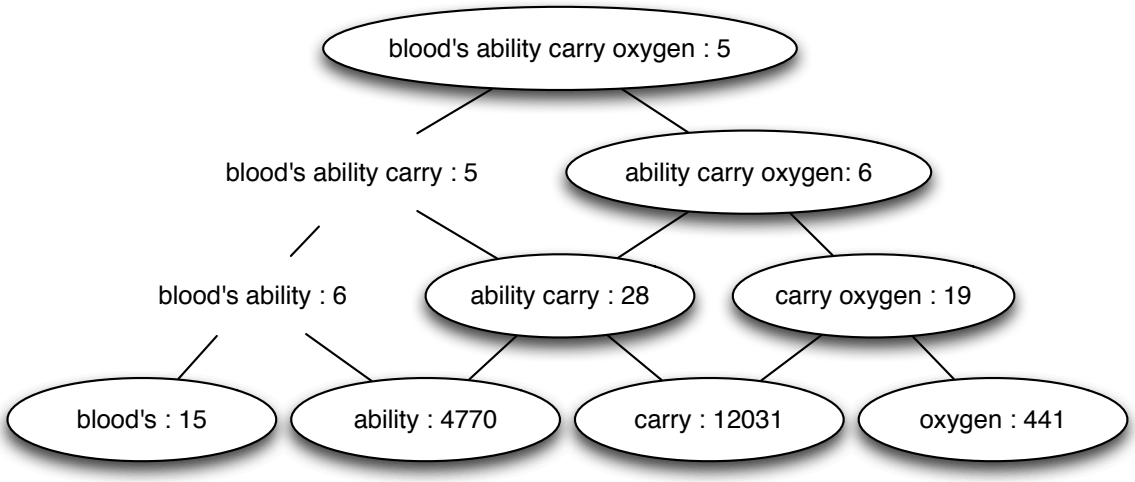

Figure 3.2: Lattice Example with Confidence of 0.9

searchers to construct a meaningful graph and improve precision and recall with co-occurrence information [73].

3.4 Data Preprocessing

Text mining is to obtain terms and rules hidden in the textual data. As the first step, we need to clean and split text into words or tokens [72]. Here, "clean" means the extraction of text that has content and removal of tags and auxiliary information in a web page or PDF document such as images and graphs. After that, we split text into tokens. In our experiments, we use scanning, stop word removal, and stemming for data processing.

3.4.1 Scanning

Scanning does most of the work for data preprocessing. It splits text into tokens and remove irrelevant tokens. In our experiments, we also separate documents into sentences in order to extract keyphrases later on.

- Punctuation marks, such as brackets and pure numbers (like 100 but not 100th) are removed except for period, question mark, and exclamation mark that indicate the end of a sentence;
- Hyphenated words are split when both sides of the hyphen are words with more than 2 letters. For example, "state-of-art", "co-founder" are kept but "pay-off" will be split into "pay" and "off";
- Apostrophes are treated similarly as hyphenated words;
- Non-token characters, such as non English letters, are removed;
- Abbreviations with dots within them are kept, such as Ph.D and U.S.

3.4.2 Stop Word Removal

Stop word list includes the words that occur most frequently in a specific language (such as "to" "the" "of" "and" etc. in English). Stop words help to form phrases and sentences, but they do not contain much content. As a result, they often need to be removed for information retrieval. Furthermore, stop words make up a large fraction of the text and by eliminating them, we can not only save memory, but also speed up processing and improve retrieval performance [20].

As noted in [20], stop words are domain dependent. For example, a document collection about computer science will probably not treat "computer", "program" "machine" as index terms. Lo, He and Ounis [43] studied how to automatically build a stop word list for an information retrieval system. The approach ranks terms from the corpus by some weight and then chooses rank threshold above which terms are treated as stop words. In our experiments, we simply use a general list of predetermined stop words found in [20].

3.4.3 Stemming

From morphology, we know that some derivationally related words (like democracy, democratic, and democratization) have similar meanings. Treating them as different words not only increase the vocabulary size, but also make it hard to relate them together.*Stemming* is a crude heuristic process to strip the suffixes of derived words and shorten them to word stems so that we can both reduce the vocabulary size and relate different words to the same stem [46].

The Lovins' (1968) and Porter's (1980) stemming algorithms are two widely used stemmers for English, both of which are based on heuristic rules. Porter's algorithm is more effective but Lovins' algorithm is more aggressive. Figure 3.3 shows the differences of these two stemmers for a sample text.

Sample text: however, general closed sequences with gaps are not suitable for natural language documents since the gaps between words are not very common within phrases

Lovins stemmer: howev , gen clos sequ with gap ar not suit for nat langu docu sint the gap between word ar not very common within phrases

Porter stemmer: howev , gener close sequenc with gap are not suitabl for natur languag document sinc the gap between word are not veri common within phrase

Figure 3.3: Comparison of Two Stemming Algorithms

3.5 Application to Information Retrieval

3.5.1 Index Term Selection

In ad hoc information retrieval systems, we have to select keywords and keyphrases (generally referred to as terms) for indexing. Leung [38] suggests that a word/phrase's frequency should not be too high nor too low to represent a document's content. For

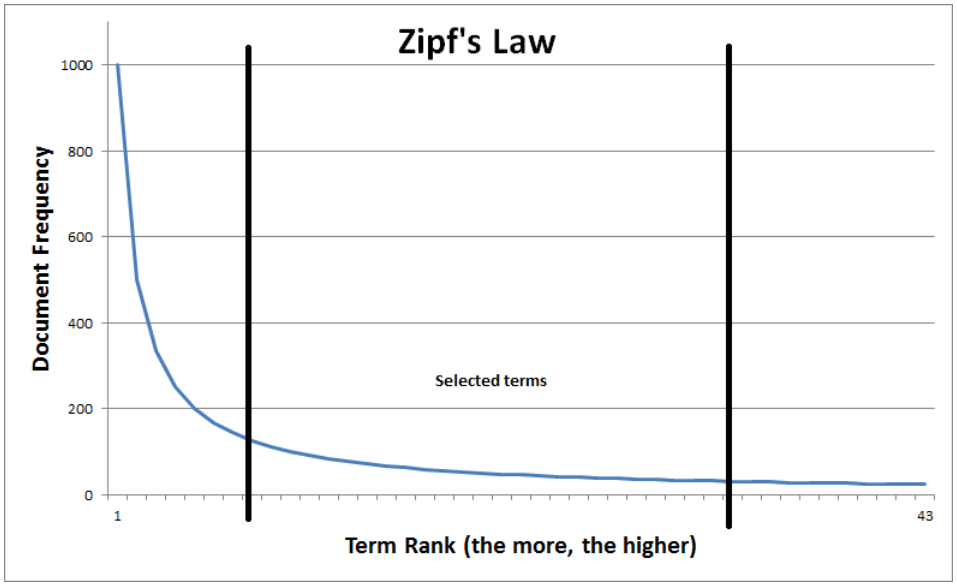

Figure 3.4: Term Distribution in Zipf's Law

example, in a collection of documents about law, the words/phrases "legal", "criminal law", "appeal courts" are very frequent. A word/phrase with a high frequency does not discriminate well one document from another, resulting in high recall but low precision in retrieval performance. On the contrary, a low frequency word/phrase can lead to high precision but low recall. Both will fail to increase the overall performance (such as F-measure) of retrieval systems. Thus, words/phrases with medium frequencies provide good choices for indexing.

The document frequencies of words and phrases follow the *Zipf's Law*, as shown in Figure 3.4. It is an empirical law based on mathematical statistics. The law indicates that given a document collection, the document frequencies of words are inversely proportional to their ranks for the documents. For example, article "the" is the most frequent word in English but its importance for distinguishing documents is almost negligible.

An ad hoc information retrieval system is developed to measure the effectiveness of keyphrase extraction and grouping. When matching keyphrases in documents and

queries, we can do either forward checking or backward checking.

Forward checking finds an n-gram in a sequence and check if it exists in the index in the forward direction. In a sequence, each word is labelled as i_{th} unigram where $0 \leq i \leq sequence\_size$. The procedure of forward checking is given as follows:

- Find the i_{th} unigram in a sequence that belongs to document d;
- Go forward to $(i+n-1)_{th}$ unigram where $i+n \leq$ sequence_size;
- Use i_{th} to $(i+n-1)_{th}$ unigrams to form an n-gram;
- Search the predefined index to check if the ngram exists as an entry;
- If so, increase the count of the ngram for document d.

For example, given the sequence $s = \{w_0, w_1, w_2, ..., w_{n-1}, ..., w_m\}$, if we want to find an ngram from the beginning of s, we get $\{w_0, w_1, w_2, ..., w_{n-1}\}$.

Backward checking Suppose that we have an ngram with $n > 2$. Backward checking goes backward to (n-1)-gram by discarding the last unigram of the given ngram, and check if the (n-1)-gram exists in the index. If yes, d's information will be updated for the (n-1)-gram. For example, if we want to do backward checking for ngram $\{w_0, w_1, w_2, ..., w_{n-1}\}$, the (n-1)-gram is $\{w_0, w_1, w_2, ..., w_{n-2}\}$. Backward checking may be applied recursively until only the first unigram w_0 is left, or terminated early, depending on the situation.

In this thesis, we use five different ways to match keywords and keyphrases.

(1) Unigram match (Baseline)

(2) Greedy match without overlap (GN)

(3) Greedy match with overlap (GO)

(4) Incremental match without overlap(IN)

(5) Incremental match with overlap (IO)

3.5.2 Unigram Model

In the baseline model, each unigram (individual word) is used to represent documents. All of the unigrams with sufficient document frequencies will be kept in the index. The unigram model assumes that each word in a documents set is independent of each other and the match between words is straightforward.

3.5.3 Greedy Match without Overlap (GN)

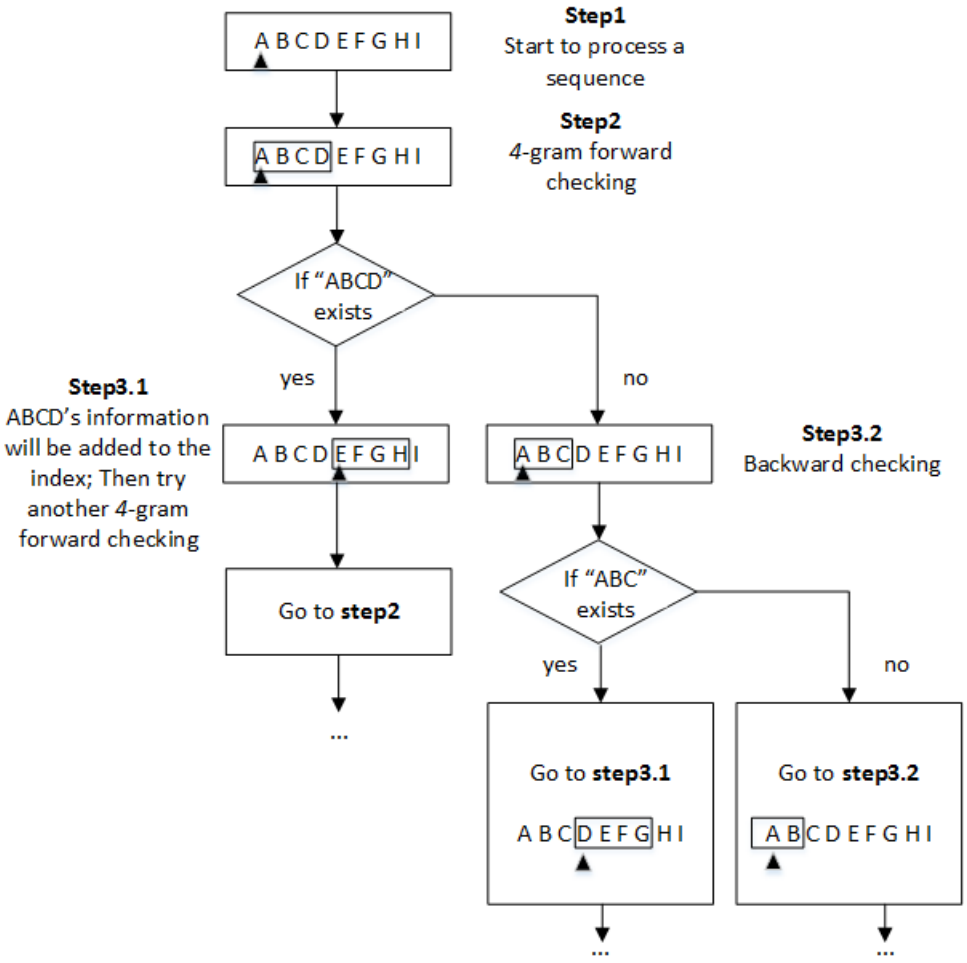

Figure 3.5: Flowchart for Greedy Match without Overlap (GN)

In this method, we first do ngram forward checking: (1) If the ngram exists in the index, the ngram's information will be added to the index. After that, GN will go beyond this ngram and start another ngram forward checking; (2) If the ngram does not exists, GN will do backward checking until it finds a sub-ngram or it cannot go backward anymore. In Figure 3.5, we show the flowchart of GN with the maximum ngram set to 4-gram.

3.5.4 Greedy Match with Overlap (GO)

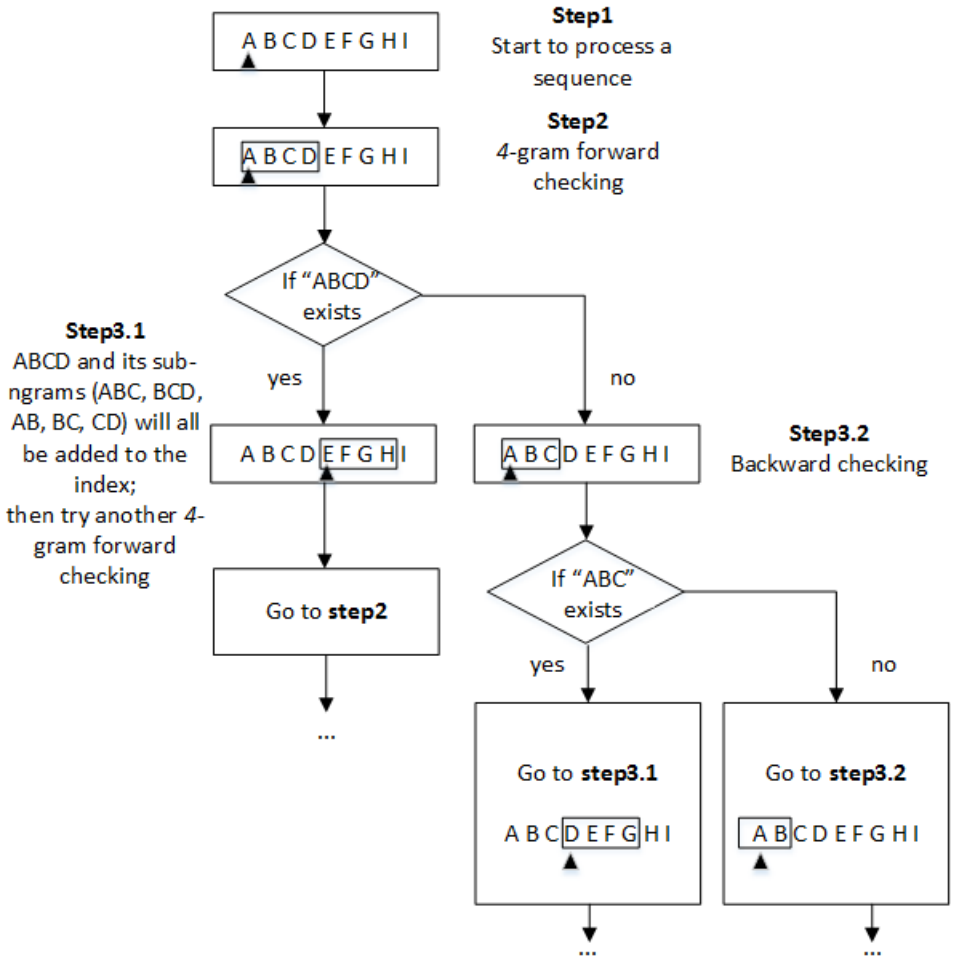

Figure 3.6: Flowchart for Greedy Match with Overlap (GO)

GO is similar to GN except that every time when GO finds a ngram, all the matched sub-ngrams will also be added to the index. In Figure 3.6, we show the flowchart of GO.

3.5.5 Incremental Match without Overlap(IN)

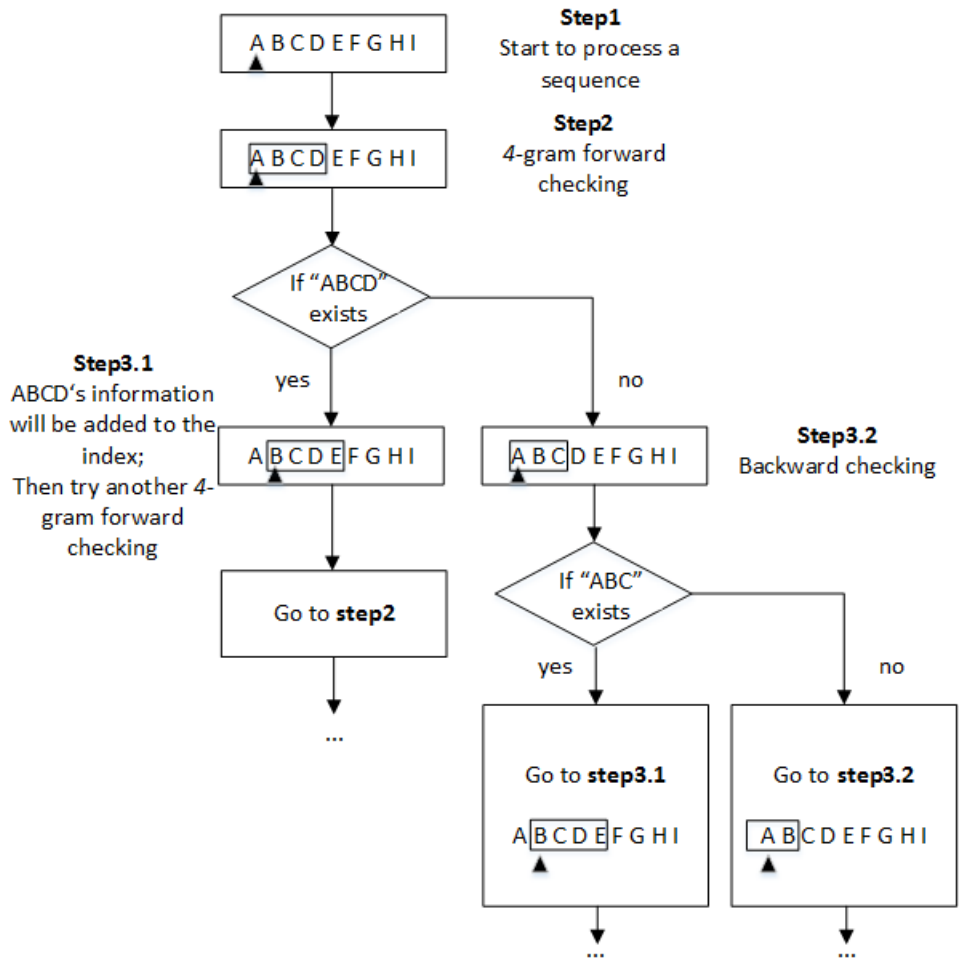

Figure 3.7: Flowchart for Incremental Match without Overlap (IN)

In this method, it first does ngram forward checking starting with the first unigram: (1) If the ngram exists in the index, the information will be added for the ngram and after that, GN will move to the next unigram and continue forward checking; (2) If the ngram does not exists, GN will do backward checking to find matched sub-ngrams or until it cannot do backward checking anymore. In Figure 3.7, we show the flowchart of IN.

3.5.6 Incremental Match with Overlap (IO)

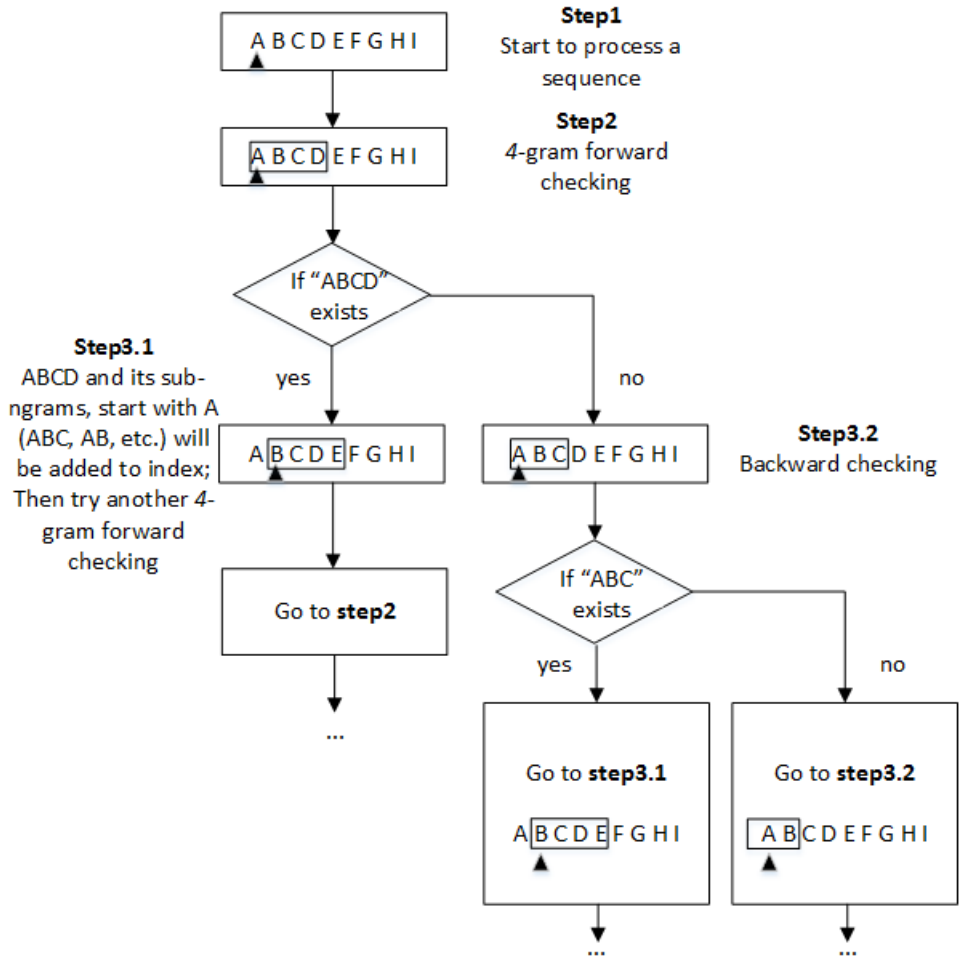

Figure 3.8: Flowchart for Incremental Match with Overlap (IO)

IO is also similar to IN except that every time IO finds a matched ngram, the sub-ngrams that share the same first unigram with the ngram will also be added to the index. For example, if "A B C D" is a 4-gram in the index, the sub-ngrams "A B C" and "A B" are also added to the index. Here, we do not need to add the unigram "A" to the index because all models automatically include all unigrams for information retrieval. In Figure 3.8, we show the flowchart of IO.

3.6 Summary

In this chapter, we described in detail our proposed solution for keyphrase extraction and grouping. In order to test the effectiveness of our approaches, we designed five information retrieval models that match keywords and keyphrases for documents and queries. In next chapter, we will test these models on standard document collections and measure their performance at different stages. Based on these results, we can select the model with best performance to demonstrate the benefits of keyphrase extraction and grouping for information retrieval.

Chapter 4

Experimental Results and Discussions

In this chapter, we first describe the datasets used in our experiments, and then the evaluation methods for both keyphrase processing and information retrieval. After that, we report our experimental results along with discussions.

In our experiments, we combine keywords and keyphrases in the index for information retrieval and use five different methods for matching keywords and keyphrases. Results are also generated for different components of the system in order to measure the impacts for the retrieval performance.

For the datasets, we use TREC4&5 document collections and TREC6, 7, 8 query sets for the evaluations of the information retrieval systems.

4.1 Datasets

4.1.1 TREC Datasets

TREC4 and TREC5 are document collections from Text REtrieval Conferences, sponsored by the National Institute of Standards and Technology (NIST). The documents are tagged using Standard Generalized Markup Language. TREC4 has 3 datasets and

TREC5 has 2, all of which have similar major structures but minor different structures. No spelling corrections or additional formatting are done so as to keep them as close as possible to the original documents. These datasets have been used as the standard evaluation datasets for information retrieval. Detailed statistics for the document sets TREC4&5 are listed in Table 4.1 [68].

Keyphrase grouping is a very time-consuming algorithm because it has the time complexity of $O(n^2)$ and takes 1-2 months to finish all of the TREC data, so we choose only LATIMES dataset in TREC5 for our experiments on information retrieval, which takes 2-3 weeks. For BIDE-like process, it only takes 2-3 days for finishing LATIMES dataset. The reason we chose LATIMES dataset is also because that it is around a quarter of TREC4&5 full datasets and the average document size of LATIMES roughly equals to that in TREC4&5 . Thus, LATIMES is a representative dataset in TREC4&5 datasets.

Table 4.1: Statistics of the Document Sets in TREC4 and TREC5

Dataset	Size(MB)	# docs	Mean# words/doc
TREC 4			
the Financial Times, 1991-1994 (FT)	564	210,258	412.7
Federal Register, 1994 (FR94)	395	55,630	644.7
Congressional Record, 1993 (CR)	235	27,922	1373.5
TREC 5			
Foreign Broadcast Information Service, 1996 (FBIS-1)	470	130,471	543.6
the Los Angeles Times 1989 and 1990 (LATIMES)	475	131,896	526.5

4.1.2 TREC Query Sets

Test query sets are needed for the evaluation of ad hoc information retrieval systems and for TREC4&5 document collections , TREC provides multiple query sets. An example TREC topic is shown in Figure 4.1. There are two main kinds of queries in a TREC topic, a set of keywords (in the $< title >$ field) and natural language sentences (in the $< desc >$ and $< narr >$ fields). The $< title >$ field intends to model web queries with 1 to 3 keywords; and the $< desc >$ and $< narr >$ fields represent natural language queries of different lengths. The shorter the query, the more challenges for the retrieval systems. Statistics for the topic sets of TREC 6, 7, and 8 are listed in Table 4.2 [68], and all of them are used in our experiments.

$< num >$ **Number**: 401
$< title >$ foreign minorities, Germany
$< desc >$ **Description**:
What language and cultural differences impede the integration of foreign minorities in Germany?
$< narr >$ **Narrative**:
A relevant document will focus on the causes of the lack of integration in a significant way; that is, the mere mention of immigration difficulties is not relevant. Documents that discuss immigration problems unrelated to Germany are also not relevant.

Figure 4.1: Sample of TREC Topic

In our experiments, we use both the $< title >$ and $< desc >$ fields as queries. Compared with these two, the $< narr >$ field is much longer, but it is rarely used in reality for information retrieval.

Since topic sets are small, only containing 50 topics each, we follow a process called cross validation to maximize their use. Cross-validation is a statistical method to train and test learning algorithms. A common procedure is to divide the data into a number of smaller subsets called *folds*. One fold is used as the testing set while the rest of the

Table 4.2: Statistics of Topic Sets in Tokens with Stop Words Included

Topics	Min	Max	Mean
TREC-6(301-350)	47	156	88.4
title	1	5	2.7
description	5	62	20.4
narrative	17	142	65.3
TREC-7(351-400)	31	114	57.6
title	1	3	2.5
description	5	34	14.3
narrative	14	92	40.8
TREC-8(401-450)	23	98	51.8
title	1	4	2.5
description	5	32	13.8
narrative	14	75	35.5

folds are used as the training set. The process is repeated for each of the folds and the performance results for all folds are averaged to give an overall measure.

We partition the query sets into three folds, which are illustrated in Table 4.3. Each TREC topic set has 50 queries manually generated for the TREC workshop in a particular year. The content of each topic set is different from the others, and thus, its retrieval performance is different from the TREC results of other years.

Table 4.3: Cross-Validation Folds in Our Experiments

Training set	Testing set
TREC topics 6, 7	8
TREC topics 6, 8	7
TREC topics 7, 8	6

4.2 Performance Metrics

In this section, we explain four performance metrics: Precision(P), Recall(R), F-measure(F), and Mean Average Precision(MAP).

4.2.1 Precision, Recall, and F-measure

Precision and **Recall** together measure the effectiveness of many natural language processing tasks.

For information retrieval, precision represents the proportion of retrieved documents that are relevant, and recall represents the proportion of relevant documents that are retrieved. Table 4.4 shows the results for the retrieval of a simple search. Equations 4.1 and 4.2 shows how precision and recall are calculated based on the relevant and non-relevant documents, and retrieved and non-retrieved documents. For example, suppose that a system only retrieves one document that is relevant to a query. The precision is 1.0. Also, suppose that there are one hundred relevant documents to the query. Then, the recall is only 0.01, implying that the system only finds 1 over 100 relevant documents. As another example, suppose that the system retrieves 1000 documents: 100 are relevant and 900 are irrelevant for the given query. Then, the recall is 1.0 (assuming all 100 relevant documents are retrieved), but precision is only 0.1, which is not good either.

Table 4.4: Results for the Retrieval of a Simple Search

	Labelled as relevant	**Labelled as not relevant**
Retrieved as relevant	**tp** (true positive)	**fp** (false positive)
Retrieved as not relevant	**fn** (false negative)	**tn** (true negative)

$$Precision(P) = \frac{tp}{tp + fp} \tag{4.1}$$

$$Recall(R) = \frac{tp}{tp + fn} \tag{4.2}$$

Clearly, precision and recall are often in conflict: high precision is usually related to low recall, and vice versa. To measure the overall performance of an information

retrieval system, we need a composite measure for precision and recall such as **F-measure**. F-measure is also called the harmonic mean of precision and recall, which is defined as follows:

$$F = \frac{2PR}{P+R} \tag{4.3}$$

4.2.2 Mean Average Precision

Mean Average Precision (MAP) is another composite measure for ad hoc information retrieval that has been used in recent TREC conferences. It sums each query's average precision and averages the sum by the number of queries. **Average Precision (AP)** for a query is summed up when a new relevant document is retrieved and is divided by the number of relevant documents at a retrieval point (which is not always equal to the number of relevant documents because cognitively the system can rarely find all the relevant documents for a query). More specifically, MAP is defined as follows [63]:

$$MAP = \frac{1}{N}\sum_{j=1}^{N} AP(q_j) = \frac{1}{N}\sum_{j=1}^{N}\left(\frac{1}{Q_j}\sum_{i=1}^{Q_j} P(rel=i)\right) \tag{4.4}$$

where Q_j is the number of retrieved relevant documents for query j; N is the number of queries, and $P(rel=i)$ the precision at the i_{th} relevant document.

For the example in Figure 4.2, MAP is calculated as follows:

query #1: $AP = (1.0 + 0.67 + 0.33 + 0.4)/4 = 0.6$

query #2: $AP = (0.5 + 0.67 + 0.33)/3 = 0.5$

$MAP = (AP1 + AP2)/2 = 0.55$

Compared with F-measure, the MAP is more suitable for our experiments due to its ability to measure the ranks of the relevant documents.

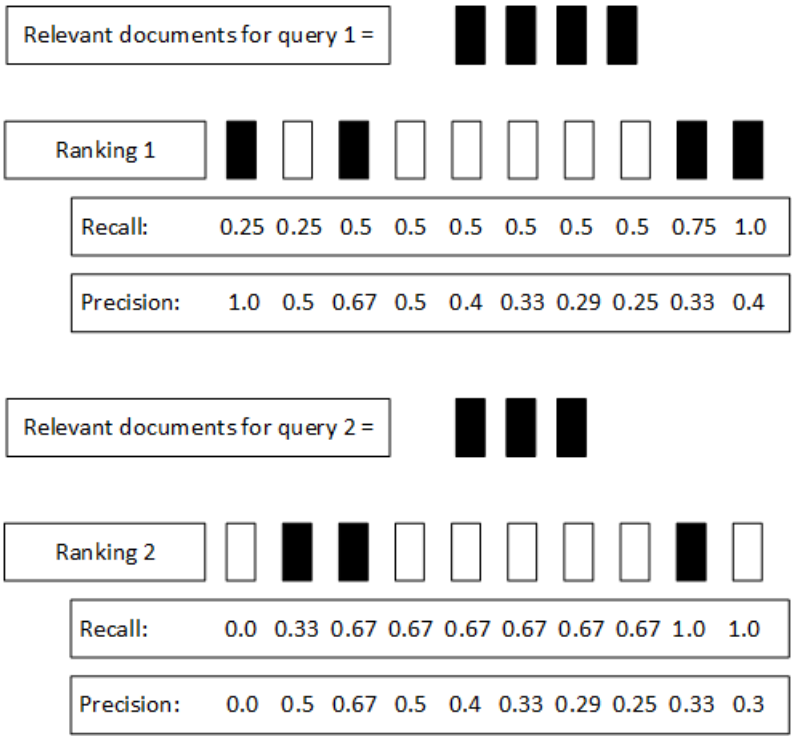

Figure 4.2: Example of Mean Average Precision Calculation

4.2.3 Significance Test

Besides the Mean Average Precision, we also need a significance test to measure if we make real improvements in our experiments. T-test is the most commonly used test for information retrieval and is defined as follows:

$$t = \frac{\overline{B-A}}{\sigma_{B-A}}\sqrt{N} \tag{4.5}$$

where $\overline{B-A}$ is the mean of the differences, σ_{B-A} is the standard deviation of the differences (where $\sigma = \sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2/(N-1)}$), and N is the size of samples.

For the sample data in Table 4.5, we have $\overline{B-A}$=0.214, σ_{B-A}=0.291, N=10, and t=2.33. The P-value is 0.04 for a two-tailed test. Thus, difference is significant for the example at level α=0.05. As a result, we can reject the null hypothesis at level 0.05 and conclude that retrieval algorithm B is better than A. In this thesis, the significance

Table 4.5: Sample Data for Two Retrieval Algorithms over 10 Queries.

Query	A	B	B-A
1	0.25	0.35	0.10
2	0.50	0.75	0.25
3	0.49	0.58	0.09
4	0.39	0.15	-0.24
5	0.52	0.50	-0.02
6	0.20	0.80	0.6
7	0.15	0.85	0.7
8	0.43	0.68	0.25
9	0.75	0.75	0.0
10	0.43	0.84	0.41

test are all based on the two-tailed test.

4.3 Results and Discussions

In this section, we show the results from five systems based on different combinations of unigrams, frequent ngrams, keyphrases selected by LocalMaxs, and synonym groups computed from the lattice structures and co-occurrence graphs, respectively.

For the first baseline system, we only use frequent unigrams in the index for information retrieval.

For the second system, we extract frequent ngrams, and use them together with frequent unigrams for information retrieval.

With LocalMaxs, we can select high quality keyphrases and use them together with frequent unigrams in the third system for information retrieval.

LocalMaxs is a bit aggressive in selecting keyphrases. By building a lattice structure, we can merge frequent ngrams with the selected keyphrases to form synonym groups and use the related groups in the fourth system for information retrieval.

Alternatively, we can also use co-occurrence graphs to compute synonym groups and incorporate them in the fifth system for information retrieval. Co-occurrence

graphs allow us to be more extensive in formulating synonym groups: not necessarily sharing the same prefix words in keyphrases, as long as they co-occur in similar sets of documents frequently.

4.3.1 Baseline for Information Retrieval

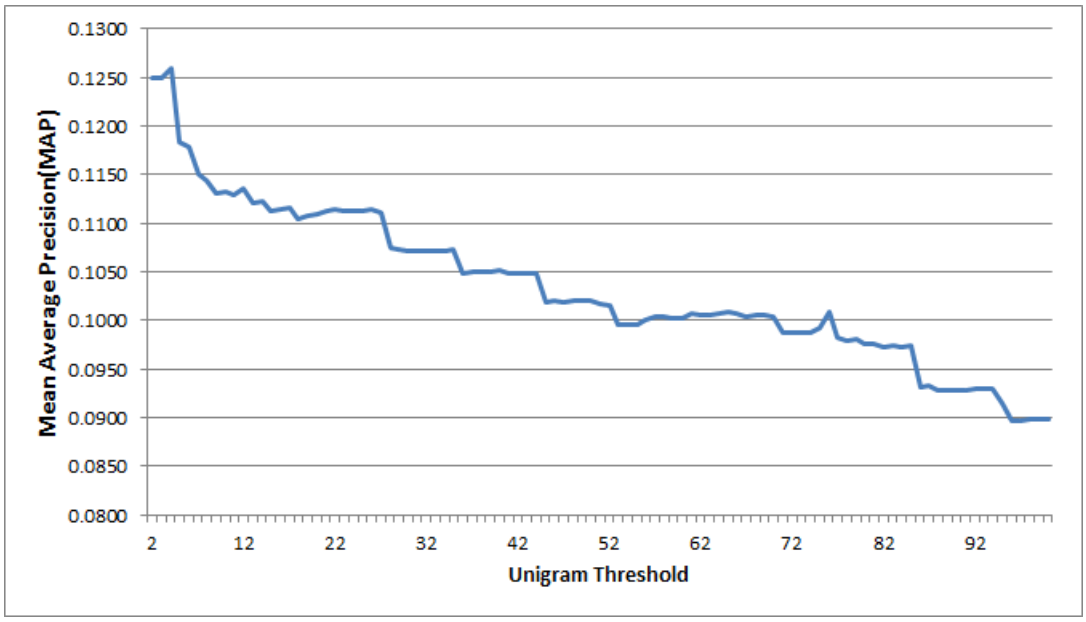

Figure 4.3: Performance of Unigram Models with Different Thresholds

In Figure 4.3, we show the performance of unigram models with different unigram thresholds from 2 to 100. Any unigrams with document frequents below these thresholds will not be included in the index. The results are measured by MAP values on TREC 6, 7, and 8 query sets.

We see that the curve rises from threshold 2 to 4, and the highest MAP (0.1256) is achieved at threshold 4. By cutting off unigrams with document frequencies below 4, a lot of rare words will be filtered out, as observed from Zipf's law. When the threshold moves beyond 4, the curve starts to go down quickly. There are two possible reasons to explain the the phenomenon. First, we only use one quarter of the TREC4&5 datasets

that may result in many words missing or with low frequencies. When such words are used in a query, we will not be able to match them with relevant documents in the dataset. Secondly, MAP is quite sensitive to the ranking of the retrieved documents, as demonstrated by the example in Figure 4.2.

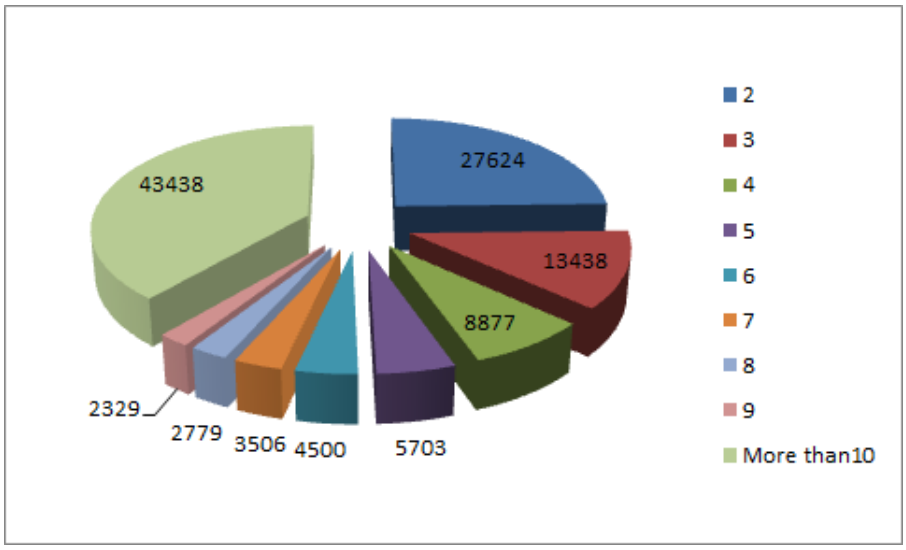

Figure 4.4: Frequency Distributions of Unigrams

In Figure 4.4, we break down all the unigrams by their document frequencies from threshold 2 to 10+. For a total of 112,194 unigrams, we find that the number of unigrams which appear 2 and 3 times in the dataset counts for 36.6% in total.

For our baseline experiment, we set the unigram threshold to 4, since it has the best performance in Figure 4.3 and also decreases the noise caused by low-frequency unigrams.

4.3.2 Selecting Keyphrases for Information Retrieval

Based on the unigrams selected from the baseline experiment, we try to use different thresholds for keyphrase selection from 2 to 10. As shown in Figure 4.5, the highest MAP(0.1475) is achieved at threshold 3, but it is very close to that achieved at 2 and 4.

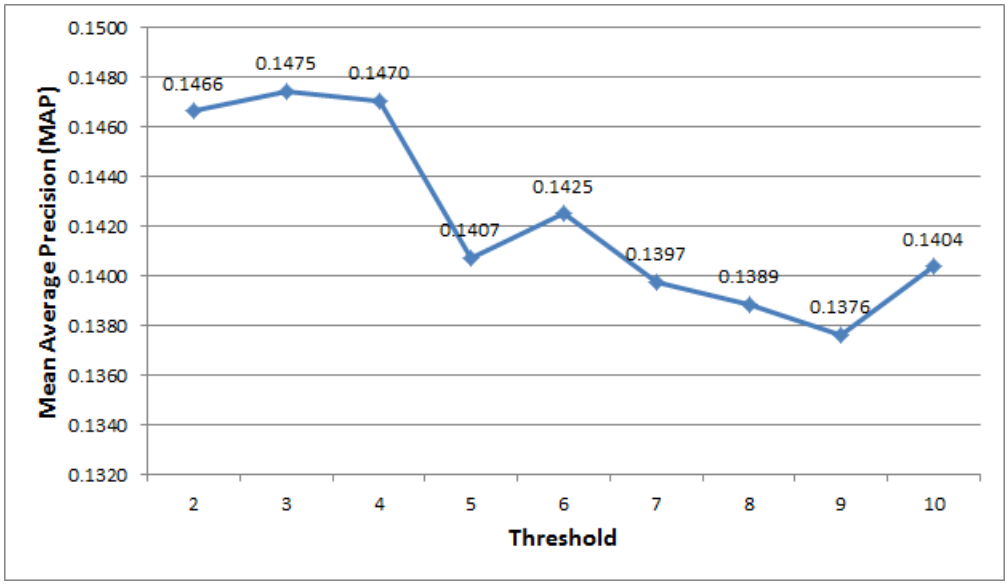

Figure 4.5: Performance of Keyphrases with Different Thresholds

As shown in Figure 4.5, the performance for thresholds 2, 3, and 4 are very close, and since we want to keep as many phrases as possible after cutting words at the unigram level, we choose the threshold of 2 to remove rare ngrams.

Compared with the baseline system for information retrieval, adding frequent ngrams that appear in two or more documents to the index helps increase the MAP value from 0.1256 to 0.1466, a significant improvement by a two-tailed t-test at level $\alpha = 0.01$ with P-value of 0.008.

4.3.3 Four Keyphrase Matching Methods after LocalMaxs

In this subsection, we compare the results of four keyphrase matching methods and analyze the possible reasons.

In Table 4.6 and 4.7, we show the MAP performance and the word-and-keyphrase size changes based on four keyphrase matching methods in different settings. For ngrams (unigrams+keyphrases), we find that except GN, the other three methods all did better than the baseline. The IN method is the best as shown in boldface in

Table 4.6: MAP Performance of Four Keyphrase Matching Methods

	Ngrams	Ngrams+LocalMaxs1	Ngrams+LocalMaxs2
GN	0.0915	0.1046	0.0932
GO	0.1491	0.1498	0.1506
IN	**0.1517**	**0.1532**	**0.1535**
IO	0.1470	0.1501	0.1492

Table 4.7: Index Size of Four Keyphrase Matching Methods

	Ngrams	Ngrams+LocalMaxs1	Ngrams+LocalMaxs2
GN	4106808	3026234	3724129
GO	5835388	3110815	3837429
IN	7276180	3550612	4577454
IO	7276180	3615428	5083942

Table4.6 with MAP values of 0.1517, 0.1532, and 0.1535 in three settings. The GN method is the worst with MAP (0.0915), corresponding to "greedy" and "no-overlap" match. This means that the GN method may lose potential matches for keyphrases, resulting in the worst performance. Furthermore, we find in Table 4.7 that the total number of ngrams for GN is much less than those for IN and IO, another reason why GN lose meaningful keyphrases since the majority of ngrams are keyphrases. The IO method is a bit worse than the IN method, but the total ngram numbers are the same. This may imply that 7,276,180 is the maximum number of ngrams based on threshold 4 for unigrams and 2 for keyphrases. The reason that IO's performance is worse than IN's performance maybe due to too much overlap, which adds duplicate counts for some ngrams.

After applying LocalMaxs1 and LocalMaxs2, we find that all four methods improved their performance, indicating that LocalMaxs is useful for keyphrase selection. However, localMaxs2 generates more numbers of ngrams than LocalMaxs1, with increase of 23.1%, 23.4%, 28.9%, and 40.6%, respectively.

Since both GO and IO methods generate overlaps of ngrams, the frequencies of

ngrams will be affected, making it hard to do keyphrase grouping with Lattice and Graphs. Therefore, we only use GN and IN methods in the next two subsections.

For the significance tests, we choose the best-performed IN model and compare its performance on the ngram model against those that use LocalMaxs1 and LocalMax2, respectively. We get P-values of 0.014 for the model with LocalMaxs1, and 0.016 for the model with LocalMaxs2, indicating that both models have made significant improvements over the ngram model at level of $\alpha = 0.05$

4.3.4 Two Keyphrases Matching Methods (GN and IN) after LocalMaxs and Lattice Grouping

In this experiment, we set the confidence level to 0.95 in order to group keyphrases in a lattice. Empirically, 0.95 is a comparatively strong confidence level, which allows us to examine the performance for keyphrase grouping relatively conservatively.

Table 4.8: MAP Performance of GN and IN after LocalMaxs and Lattice Grouping

	Ngram + LocalMaxs1 + Lattice0.95	Ngram + LocalMaxs2 + Lattice0.95
GN	0.1018	0.0919
IN	**0.1524**	0.1522

Table 4.9: Index Size of GN and IN after LocalMaxs and Lattice Grouping

	Ngram + LocalMaxs1 + Lattice0.95	Ngram + LocalMaxs2 + Lattice0.95
GN	3524483	4573169
IN	5019122	6747613

Table 4.10: Rates of Index Size Increase after LocalMaxs and Lattice Grouping

GN	16.5%	22.8%
IN	41.4%	47.4%

From Tables 4.8 and 4.9, we find that although Lattice grouping increases the index size, both GN and IN's performance is decreased slightly. By examining the results for

individual queries, we find that for some queries, the ranking of the retrieved results is worse than before. The main reason is that Lattice grouping may introduce noise in merging keyphrases, resulting in the decreased precision and recall.

Compared with index sizes of GN and IN before and after Lattice grouping, we find that the index sizes are all increased significantly, as shown in Table 4.10.

4.3.5 Two Keyphrases Matching Methods (GN and IN) after Co-occurrence Graph Based Grouping

As mentioned in the previous subsection, we use a strong confidence value to group keyphrases in the lattice, even though the performance is decreased. Since GN's performance is much worse than the baseline and the graph-based grouping is time consuming, we will focus only on the IN method in our later experiments. In the following, we describe our results in two parts. First, we do a strict grouping based on a strong confidence value, and second, we do a less strict grouping by lowering the confidence value.

Strict Grouping

Table 4.11: MAP Results after Graph Grouping is Added

	Ngram + LocalMaxs1 + Lattice0.95 + Graph0.95	Ngram + LocalMaxs2 + Lattice0.95 + Graph0.95
GN	0.1093	0.0983
IN	**0.1586**	0.1547

In Tables 4.11 and 4.12, we show the results after graph-based grouping is added at the confidence level of 0.95. We find that the IN's performance is improved significantly when compared with that of IN after LocalMaxs. Although LocalMaxs2 keeps a large number of keyphrases in the index, the retrieval performance is only increased

Table 4.12: Index Sizes and Group Numbers after Graph Grouping is Added

Ngram+LocalMaxs1+Lattice0.95+Graph0.95		
	Index size	Group Number
GN	3524483	3031251
IN	5019122	3175589
Ngram+LocalMaxs2+Lattice0.95+Graph0.95		
	Index size	Group Number
GN	4573169	3431251
IN	6747613	4275589

slightly. A two-tailed significance test between LocalMaxs2 with and without graph-based grouping for the IN method has P=0.071, which is not significantly different. On the other hand, a two-tailed significance test between LocalMaxs1 with and without graph-based grouping for the IN method has P=0.047, which is significantly different at level 0.05. As a result, using graph-based grouping indeed helps improve the retrieval performance.

For the significance tests, we compare the IN model (with the MAP value of 0.1586 in Table 4.11) with its corresponding model using LocalMaxs1 and Lattice (with the MAP value of 0.1524 in Table 4.8) and get the the P-value of 0.046, which is significant at level of α=0.05. We also compare the IN model (with the MAP value of 0.1586 in Table 4.11) with its corresponding model using LocalMaxs1 (with the MAP value of 0.1532 in Table 4.6) and get the the P-value of 0.042, which is also significant at level of α=0.05. Therefore, we conclude that the grouping of keyphrases that uses ngrams, LocalMaxs1, Lattices, and co-occurrence graphs (at 0.95 confidence level) is more effective than the other models developed in our experiments for information retrieval.

Table 4.13: Map Results with Graph Grouping at Different Confidence Levels

	Lattice0.95	**Lattice0.95+Graph**	**Confidence_Level**
IN	0.1524	0.1586	**0.95**
		0.1572	**0.85**
		0.1534	**0.75**
	Lattice0.90	**Lattice0.90+Graph**	**Confidence_Level**
	0.1507	0.1584	**0.90**
		0.1542	**0.80**
		0.1494	**0.70**

Less Strict Grouping

Encouraged by the results above, we try to decrease the confidence level to 0.85 and 0.75 for graph grouping. Additionally, we decrease confidence level to 0.90 for Lattice grouping and try the confidence levels 0.90, 0.80, and 0.70 for graph-based grouping.

As shown in Table 4.13, we find that with lower confidence levels for lattice and graph-based groupings, the performance of the IN method drops gradually, indicating that weak confidence levels are not suitable for information retrieval.

4.3.6 Examples of Keyphrase Grouping

To show the effect of keyphrase grouping, we use some examples from the query set to explain how it works. After grouping, "post polio" (from query 302) can be matched with "post polio syndrome" since they are put in the same group. Similarly, "Post-menopausal estrogen" (from query 355) is in the same group with "menstrual cycle fertile", and "european convention" is in the same group with "european convention human rights".

By examining our data, however, we also find some examples that are either uncertain or not related. For example, "so-called orphan drug" (from query 390) is in the same group with "anemia suffer patient", which may be related that is not certain. For another example, "african killer" is grouped with "african killer bee", which have

different meanings and should not be in the same group. Hopefully, such false positives can be controlled by setting appropriate confidence levels.

4.3.7 Discussion

From the above experiments, we find that phrase grouping is both interesting and useful, as shown in the examples of "post polio" and "post polio syndrome". If we use the ngram model, "post polio" will be found, but it will not be matched with "post polio syndrome", resulting a lower recall. After LocalMaxs, "post polio" is eliminated because its glue value is less than that for "post polio syndrome". With Lattice grouping, we get back "post polio", but at the same time, bring along noisy groupings. Finally, through graph-based grouping, we merge those ngrams that co-occur frequently together, which not only capture more keyphrases, but also reduce them to related groups. As a result, we have a potential to improve both precision and recall performance. In Figure 4.6, we show the changes of index sizes as we go through those steps for keyphrase extraction and grouping.

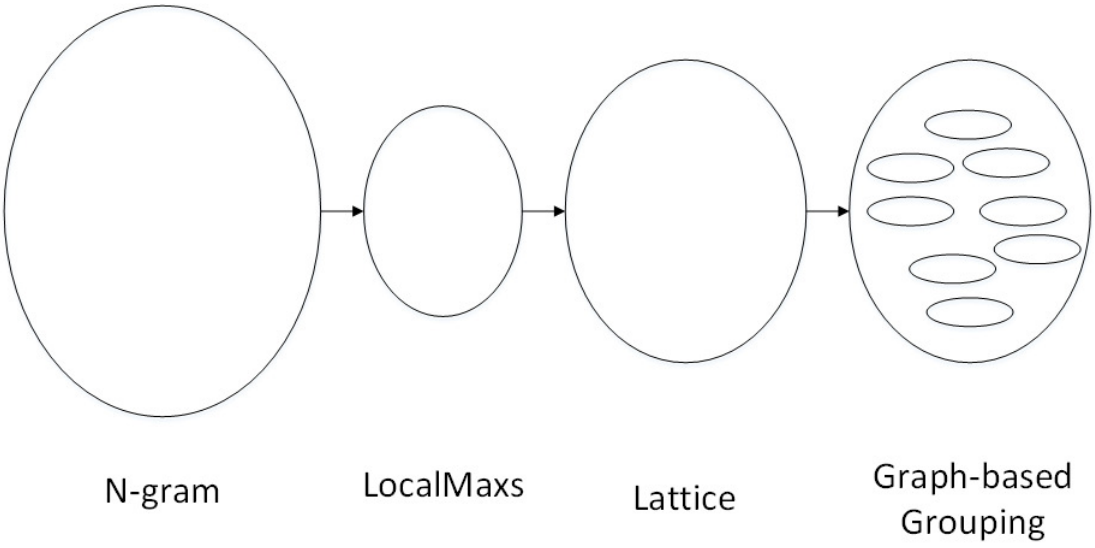

Figure 4.6: Changes of Index Size through Extraction and Grouping

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this thesis, we focused on the effective ways for keyphrase extraction and grouping based on association rule mining along with its application to information retrieval.

For keyphrase extraction, we first adapt the state-of-art BIDE algorithm for generating frequent ngrams which dramatically reduces the memory space by avoiding enumerating all possible ngrams and at the same time speeds up the process by pruning the search space.

To extract meaningful keyphrases from the frequent ngrams, we further apply the LocalMaxs method that does not require linguistic knowledge and is also language-independent. However, although LocalMaxs helps extract high quality keyphrases, it is a bit too aggressive in that some useful keyphrases may get filtered out, making it difficult to match the related keyphrases for information retrieval.

Consequently, we explore the use of lattice structures and co-occurrence graphs for keyphrase grouping so that the frequent ngrams generated by BIDE can be merged with the keyphrases extracted by LocalMaxs to form synonym groups. All keyphrases in a related group tend to co-occur together in the same set of documents and can be

matched against each other for information retrieval.

To demonstrate the effectiveness of our solution for keyphrase extraction and grouping, we apply it to information retrieval. With keyphrases, there can be different ways to match them in documents and queries. We tested our implementations on the standard TREC datasets. Our results indicate that adding frequent ngrams improves the retrieval performance significantly over the baseline made of keywords only. In addition, the performance can be further improved by selecting high quality keyphrases with the LocalMaxs method. Finally, by merging related keywords and keyphrases into synonym groups, we can increase the MAP value to 0.1586 from the baseline result of 0.1256, demonstrating the benefits of keyphrase extraction and grouping for information retrieval.

5.2 Future Work

5.2.1 Keyphrase Extraction

For keyphrase extraction, it would be useful to integrate linguistic knowledge, such as Part-of-Speech (POS) tags during data preprocessing and test on our datasets to see if the performance can be further improved.

5.2.2 Keyphrase Grouping

In our experiments, we propose a simple graph-based grouping mechanism for keyphrases which replies only on the co-occurrences of keyphrases. For future work, we could explore other semantic relations to enhance the quality of graph-based grouping, such as those used in PageRank, Wikipedia and WordNet. In particular, we could bring Wikipedia's semantic relation into groups right after the LocalMaxs is applied for keyphrase extraction.

5.2.3 Information Retrieval

In our experiments, we used the vector space model for information retrieval. It would be interesting to apply keyphrase extraction and grouping to other retrieval models such as language models.

Due to the time constraint and time consuming need of graph-based grouping, we only tested our solutions on a subset of the whole TREC4&5 datasets. With further improvement on the efficiency of the graph-based grouping, we could test our system on the whole TREC4&5 datasets to see if our results can be scaled up for larger datasets.

References

[1] Rakesh Agrawal, Tomasz Imielinski, and Swami Arun. Mining association rules between sets of items in large databases. In *ACM SIGMOD Record*, volume 22, pages 207–216. ACM, 1993.

[2] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, volume 1215, pages 487–499, 1994.

[3] James Allan, Jay Aslam, Nicholas Belkin, Chris Buckley, Jamie Callan, Bruce Croft, Sue Dumais, Norbert Fuhr, Donna Harman, David J Harper, et al. Challenges in information retrieval and language modeling: report of a workshop held at the center for intelligent information retrieval, university of massachusetts amherst, september 2002. In *ACM SIGIR Forum*, volume 37, pages 31–47. ACM, 2003.

[4] Jay Ayres, Jason Flannick, Johannes Gehrke, and Tomi Yiu. Sequential pattern mining using a bitmap representation. In *Conference on Knowledge Discovery in Data: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, volume 23, pages 429–435, 2002.

[5] Ken Barker and Nadia Cornacchia. Using noun phrase heads to extract document keyphrases. *Advances in Artificial Intelligence*, pages 40–52, 2000.

[6] Adam L. Berger and Vibhu O. Mittal. Ocelot: a system for summarizing web pages. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 144–151. ACM, 2000.

[7] Didier Bourigault. Surface grammatical analysis for the extraction of terminological noun phrases. In *Proceedings of COLING*, volume 92, pages 977–981, 1992.

[8] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1):107–117, 1998.

[9] Lee-Feng Chien. Pat-tree-based keyword extraction for chinese information retrieval. In *ACM SIGIR Forum*, volume 31, pages 50–58. ACM, 1997.

[10] Jonathan D Cohen. Highlights: Language- and domain-independent automatic indexing terms for abstracting. *JASIS*, 46(3):162–174, 1995.

[11] W. Bruce Croft and John Lafferty. *Language modeling for information retrieval*, volume 13. Springer, 2003.

[12] W. Bruce Croft, Howard R. Turtle, and David D. Lewist. The use of phrases and structured queries in information retrieval. In *Proceedings of the 14th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 32–45. ACM, 1991.

[13] David Crystal. *Dictionary of linguistics and phonetics*, volume 30. Wiley-Blackwell, 2011.

[14] Joaquim Ferreira da Silva, Gael Dias, Sylvie Guillore, and Jose Gabriel Pereira Lopes. Using localmaxs algorithm for the extraction of contiguous and non-contiguous multiword lexical units. *Progress in Artificial Intelligence*, pages 849–849, 1999.

[15] Joaquim Ferreira da Silva and Gabriel Pereira Lopes. Extracting multiword terms from document collections. In *Proceedings of the VExTAL: Venezia per il Trattamento Automatico delle Lingue*, pages 22–24, 1999.

[16] Bharath Dandala. *GRAPH-BASED KEYPHRASE EXTRACTION USING WIKIPEDIA*. PhD thesis, UNIVERSITY OF NORTH TEXAS, 2010.

[17] Ernesto D'Avanzo and Bernardo Magnini. A keyphrase-based approach to summarization: the lake system at duc-2005. In *Proceedings of DUC*, 2005.

[18] Ernesto DAvanzo, Bernardo Magnini, and A. Vallin. Keyphrase extraction for summarization purposes: The lake system at duc-2004. In *Proceedings of the 2004 Document Understanding Conference*, 2004.

[19] Gonenc Ercan and Ilyas Cicekli. Using lexical chains for keyword extraction. *Information Processing & Management*, 43(6):1705–1714, 2007.

[20] William B. Frakes and Ricardo Baeza-Yates. *Information retrieval: data structures and algorithms*. Prentice Hall PTR, 1992.

[21] Eibe Frank, Gordon W. Paynter, Ian H. Witten, Carl Gutwin, and Craig G. Nevill-Manning. Domain-specific keyphrase extraction. 1999.

[22] Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer-Verlag, Berlin Heidelberg, 1999.

[23] Fabrice Guillet and Howard J Hamilton. *Quality measures in data mining*, volume 43. Springer, 2007.

[24] Carl Gutwin, Gordon W. Paynter, Ian Witten, Craig Nevill-Manning, and Eibe Frank. Improving browsing in digital libraries with keyphrase indexes. *Decision Support Systems*, 27(1):81–104, 1999.

[25] Khaled M. Hammouda, Diego N. Matute, and Mohamed S. Kamel. Corephrase: Keyphrase extraction for document clustering. *Machine Learning and Data Mining in Pattern Recognition*, pages 634–634, 2005.

[26] Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery*, 15(1):55–86, 2007.

[27] Jiawei Han and Micheline Kamber. *Data mining: concepts and techniques*. Morgan Kaufmann, 2006.

[28] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *ACM SIGMOD Record*, volume 29, pages 1–12. ACM, 2000.

[29] Eduard Hovy, Laurie Gerber, Ulf Hermjakob, Michael Junk, and Chin-Yew Lin. Question answering in webclopedia. In *Proceedings of the TREC-9 Conference*, 2000.

[30] Fei Huang, Ying Zhang, and Stephan Vogel. Mining key phrase translations from web corpora. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 483–490. Association for Computational Linguistics, 2005.

[31] Anette Hulth. Improved automatic keyword extraction given more linguistic knowledge. In *Proceedings of the 2003 conference on Empirical methods in natural language processing*, pages 216–223. Association for Computational Linguistics, 2003.

[32] Weiwei Huo. *Automatic Multi-word Term Extraction and its Application to Web-page Summarization*. PhD thesis, University of Guelph, 2012.

[33] Christian Jacquemin, Judith L. Klavans, and Evelyne Tzoukermann. Expansion of multi-word terms for indexing and retrieval using morphology and syntax. In *Proceedings of the eighth conference on European chapter of the Association for Computational Linguistics*, pages 24–31. Association for Computational Linguistics, 1997.

[34] Mikalai Krapivin, Aliaksandr Autayeu, Maurizio Marchese, Enrico Blanzieri, and Nicola Segata. Keyphrases extraction from scientific documents: improving machine learning approaches with natural language processing. *The Role of Digital Libraries in a Time of Global Change*, pages 102–111, 2010.

[35] Roland Kuhn, Boxing Chen, George Foster, and Evan Stratford. Phrase clustering for smoothing tm probabilities–or, how to extract paraphrases from phrase tables. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, volume 2, pages 608–616, 2010.

[36] Cherif Chiraz Latiri, Hatem Haddad, and Tarek Hamrouni. Towards an effective automatic query expansion process using an association rule mining approach. *J. Intell. Inf. Syst.*, 39(1):209–247, 2012.

[37] Chiraz Latiri, Kamel Smaïli, Caroline Lavecchia, and David Langlois. Mining monolingual and bilingual corpora. *Intelligent Data Analysis*, 14(6):663–682, 2010.

[38] Chi-Hong Leung and Wing-Kay Kan. A statistical learning approach to automatic indexing of controlled index terms. *Journal of the American Society for Information Science*, 48(1):55–66, 1997.

[39] Su-Jian Li, Hou-Feng Wang, Shi-Wen Yu, and Cheng-Sheng Xin. Research on maximum entropy model for keyword indexing. *Chinese journal of computers*, 27(9):1192–1197, 2004.

[40] Dekang Lin and Xiaoyun Wu. Phrase clustering for discriminative learning. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 1030–1038. Association for Computational Linguistics, 2009.

[41] Bing Liu. *Web data mining: exploring hyperlinks, contents, and usage data*. Springer, 2007.

[42] Bing Liu, Wynne Hsu, and Yiming Ma. Integrating classification and association rule mining. In *Proceedings of the 4th*, 1998.

[43] Rachel Tsz-Wai Lo, Ben He, and Iadh Ounis. Automatically building a stopword list for an information retrieval system. *Proc. of DIR*, 5, 2005.

[44] Patrice Lopez and Laurent Romary. Humb: Automatic key term extraction from scientific articles in grobid. pages 248–251, 2010.

[45] Hans Peter Luhn. A statistical approach to mechanized encoding and searching of literary information. *IBM Journal of research and development*, 1(4):309–317, 1957.

[46] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*, volume 1. Cambridge University Press Cambridge, 2008.

[47] Yutaka Matsuo and Mitsuru Ishizuka. Keyword extraction from a single document using word co-occurrence statistical information. *International Journal on Artificial Intelligence Tools*, 13(01):157–169, 2004.

[48] Andrew McCallum and Kamal Nigam. A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48. Citeseer, 1998.

[49] Olena Medelyan and Ian H. Witten. Thesaurus based automatic keyphrase indexing. In *Proceedings of the 6th ACM/IEEE-CS joint conference on Digital libraries*, pages 296–297. ACM, 2006.

[50] Rada Mihalcea and Paul Tarau. Textrank: Bringing order into texts. In *Proceedings of EMNLP*, volume 4, pages 404–411. Barcelona, Spain, 2004.

[51] Rada Mihalcea, Paul Tarau, and Elizabeth Figa. Pagerank on semantic networks, with application to word sense disambiguation. In *Proceedings of the 20th international conference on Computational Linguistics*, page 1126. Association for Computational Linguistics, 2004.

[52] Jay M. Ponte and W. Bruce Croft. A language modeling approach to information retrieval. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 275–281. ACM, 1998.

[53] Korbinian Riedhammer, Benoit Favre, and Dilek Hakkani-Tur. A keyphrase based approach to interactive meeting summarization. In *Spoken Language Technology Workshop, 2008. SLT 2008. IEEE*, pages 153–156. IEEE, 2008.

[54] Marian-Andrei Rizoiu and Julien Velcin. Topic extraction for ontology learning. *Ontology Learning and Knowledge Discovery Using the Web: Challenges and Recent Advances*, 2011.

[55] Gerard Salton. Automatic information organization and retrieval. 1968.

[56] Gerard Salton, Chung-Shu Yang, and Clement T Yu. A theory of term importance in automatic text analysis. *Journal of the American society for Information Science*, 26(1):33–44, 1975.

[57] Amit Singhal. Modern information retrieval: A brief overview. *IEEE Data Engineering Bulletin*, 24(4):35–43, 2001.

[58] Fei Song and W. Bruce Croft. A general language model for information retrieval. In *Proceedings of the eighth international conference on Information and knowledge management*, pages 316–321. ACM, 1999.

[59] Michael M. Stark and Richard F. Riesenfeld. Wordnet: An electronic lexical database. In *Proceedings of 11th Eurographics Workshop on Rendering*. Citeseer, 1998.

[60] Michael Strube and Simone Paolo Ponzetto. Wikirelate! computing semantic relatedness using wikipedia. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 1419. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

[61] Gerd Stumme, Rafik Taouil, Yves Bastide, and Lotfi Lakhal. Conceptual clustering with iceberg concept lattices. *Proc. of GI-Fachgruppentreffen Maschinelles Lernen*, 1, 2001.

[62] Gerd Stumme, Rak Taouil, Yves Bastide, Nicolas Pasquier, and Lotfi Lakhal. Computing iceberg concept lattices with titanic. *Data & knowledge engineering*, 42(2):189–222, 2002.

[63] Simone Teufel. An overview of evaluation methods in trec ad hoc information retrieval and trec question answering. In *Evaluation of Text and Speech Systems*, pages 163–186. Springer, 2007.

[64] Amy J.C. Trappeya, Fu-Chiang Hsua, Charles V. Trappeyb, and Chia-I. Linc. Development of a patent document classification and search platform using a back-propagation network. *Expert Systems with Applications*, 31(4):755–765, 2006.

[65] Chiu-yu Tseng and Shao-huang Pin. Modeling prosody of mandarin chinese fluent speech via phrase grouping. *Proceedings of ICSLT-O-COCOSDA 2004*, 2004.

[66] Peter D. Turney. Learning to extract keyphrases from text. 1999.

[67] Peter D. Turney. Learning algorithms for keyphrase extraction. *Information Retrieval*, 2(4):303–336, 2000.

[68] Ellen M. Voorhees and Donna Harman. Overview of the fifth text retrieval conference (trec-5). In *The Fifth Text REtrieval Conf..(TREC-5)*, 1996.

[69] Xiaojun Wan and Jianguo Xiao. Single document keyphrase extraction using neighborhood knowledge. In *Proceedings of AAAI*, pages 855–860, 2008.

[70] Xiaojun Wan and Jianguo Xiao. Exploiting neighborhood knowledge for single document summarization and keyphrase extraction. *ACM Transactions on Information Systems (TOIS)*, 28(2):8, 2010.

[71] Jianyong Wang and Jiawei Han. Bide: Efficient mining of frequent closed sequences. In *Data Engineering, 2004. Proceedings. 20th International Conference on*, pages 79–90. IEEE, 2004.

[72] Yanbo Wang. Various approaches in text pre-processing. *TM Work Paper No*, 2, 2004.

[73] Christian Wartena, Rogier Brussee, and Wout Slakhorst. Keyword extraction using word co-occurrence. In *Database and Expert Systems Applications (DEXA), 2010 Workshop on*, pages 54–58. IEEE, 2010.

[74] Ian H. Witten, Gordon W. Paynter, Eibe Frank, Carl Gutwin, and Craig G. Nevill-manning. Kea: Practical automatic keyphrase extraction. In *Proceedings of the fourth ACM conference on Digital libraries*, pages 254–255. ACM, 1999.

[75] Mohammed J Zaki. Mining non-redundant association rules. *Data mining and knowledge discovery*, 9(3):223–248, 2004.

[76] Oren Eli Zamir. *Clustering web documents: a phrase-based method for grouping search engine results*. PhD thesis, University of Washington, 1999.

[77] Torsten Zesch. *Study of Semantic Relatedness of Words Using Collaboratively Constructed Semantic Resources*. PhD thesis, TU Darmstadt, 2010.

[78] Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 334–342. ACM, 2001.

[79] Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Transactions on Information Systems (TOIS)*, 22(2):179–214, 2004.

[80] Kuo Zhang, Hui Xu, Jie Tang, and Juanzi Li. Keyword extraction using support vector machine. In *Advances in Web-Age Information Management*, pages 85–96. Springer, 2006.

[81] Yanchang Zhao, Chengqi Zhang, and Longbing Cao. *Post-mining of association rules: techniques for effective knowledge extraction*. Information Science Reference-Imprint of: IGI Publishing, 2009.