School of Information Technology

A Frequent Max Substring Technique for Thai Text Indexing

Todsanai Chumwatana

This thesis is presented for the Degree of

Doctor of Philosophy of

Murdoch University

May 2011

Declaration

I declare that this thesis is my own account of my research and contains as its main content work which has not previously been submitted for a degree at any tertiary education institution.

Todsanai Chumwatana

May, 2011

Acknowledgments

I would like to take this opportunity to acknowledge and thank the following people and organizations that helped me to complete this thesis.

First of all I would like to thank my principal supervisor, Associate Professor Dr Kevin Wong, for his support, guidance, comments and encouragement throughout the period of my PhD research. From the first day of my study, Professor Kevin Wong always offered me opportunities and taught me how to do good research. Throughout my study, Professor Kevin Wong always walked beside me and supported me. I am very grateful for his helpful advice and his efforts to explain things to me.

I would also like to thank my associate supervisor, Dr Hong Xie, for his suggestions. Besides my principal supervisor and associate supervisor, I wish to express my gratitude to Associate Professor Dr Lance Chun Che Fung. Professor Fung has always offered me and his students support, opportunities and mental stimulation. Many thanks to my PhD colleagues and Thai students who have supported me and given me wonderful friendships. Thanks to all of them for a memorable time.

My research would not have become a reality without the financial support of my family during the period of my study. I cannot forget to thank my lovely family: my father Ponlasak Chumwatana, my mother Tassanee Kaewbavon and my two older sisters Nattaya Mohjhaw and Chalakod Chumwatana for everything. They have always supported and encouraged me and lifted my spirits since I opened my eyes to see the world.

Finally, I would like to thank the School of Information Technology, Murdoch University, for providing me with all the necessary facilities for my research.

Abstract

This research details the development of a novel methodology, called the frequent max substring technique, for extracting indexing terms and constructing an index for Thai text documents.

With the rapidly increasing number of Thai digital documents available in digital media and websites, it is important to find an efficient Thai text indexing technique to facilitate search and retrieval. An efficient index would speed up the response time and improve the accessibility of the documents. Up to now, not much research in Thai text indexing has been conducted as compared to more commonly used languages like English. The more commonly used Thai text indexing technique is the word inverted index, which is language-dependent (i.e. requires linguistic knowledge). This technique creates word document indices on document collection to enable an efficient keyword based search. However, when using the word inverted index technique, Thai text documents need to be parsed and tokenized into individual words first. Therefore, one of the main issues is how to automatically identify the indexing terms from the Thai text documents before constructing the index. This is because the syntax of Thai language is highly ambiguous and Thai language is non-segmented (i.e. a text document is written continuously as a sequence of characters without explicit word boundary delimiters). To index Thai text documents, most language-dependent indexing techniques have to rely on the performance of a word segmentation approach in order to extract the indexing terms before constructing the index. However, word segmentation is time consuming and segmentation accuracy is heavily dependent either on the linguistic knowledge used in the underlying segmentation algorithms, or on the dictionary or corpus used in the segmentation. It is for this reason that most language-dependent indexing techniques are time consuming and require additional storage space for storing dictionary or corpus or manually crafted rules resource.

Apart from the language dependant indexing techniques, some language-independent techniques have been proposed as an alternative indexing technique for Thai language such as the n-gram inverted index and suffix array approaches. These approaches are simple and fast as they are language-independent, and do not require linguistic knowledge of the language, or the use of a dictionary or a corpus. However, the limitation of these techniques is that they require more storage space for extracting the indexing terms and constructing the index.

To address the above limitations, this thesis has developed a frequent max substring technique that uses language-independent text representation, which is computationally efficient and requires small storage place. The frequent max substring technique improves the performance in terms of construction time over the language-dependent techniques (i.e. the word inverted index) as this technique does not require text pre-processing tasks (i.e. word segmentation) in extracting the indexing terms before indexing can be performed. This technique also improves space efficiency compared to some existing language-independent techniques. This is achieved by retaining only the frequent max substrings, which are strings that are both long and frequently occurring, in order to reduce the number of insignificant indexing terms from an index.

To demonstrate that the frequent max substring technique could deliver its performance, experimental studies and comparison results on indexing Thai text documents are presented in this thesis. The technique was evaluated and compared in term of indexing efficiency and retrieval performance. The results show that the frequent max substring technique is more computationally efficient when compared to the word inverted index, and also that it requires less space for indexing when compared to some language-independent techniques.

Additionally, this thesis shows that the frequent max substring technique has an advantage in terms of versatility, as it can also be combined with other Thai language-dependent techniques to become a novel hybrid language-dependent technique, in order to further improve the indexing quality. This technique can also be used with a neural network to enhance non-segmented document clustering. The frequent max substring technique also has the flexibility to be applied to other non-segmented texts like the Chinese language and genome sequences in bioinformatics due to its language-independency feature.

List of Publications Related to this Thesis

P1. T. Chumwatana, K. W. Wong and H. Xie, ‘Using Frequent Max Substring Technique for Thai Text Indexing’, accepted for publication in the *Australian Journal of Intelligent Information Processing Systems (AJIIPS).*

P2. T. Chumwatana, K. W. Wong and H. Xie, ‘A SOM-Based Document Clustering Using Frequent Max Substrings for Non-Segmented Texts’, *In the Journal of Intelligent Learning Systems and Applications (JILSA),* 2010.

P3. T. Chumwatana, K. W. Wong and H. Xie, ‘Using Frequent Max Substring Technique for Thai Keyword Extraction used in Thai Text Mining’, *In Proceedings of the 2nd International Conference on Soft Computing, Intelligent System and Information Technology (ICSIIT 2010),* Bali, Indonesia, 1-2 July 2010.

P4. T. Chumwatana, K. W. Wong and H. Xie, ‘Non-segmented Document Clustering Using Self-organizing map and Frequent Max Substring Technique’, Lecture Notes in Computer Science, Springer Berlin/Heidelberg, LNCS 5864, 2009, pp. 691–698.

P5. T. Chumwatana, K. W. Wong and H. Xie, ‘Non-segmented Document Clustering Using Self-organizing map and Frequent Max Substring Technique’, *In16th International Conference on Neural Information Processing (ICONIP 2009),* Bangkok, Thailand, 2009.

P6. T. Chumwatana, K. W. Wong and H. Xie, ‘Indexing Non-Segmented Texts using n-Gram Inverted Index and Frequent Max Substring: A Comparison of Two Techniques,’ *In 10th Postgraduate Electrical Engineering & Computing Symposium (PEECS2009),* Perth, Australia, 2009.

P7. T. Chumwatana, K. W. Wong and H. Xie, ‘An Automatic Indexing Technique for Thai Texts using Frequent Max Substring,’ *In The 8th International Symposium on Natural Language Processing, 2009 (SNLP '09),* Bangkok, Thailand, 2009.

P8. T. Chumwatana, K. W. Wong and H. Xie, ‘An Efficient Text Mining Technique’, *In 9th Postgraduate Electrical Engineering & Computing Symposium (PEECS2008),* Perth, Australia, 2008.

P9. T. Chumwatana, K. W. Wong and H. Xie, ‘Thai Text Mining to Support Web Search for E-commerce’, *In The 7th International Conference on e-Business 2008 (INCEB2008),* Bangkok, Thailand, 2008.

P10. T. Chumwatana, K. W. Wong and H. Xie, ‘Frequent Max Substring Mining for Indexing’, *International Journal of Computer Science and System Analysis (IJCSSA), India,* 2008.

P11. T. Chumwatana, K. W. Wong and H. Xie, ‘Frequent Max Substring Mining’, *In 8th Postgraduate Electrical Engineering & Computing Symposium (PEECS2007),* Perth, Australia, 2007.

Contributions of this Thesis

The contributions in this thesis which have already been published and reported are described below and summarized in Table 1.

A survey and review of various techniques in the Thai text indexing area has been completed. This work forms the basis of Chapter 2. Parts of this work have been published in conference papers P3, P7, P9 and journal paper P1.

The development of the novel indexing technique, called the frequent max substring technique, forms a part of Chapter 3. Results from this work are reported in conference papers P3, P7, P8, P9, P11 and journal papers P1 and P10. Some of these publications also include the experimental studies, comparison results and discussion on indexing Thai text documents. Another contribution documented in Chapter 3 is the establishment of a methodology for evaluating the frequent max substring technique by comparing it to other indexing techniques such as the word inverted index, n-gram inverted index, Vilo's technique and suffix array. The comparison was based on indexing efficiency and retrieval performance.

The contribution in Chapter 4 is the establishment of the integration of the frequent max substring technique with other Thai language-dependent techniques to create a novel language-dependent technique. The hybrid method is used for extracting and indexing meaningful indexing terms from Thai text documents. Parts of this chapter have been published in journal paper P1.

The work on an integrated method using the frequent max substring technique with self-organizing map (SOM) for non-segmented document clustering is published in conference paper P5 and lecture notes in computer science paper P4. Conference paper P5 was later extended to journal paper P2, which has been described in Chapter 5. Journal paper P2 was published in *Journal of Intelligent Learning Systems and Applications* and it showed that the frequent max substring technique can be used with self-organizing map to enhance non-segmented document clustering in order to improve the efficiency of Thai information retrieval.

Chapter 6 discussed the application of the frequent max substring technique to other non-segmented texts such as non-segmented languages (the Chinese language) and genome sequences. This demonstrated that the frequent max substring technique is versatile as it can be used not only for Thai text indexing but also for indexing other non-segmented texts. This work is reported in conference paper P6. The paper also presents some comparison results and discussion on indexing non-segmented texts.

Table 1 Summary of the Contribution of the Thesis

Chapter	Contributions	Paper No
***Chapter 2*: Thai Text Indexing**	Presents a literature survey on previous research in the Thai text indexing area and identifies the limitations of existing Thai text indexing techniques.	P1, P3, P7, P9
***Chapter 3*: Frequent Max Substring Technique**	Successfully developed the frequent max substring technique to perform Thai text indexing for language-independent technique.	P1, P3, P7, P8, P9, P10, P11
***Chapter 4*: Hybrid Method: Integration of the Frequent Max Substring Technique and Thai Language-Dependent Technique**	Successfully developed a hybrid method by combining the frequent max substring technique and language-dependent technique to perform Thai text indexing using linguistic knowledge.	P1
***Chapter 5*: Non-Segmented Document Clustering Using Self-Organizing Map and the Frequent Max Substring Technique**	Developed an integrated method using the frequent max substring technique with self-organizing map (SOM) to enhance the non-segmented document clustering.	P2, P4, P5
***Chapter 6*: Non-Segmented Text Problems**	Successfully implemented the proposed technique with some other non-segmented texts like Chinese and genome sequence.	P6

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Overview

1.1 Overview

In recent years, there has been a rapid growth in the number of online documents in the Thai language. In order to facilitate search and retrieval of these online documents, it is necessary to build an efficient index for these documents first. Indexing Thai text documents is one of the essential processes in Thai information retrieval [1], [2], [3]. An efficient index would speed up the response time of the search engine and improve the accessibility of online documents. However, the syntax of the Thai language is highly ambiguous and there has not been much research into Thai text indexing when compared to more commonly used languages like English. Thus, building an efficient index for Thai text documents remains a challenging task.

A Thai text document consists of a string of symbols without explicit word boundary delimiters. To some extent, this lack of explicit word delimiters is a feature that also exists in many other Asian languages, including Chinese, Japanese and Korean (CJK) [4], [5] [6]. These languages are known as non-segmented languages. They are very different from English and other European languages, the words of which are explicitly delimited by space or other symbols [2]. English and other European languages are often referred to as segmented languages [7]. While it is relatively easy for a native user of a non-segmented language to determine these word boundaries, it is extremely hard for a computer to detect the boundaries. In the case of the Thai language, there are a number of additional problems due to the ambiguity of the structure of the language [8].

For instance, there is no word inflection or change in the word form as an expression of case, tense and gender [4]. In Thai, the same form of a word in different positions contains different syntactic properties and therefore has different meanings [4]. In addition, there are no special characters to separate words, phrases and sentences in the Thai writing system (i.e. Thai words are not delimited by spaces or other special symbols like the comma and full stop). The spaces are only used to break the ideas into a format that can gain readers' attention, but they are not used to split words, phrases or sentences. Furthermore, unlike English, the Thai language has no capital letters to identify a proper noun or the beginning of a sentence [9]. Consequently, indexing Thai text documents is much harder than indexing English text documents. One of the main issues here is how to automatically identify the indexing terms from a Thai text document when constructing an index. In prior work, Thai indexing terms were manually identified by experts but this process can be very time consuming and labor-intensive [10]. Meanwhile, most of the indexing techniques used in information retrieval systems are designed for English and other European languages, where the word boundary and characteristic are clearly defined [7]. For these segmented languages, there are many successful techniques used to identify the indexing terms from the text document. However, these methods are usually not applicable to the Thai language which is a non-segmented language. Although some techniques have been developed for extracting the indexing terms from non-segmented languages such as Chinese, Japanese, Korean, Thai or genome sequence, many challenging tasks for these non-segmented languages still remain. To gain a better understanding, those challenges will be discussed as follows.

The majority of indexing term extraction techniques for the Thai language are based on word segmentation, sometimes called word based approach (i.e. requires understanding of the language) which is one of the most widely used information extraction techniques in Natural Language Processing (NLP) [4]. However, the preparation of these approaches is time consuming, as well-defined linguistic knowledge is required for these techniques. In addition, there are also many disadvantages for these approaches due to the limitations of the segmentation ability [5], [11], [2], [10]. Currently the methods for extracting indexing terms from Thai text documents can be classified into the following three types: dictionary based, Thai grammar rule based and machine learning based [2], [12].

The dictionary based approach uses a set of possible words in a dictionary for matching and segmenting an input text document into words. The input text documents are parsed and segmented into indexing terms for constructing an inverted index. An indexing term extracted by using the dictionary based approach is a complete linguistic unit from the input text documents, rather than a fraction of words, because every segmented indexing term has a meaning. However, the performance of the dictionary based approach depends heavily on the quality, domain and size of the dictionary [13], [14]. This means that by using a different dictionary, this method could produce different results. Firstly, the size of the dictionary (i.e. the amount of vocabulary in the dictionary) can impact on the performance of word segmentation. A bigger dictionary could generate more positive matches thus giving a better result for word segmentation. Secondly, the performance also depends on the quality and domain of the dictionary. For instance, using a dictionary of computer technology on the input text documents from business domains will be unlikely to provide a good result. Furthermore, the dictionary based approach would not perform well if the input text documents contain too many new words or proper nouns, as these words are unlikely to be found in the dictionary. It must be noted that, unlike many European languages, word capitalization for identifying proper nouns is not utilized in Thai and many Asian languages [9]. Hence, with the dictionary based approach, it is even harder to identify a word if that word is not in the dictionary. Additionally, segmentation ambiguity can happen when the same text can be segmented in many different ways (especially true for Thai), leading to different meanings or pronunciation, as will be described in Chapter 2 [4].

The Thai grammar rule based method uses basic Thai word formation rules to analyse the input texts in order to segment and identify words from the text documents [15], [16]. This technique has been used for syllable segmentation, using the rule of syllable combination for segmentation, rather than word segmentation. In the Thai language, grammar has many exceptions but the majority of the word usages still follow basic word formation rules. The regular expression for the monosyllable will be formed and used in segmentation. These basic monosyllabic words may or may not be meaningful. The weakness of the Thai grammar rule based approach is that it has low segmentation accuracy, as this technique usually extracts a fraction of a word rather than a complete word [17], [5], [10]. Another disadvantage is that this approach requires manually crafted rules for segmentation [5].

The machine learning based approach uses some learning algorithms to realise the model [12], [2], [18], [19]. This approach uses the machine learning based technique to learn from text collections. This approach requires an appropriate word-segmented Thai text corpus to provide enough information to train the system. The machine learning based approach aims to address the drawbacks of dictionary based approaches. Using a tagged corpus in which word boundaries are explicitly marked with special annotations, machine learning based algorithms build statistical models based on the features of the characters surrounding the boundaries. The common features for Thai word segmentation models are the identities and categories of characters within an n-gram of characters surrounding a candidate word boundary [2]. The machine learning based technique therefore does not require word patterns or dictionaries. However, the main disadvantage of the machine learning based approach is that its performance depends critically on the characteristics of the document domain and the size of the training corpus. Besides, the construction of the corpus is very time consuming and requires more storage space. For example, if a model is constructed based on a corpus from one specific domain, it might not perform well on documents from another domain [19]. In addition, the problem of segmentation in the Thai language is difficult for machine learning. This is due to the complexity in the linguistic structure and the variety of context cues to determine the correct segmentation.

The three approaches discussed above are all language-dependent. They rely on the use of a dictionary, a corpus, or linguistic knowledge of the Thai language in order to extract indexing terms before constructing an index. These methods can also be applied to other Asian languages such as Chinese, Japanese and Korean (CJK) but well-defined linguistic knowledge for each particular language is required to perform word segmentation. The effectiveness of these three approaches can be measured by comparing the segmented words with the hand segmentation corpus that is manually prepared by Thai experts. Although techniques for word segmentation have been widely studied in natural language processing, there are still many challenges due to the ambiguity in Thai language usage [10], [20].

For Thai information retrieval, it has been shown that word segmentation may not be necessary for identifying indexing terms in some cases, as long as users are able to find the relevant documents related to their query [10]. The main issue for information retrieval is to automatically identify the indexing terms from a large collection of text documents or text database, and use these indexing terms to retrieve documents that are likely to be relevant to the user’s query for efficient retrieval.

As a result, an alternative indexing technique, known as n-gram based approach, was proposed. The n-gram based approach is acknowledged as a workable solution to information retrieval problems [21], [22], [10]. This approach is language-independent as it does not require linguistic knowledge of the language, or the use of a dictionary or a corpus. This approach is not concerned with the meaning of indexing terms. Currently, this technique has been one of the most often used indexing techniques for Asian language documents (Chinese, Japanese and Korean) and it has also been applied to many related information retrieval fields such as document similarity comparison and language classification [23], [24]. It has even been used in analyzing genome sequences in bioinformatics [25]. However, the n-gram based approach has never been used for indexing Thai text documents, although this technique has often been applied for indexing many non-segmented languages.

While the n-gram based approach outperformed other methods in terms of retrieval effectiveness for many Asian languages and other sequence patterns [10], its disadvantage is that it requires query processing and text pre-processing to extract n-gram terms before retrieval and indexing can be performed. This technique also requires a larger space for storing indexing terms and uses more retrieval time in looking up the relevant documents when compared to a word based approach [10], [26].

Additionally, the suffix array approach is one of the language-independent techniques used for indexing Thai text documents [1]. The suffix array approach identifies substring indexing based on suffixes, which does not require text pre-processing and query processing. This technique is used to construct a substring index, in order to allow for finding the relevant documents containing the user's query. However, the drawback in terms of index size of the suffix array technique seems very critical. Since the size of electronically stored information in the Thai language has grown exponentially, the method of suffix array is not practical to use in some applications, especially for the large collection of text documents, as this technique constructs the complete set of substrings as will be described in Chapters 2 and 3.

In order to gain a better understanding of the problems of Thai text indexing techniques, the drawbacks of Thai text indexing techniques are summarized in Figure 1.1.

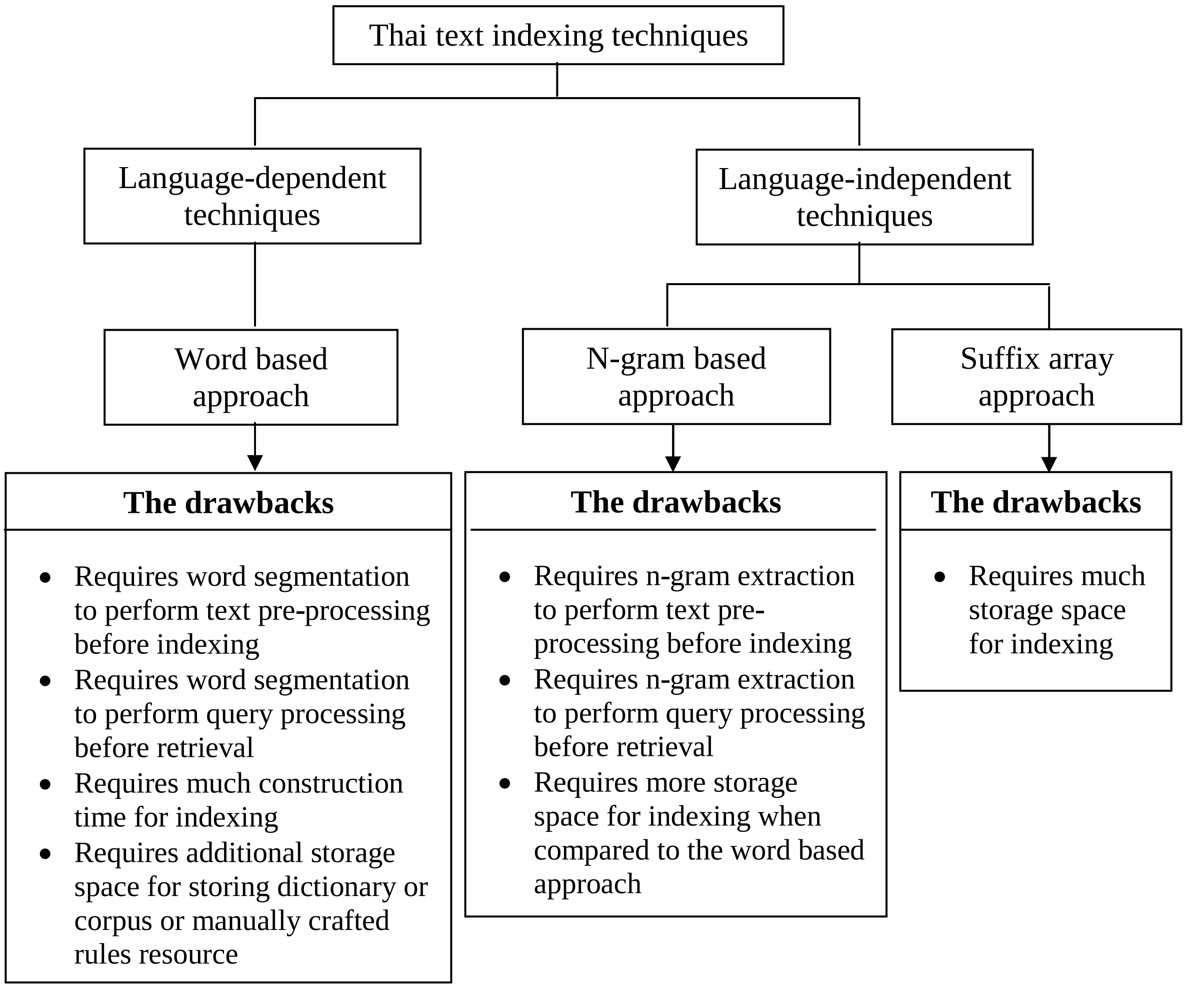

Thai text indexing techniques
Language-dependent techniques
Language-independent techniques
Word based approach
N-gram based approach
Suffix array approach
The drawbacks
• Requires word segmentation to perform text pre-processing before indexing
• Requires word segmentation to perform query processing before retrieval
• Requires much construction time for indexing
• Requires additional storage space for storing dictionary or corpus or manually crafted rules resource
The drawbacks
• Requires n-gram extraction to perform text pre-processing before indexing
• Requires n-gram extraction to perform query processing before retrieval
• Requires more storage space for indexing when compared to the word based approach
The drawbacks
• Requires much storage space for indexing

Figure 1.1. Some drawbacks of existing Thai text indexing techniques

1.2 Objectives

In order to address the drawbacks of the existing techniques for Thai text indexing, this research hence focuses on the following objectives:

- **Developing an efficient Thai text indexing technique:**

  The primary objective of this research in the domain of Thai information retrieval is to develop a new indexing technique that does not rely on linguistic knowledge of the Thai language and requires less memory space for extracting indexing terms and constructing an index for Thai text documents. In order to be a fast and simple approach, this technique should be language-independent (i.e. does not require the use of a dictionary or corpus or grammatical knowledge of a language). Hence it would not rely on the availability and quality of dictionaries and corpuses, or grammatical knowledge of a language. This is intended to address one of the main weaknesses of the most often used Thai text indexing technique, which is heavily dependent on in-depth linguistic knowledge or the use of a dictionary or a corpus in order to extract indexing terms before constructing an index.

- **Developing an efficient algorithm to improve the space and time complexity:**

  The other aim of the research is to improve the space efficiency in text indexing to overcome the drawbacks of some of the existing language-independent techniques. It is the intention of this research to extract only strings that are both long and frequently occurring, rather than individual words, from Thai text documents as the indexing terms. It is hoped that by doing so the storage size of the index would be reduced without affecting the functionality of the index.

- **The ability to integrate with language-dependent techniques:**

  The developed technique should also be able to integrate with existing Thai language-dependent techniques, to become a novel language-dependent technique for indexing Thai text documents.

- **The ability to integrate with document clustering techniques:**

  To increase efficiency further, it should be possible to integrate the developed technique in this research with document clustering techniques. This indexing technique should be able to work with other document clustering techniques to enhance the performance of non-segmented document clustering.

1.3 Contributions

The key contribution of this thesis is the successful development of a novel text indexing technique suitable for Thai information retrieval. In particular, this thesis provides several contributions as follows:

.

- **Successfully developed a novel indexing technique, called the frequent max substring technique, to perform Thai text indexing for language-dependent and language-independent techniques:**

  The frequent max substring technique is a novel indexing technique proposed for extracting indexing terms and constructing an index for Thai text documents. The set of indexing terms extracted from the frequent max substring technique contains all substrings occurring frequently in Thai text documents. Only one pass is required to obtain this set of indexing terms. One of the strengths of this proposed technique is that it retains a relatively smaller number of indexing terms without sacrificing its effectiveness in information retrieval, when compared to most language-independent techniques. This is because each frequent max substring extracted can represent multiple frequent substrings that occur in the text documents. As a result, applying the frequent max substring technique to indexing Thai text documents leads to a saving in storage space for storing the index. This technique also improves the retrieval performance in order to prospectively support the continuous growth of Thai electronic documents.

- **Developed an integrated method using the proposed frequent max substring technique with self-organizing map (SOM) to enhance non-segmented document clustering:**

  The proposed frequent max substring technique is also versatile, as it can be used with other techniques to enhance non-segmented document clustering. Improvement has been achieved when the frequent max substring technique is integrated with self-organizing map (SOM) for document clustering. This new method improves the efficiency of information retrieval, as the document cluster map generated from the proposed method can successfully be used to find the relevant documents more efficiently, when compared to the hierarchical based document clustering using single words.

- **Successfully applied the proposed technique with some other non-segmented texts like Chinese and genome sequence:**

  Apart from its application to indexing problem for Thai text documents, the proposed technique is also successfully applied to the following areas: indexing genome sequences in bioinformatics and indexing text documents for other non-segmented languages like Chinese. The main strength of the proposed technique is that it was proposed as a language-independent technique, which does not rely on the use of any dictionary or corpus or grammatical knowledge of language. Because of this, the proposed technique is not limited to just indexing Thai text documents. It could also be applied for many non-segmented languages such as Chinese and genome sequences, which are regarded as non-segmented texts. This shows that the proposed technique can be applied to other similar applications.

1.4 Thesis outline

This chapter has presented the introduction and overview of the research. It also includes the objective and contribution of this thesis. The rest of this thesis is organized as follows.

Chapter 2 gives a brief overview of Thai text indexing, and the linguistic characteristics of the Thai language. Thai text indexing techniques and related past research works are reviewed. This review also discusses the drawbacks of Thai text indexing techniques and Thai segmentation problems, which were the motivation behind this research into the understanding and design of the frequent max substring technique for indexing Thai text documents.

Chapter 3 introduces the frequent max substring technique and the frequent max substring terminologies. The algorithm and procedure of constructing the frequent suffix trie structure for indexing Thai text documents is described. Substring indexing and frequent substring indexing, which are used as a basic concept to design the frequent max substring technique, are also examined in this chapter. The experimental studies, comparison results and discussion on indexing Thai text documents are finally presented in Chapter 3.

Chapter 4 presents the use of a hybrid method as a language-dependent technique. The hybrid method that combines the proposed frequent max substring technique and a Thai language-dependent technique is described to process Thai text documents into meaningful indexing terms. Some experimental results are also presented to show the performance of the hybrid method when compared to other language-dependent techniques.

In Chapter 5, the application of the frequent max substring technique together with the self-organizing map for non-segmented document clustering is presented. An overview of document clustering and keyword extraction is given in this chapter. This chapter is also included with the presentation of experimental studies and comparison results on clustering the Thai text documents to substantiate the discussion.

Chapter 6 investigates non-segmented text problems and understands the characteristic of non-segmented texts like non-segmented languages (i.e. the Chinese language) and genome sequences. The related research works on the Chinese language and genome sequences are reviewed. Applying the frequent max substring technique to the Chinese language and genome sequences are demonstrated. The presentation of experimental studies, comparison results and the discussion on indexing Chinese text documents and genome sequences is finally presented in this chapter.

Finally, Chapter 7 presents the thesis conclusion, which summarizes the major contribution and outcomes of this research. The limitations and possible directions of future work are also discussed.

Chapter 2

Thai Text Indexing

2.1 Introduction

There is an ongoing challenge to develop more efficient text indexing techniques for Thai text documents, in order to enhance the performance of Thai information retrieval. To meet the challenges, a number of indexing techniques have been proposed for the Thai language. These techniques were designed to be used in information retrieval systems, to find required information from the huge amount of text documents. Figure 2.1 provides an overview of a general indexing and retrieval system, before the detail of the Thai text indexing will be described.

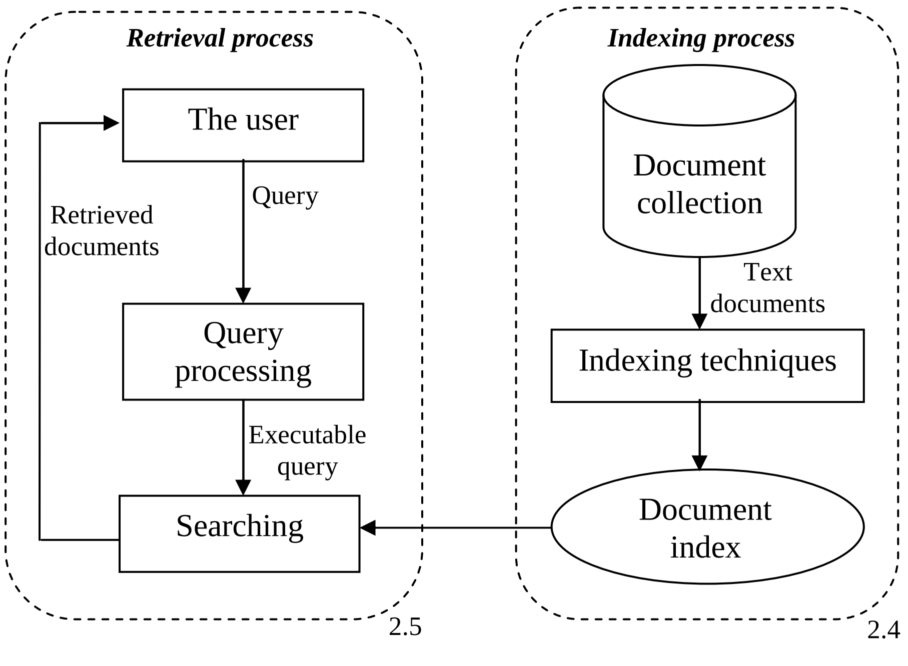


Figure 2.1. A general indexing and retrieval system (numbers beside each dash-line box indicate sections that cover corresponding topic)

From Figure 2.1 it can be seen that indexing is one of the more important processes for managing information of text documents in order to facilitate the information retrieval process. Text documents are by necessity indexed using some indexing techniques for efficient retrieval and by means of simple indexing lookup algorithms [7], [27]. The area of the search algorithms is not in the scope of this thesis. The search process mentioned in this thesis refers to simple indexing lookup algorithms. The lookup algorithm refers to the process of finding the document index when a query is presented. However, it has long been known that the advancement of Thai text indexing is challenging due to its nature of being a non-segmented language.

The Thai language belongs to a class of non-segmented languages, the sentences of which consist of a string of symbols without explicit word delimiters. Words in the Thai language are not naturally separated by any word-delimiting symbols. Due to this characteristic, indexing Thai text documents is a challenging task and has become one of the research focuses in the area of Thai information retrieval. Many techniques have been proposed to index Thai text documents, including n-gram inverted index [8], word inverted index and suffix array approaches [1].

The word inverted index approach, sometimes called word based approach, relies on an effective word segmentation technique in order to extract the indexing terms from Thai text documents [12], [2], [5]. However, most word segmentation approaches require complex language analysis and lengthy computational time. The success of a word segmentation approach relies very much on the language analysis techniques or on the use of an appropriate dictionary or corpus.

Of all the Thai text indexing techniques, word inverted index approach is the more prominent and it is language-dependent. There are some other techniques, such as n-gram inverted index [10], [8] and suffix array [1], [28], which do not rely on language analysis or on the use of a dictionary or corpus. These techniques are language-independent and are commonly used to tackle many un-delimited Asian languages.

In the following sections, overviews of Thai text indexing techniques used in the area of Thai information retrieval systems are presented. These will offer help to understand the challenges present in Thai text indexing.

2.2 Overview of Thai text indexing

Indexing Thai text documents is an essential process in Thai information retrieval [1], [2], [3]. Indexing is a process that creates the necessary data structure, known as indices, for mapping keywords (also called indexing terms) to those text documents containing the keywords. In an information retrieval system, it is necessary to index a text document first to enable efficient lookup and retrieval of the text document later. A number of data structures are used in indexing Thai text documents. They include an inverted index, also called inverted file, and suffix array. Inverted index is currently regarded as one of the better index data structures for most applications [7].

However, a challenging task for Thai text indexing is extracting the indexing terms, because Thai text documents are non-segmented. Most of the semantic indexing terms are usually carried by nouns, although a sentence in natural language text is composed of nouns, pronouns, articles, verbs, adjectives, adverbs, and connectives. Based on the frequency of words in the Thai dictionary, it was found that most Thai words are nouns [10]. However, it is still a complex task to indicate how frequently these words are used in normal Thai text documents, as the frequency of these words is dependant on the document size and domain.

In Thai text documents, the extraction of indexing terms becomes a main issue because they cannot be specified automatically from text documents, due to the nature of Thai texts being non-segmented. Although indexing terms can be manually specified by experts, this process is very time consuming and dictionaries can be costly to maintain [10], [29]. As a result, there are many challenges for indexing Thai text documents. Thai text indexing techniques can be divided into two main categories: a *language-dependent method* and a *language-independent method*. When indexing Thai text documents using a language-dependent method, a word segmentation technique [12], [18], [30], [31] is generally essential during the pre-processing stage for extracting the indexing terms before an inverted index can be constructed. This technique is known as the word inverted index. The techniques for Thai word segmentation can be broadly classified into four approaches: dictionary based [14], [13], rule based [15], [16], [11], hybrid [32], [33], and machine learning based approaches [2], [19]. These four methods are usually performed to segment Thai text documents into indexing terms before constructing an index to allow retrieval.

On the other hand, n-gram inverted index [22], [21] and suffix array [28], [34] were proposed as the alternative indexing techniques which do not require linguistic knowledge of a language. These techniques are language-independent and most widely used to tackle Asian languages, many of which are un-delimited languages. An n-gram inverted index is the main indexing method that has been widely used in many Asian language text retrieval systems [35], [36], [37], [38], [39], [40] such as Chinese, Japanese and Korean (CJK). This is because these Asian languages share similar difficulties in segmenting texts and specifying indexing terms. Therefore, this technique is acknowledged by many Asian researchers as a workable solution to information retrieval problems for many Asian languages [40], [22], [35]. Another approach is by using a data structure called suffix array for indexing Thai text documents [41], [1]. In the suffix array, a given text document is viewed as a sequence of characters that can be constructed as an array containing character occurrences without indexing term extraction requirements.

Before Thai text indexing techniques can be described in more detail, the linguistic characteristics of the Thai language are first provided in the next section to assist with understanding of the problem.

2.3 Linguistic characteristics of the Thai language

In the Thai writing system, sentences are formed as a long sequence of characters without word boundaries or sentence separators to delimit words, phrases or sentences. The spaces in the Thai language are usually used to interrupt an idea or to help the reader pay attention to the text, but they do not signify a split between words, phrases or sentences [29]. Additionally, the Thai language has no capital letters to identify proper nouns or starting points of sentences like the English language. Figure 2.2 shows an example of the Thai language.

การจะเป็นหมอนวดไทยแบบราชสำนักที่ดีต้องมีนิ้วที่มีกำลังแข็งแรง ไม่อ่อนล้าหมดกำลังไปก่อนจะทำการนวดรักษาคนไข้เสร็จสิ้น อาจารย์ณรงค์สักข์ บุญรัตนหิรัญ ได้นำมาฝึกนักศึกษาแพทย์แผนไทยแบบประยุกต์ทุกคน โดยการให้นั่งแบบง่ายๆ และแบบยากๆ คือ ก่อนที่จะฝึกทำการนวดอาจารย์ผู้สอนจำเป็นต้องให้ผู้ฝึกทำการนวดอาจารย์ผู้สอนจำเป็นต้องให้ผู้ฝึกนวดฝึกกำลังนิ้วมือก่อน ขั้นตอนอาจให้ฝึกบีบเทียนหนัก ๔ บาทให้อ่อนนิ่มแล้วจึงให้ฝึกยกกระดานในท่าเหยียดขาตรงมือโหย่งยกตัวให้ลอยขึ้นจากพื้น จากนั้นจึงฝึกยกกระดานในท่านั่งขัดสมาธิเพชร

Figure 2.2. Example of the Thai language

In the Thai character set, there are a total of 76 characters, consisting of 44 consonants, 14 vowels, four tone marks, ten Thai digits, and four special characters, as shown in Figure 2.3.

Type of Thai Characters	**Member**
Consonants	ก ข ฃ ค ฅ ฆ ง จ ฉ ช ซ ฌ ญ ฎ ฏ ฐ ฑ ฒ ณ ด ต ถ ท ธ น บ ป ผ ฝ พ ฟ ภ ม ย ร ล ว ศ ษ ส ห ฬ อ ฮ
Vowels	ะ า ิ ี ึ ื ุ ู ั ำ เ แ โ ใ ไ ฤ ฦ
Tone Marks	่ ้ ๊ ๋
Thai Digits	๑ ๒ ๓ ๔ ๕ ๖ ๗ ๘ ๙ ๐
Special Characters	็ ์ ๆ ฯ

Figure 2.3. Thai character set

Jaruskulchai [42] showed that the Thai language has similar grammatical categories to the English language in term of the parts of speech. Thai words can be classified into 14 categories: nouns, pronouns, verbs, auxiliary verbs, determiners, adjectives, adverbs, classifiers, prepositions, conjunctions, interrogatives, prefixes, suffixes, and negative verbs.

The Thai writing system exhibits SVO (Subject-Verb-Object) word-order [43], and it reads from left to right and from top to bottom. There are four levels of the appearance of Thai characters: the tone, upper vowel, middle, and lower vowel as shown in Figure 2.4. These levels can be used to indicate the character types. For instance, the middle level usually contains the consonants but sometimes contains vowels, Thai digits and special characters. The tone marks are always located in the tone level and most vowels are usually placed on the upper vowel and lower vowel levels. For other characters, they also have their specific positions. Figure 2.4 shows the position lines in order from the top level to lower levels of the phrase ‘ข้อมูลเนื้อสัตว์’ (meat information).

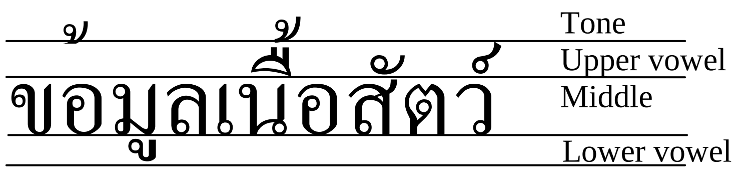


Figure 2.4. Four levels of appearance of Thai characters

According to W. Aroonmanakun [30], most Thai words are defined as a linguistic unit that is usually made up of simple words or monosyllables. However, Thai words can also be a combination of two or more morphemes, these are known as compound words. Therefore, Thai words can be viewed as a combination of syllables, and they can be distinguished into two types of words under Thai grammar rules as follows:

- **Simple words**

A simple word in the Thai language can function as a noun, verb, adjective, etcetera. It can have one or more syllables. However, in the case of a multisyllabic word, each syllable may or may not have a meaning on its own, and the meaning of the word may or may not be related to the meaning of any particular syllable. Instances of these simple words are นอน (sleep), อ่าน (read) which are monosyllabic words, and สะ-พาน (bridge), นา-ฬิ-กา (clock) which are multisyllabic words, etcetera. Here, a character ‘-’ indicates a monosyllabic point.

- **Compound words**

A compound word is formed by combining two or more simple words. A compound word can be created with many structural forms: noun + noun, noun + adjective, adjective + adjective, noun + verb, verb + verb, and so on. The meaning of the compound words may or may not be related to the meaning of any particular simple words contained in the compound word. For example, แม่น้ำ (river) is composed of แม่ (mother) and น้ำ (water); though the meaning of แม่น้ำ (river) is not 'the mother of water', it is related to water. The meaning of some compound words could be different from the meaning of their parts. As an example, ยินดี (glad) is composed of ยิน (hear) and ดี (good), หายใจ (breathe) is composed of หาย (lost) and ใจ (heart). Some compound words are created by conjoining two simple words that are quite similar in meaning. ดูแล (look after) is composed of ดู (look) and แล (see), while สวยงาม (beautiful) is composed of สวย (pretty) and งาม (beautiful). Some words are created by conjoining the same word, such as แดง ๆ (red red) or ดำ ๆ (black black) where ๆ is a symbol for duplication.

Although the criteria above seem to be clear, when looking at the real data, it is not always easy to determine the number of words in a given text. For example, หม้อหุงข้าว (rice cooker) can be analyzed as one compound word, or three simple words หม้อ (pot), หุง (cook), and ข้าว (rice). As a result, the problem of defining the word exists because of compound words that can be formed from the combination of two or more simple words. Because of these problems, Thai language processing is a challenge due to its nature of being a non-segmented language. This has called for the need to research and develop effective Thai word segmentation and Thai text indexing techniques. A review of the research in these areas will be described in following section.

2.4 Thai text indexing techniques

In the area of Thai information retrieval, research into Thai text indexing has been around for about two decades [20]. Many techniques have been proposed for solving the problems of extraction of Thai indexing terms and producing Thai text indexing algorithms. In this section, existing indexing techniques used in Thai information retrieval are discussed. Thai text indexing techniques can be divided into two main categories: a language-dependent technique and language-independent technique, and they are subdivided into three approaches: the word inverted index, the n-gram inverted index and suffix array approach as shown in Figure 2.5.

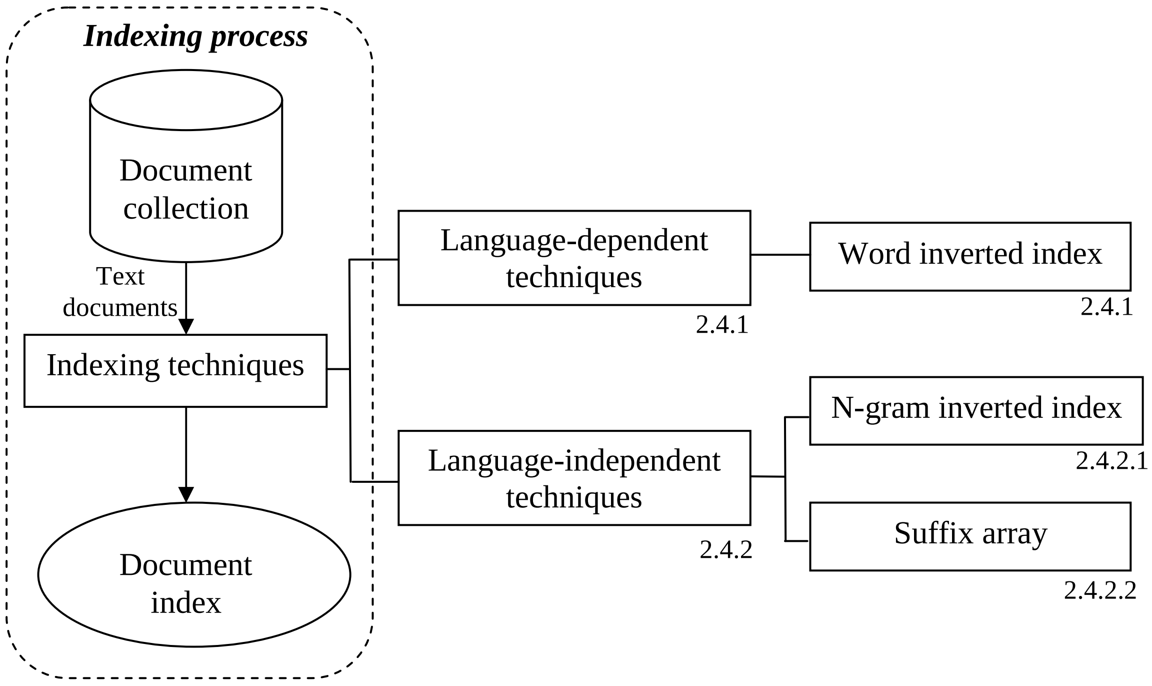


Figure 2.5. Existing Thai text indexing techniques (numbers beside each box indicate sections that cover corresponding topic)

2.4.1 Using language-dependent method for Thai text indexing

In using a language-dependent method, the prominent approach is to use the more widely adopted solution called the word inverted index [1] ,[7], [34], [44]. The word inverted index can be viewed as a word based approach which has been shown to work effectively for segmented languages such as English or other European languages. The word inverted index is a popular and important index data structure used in the area of information retrieval. It is used to speed up the process of building the indexing structure and it is efficient at the retrieval of text documents containing a query. This indexing structure usually collects indexing terms from text documents that describe the contents of the documents.

Therefore, when using the word inverted index for Thai text indexing, Thai text documents need to be parsed and tokenized into individual words. The word segmentation technique is generally an essential part in performing the indexing term extraction, before the word inverted index can be constructed.

Additionally, stopword removal is another text pre-processing task, which is also normally performed before constructing the word inverted index, in order to reduce noisy words. In Figure 2.6, Thai text indexing architecture using the word inverted index technique is depicted.

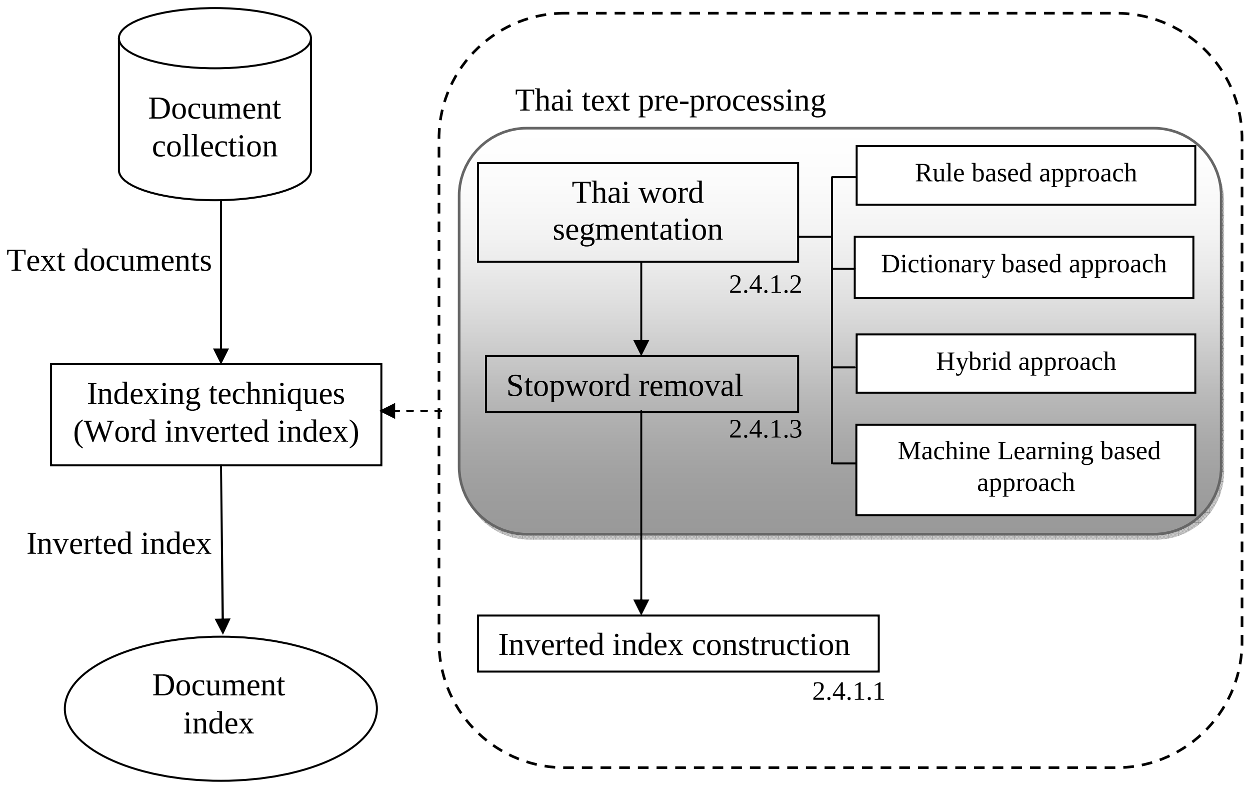
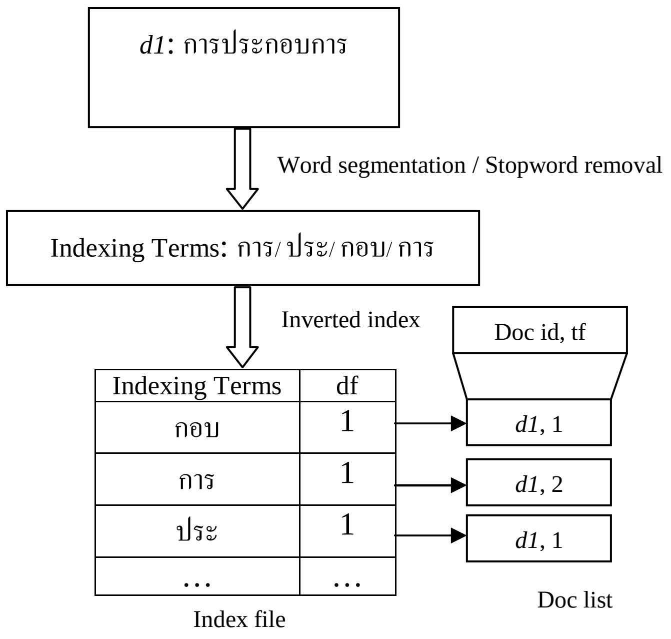


Figure 2.6. General process of indexing Thai text documents using word inverted index (numbers beside each box indicate sections that describe corresponding topic)

To illustrate the word inverted index technique in more detail, a typical process of the word inverted index approach, which is created on the document *d1* containing the string *s* 'การประกอบการ' that means 'engage in business' in English, is shown in Figure 2.7.



Where df = document frequency and tf = term frequency

Figure 2.7. Example of the word inverted index

For Thai information retrieval, after Thai text documents are segmented into a sequence of indexing terms using word segmentation and stopwords removal, all tokenized indexing terms are then stored in alphabetical order in the inverted index for fast retrieval. The inverted index is composed of two elements: the vocabulary and posting file. The vocabulary contains the set of all distinct indexing terms that occur in the documents. The posting file contains a list of pointers or indexing term positions where they appear in the text documents. The posting file also consists of the identifier of the document that contains the indexing terms and the list of the offsets where the indexing terms occur in the text document. For each indexing term *t*, there is a posting list that contains postings $< d, f, [o_1, ..., o_f] >$, where *d* is a document identifier (document ID), *f* is the frequency of the indexing term *t* in the document *d* and $[o_1, ..., o_f]$ is a list of offsets *o* that can refer to indexing term or character positions. The posting file can be used to provide faster and more accurate information retrieval. However, the size of the posting file is sometimes larger than the size of the documents since the main space used by the inverted index is for the storage of document IDs, frequency and offsets of each indexing term [45]. The storage used by the inverted index can reach up to 300 per cent of the original file size [46]. The following shows the organization of the indexing terms into the inverted index.

An example of the vocabulary and posting file of indexing terms, which are created on the document *d1* containing the string *s* ‘การประกอบการ’, can be shown in Figure 2.8.

d1:	ก	า	ร	ป	ร	ะ	ก	อ	บ	ก	า	ร
	1	2	3	4	5	6	7	8	9	10	11	12

Word segmentation

Vocabulary	**Posting file**
กอบ	<*d1*, 1, [7]>
การ	<*d1*, 2, [1, 10]>
ประ	<*d1*, 1, [4]>

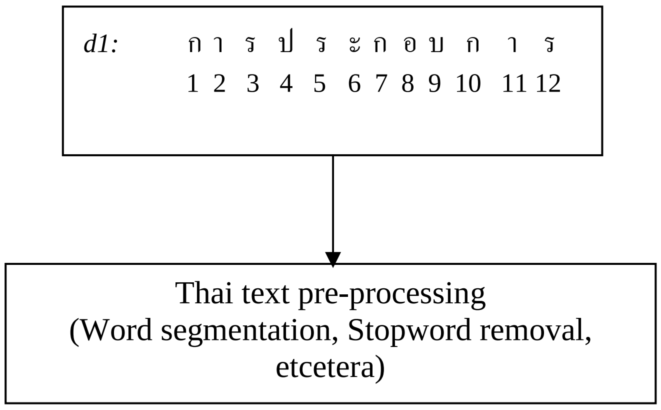
Figure 2.8. Example of vocabulary and posting file of document containing the string *s* ‘การประกอบการ’

2.4.1.1 Inverted index construction

To build the inverted index efficiently, the trie data structure [47] is employed. The trie data structure is a tree-based data structure allowing fast string matching. The trie data structure is used to perform a fast search in a large text collection, such as searching words in a large text document collection or Oxford English dictionary that contains many gigabytes of text. All indexing terms in the vocabulary are represented by paths on the trie data structure. In addition, the posting file is kept in the trie data structure as well, in order to speed up the accessing of the indexing terms. The trie data structure is built from the given string contained in the documents and the leaf nodes of each path contain the indexing terms and the lists of their positions. This structure is particularly useful for information retrieval.

In constructing the trie data structure, all indexing terms are stored by collecting one letter at a time in lexicographical order in the trie data structure. If two or more terms have the same prefix, they will be kept in the same subtree before moving to the next character. Otherwise, if the current character does not match the current nodes in the trie data structure, a new branch will be made to collect the mismatched character, and then move to the next character. In addition, at any time when the indexing terms are kept in a leaf node, the list of term positions is also shown in the leaf node on the trie data structure. Finally, these processes are repeated until the end of the vocabulary.

The illustration of building the inverted index using this process is shown in Figure 2.9.


d1: ก า ร ป ร ะ ก อ บ ก า ร
1 2 3 4 5 6 7 8 9 10 11 12
Thai text pre-processing
(Word segmentation, Stopword removal, etcetera)

Inverted index

Vocabulary	**Posting file**
กอบ	<*d1*, 1, [7]>
การ	<*d1*, 2, [1, 10]>
ประ	<*d1*, 1, [4]>

A word inverted index construction using the trie data structure

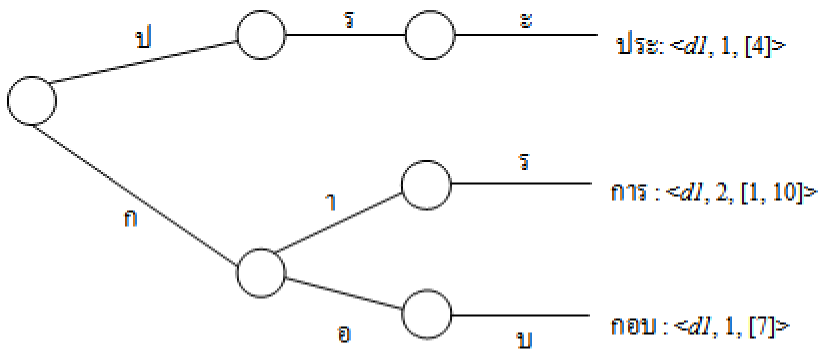


Figure. 2.9. Illustration of building word inverted index for documents containing the string *s* 'การประกอบการ'

Although it is simple to store the inverted index using the trie data structure, which can be done quickly by using tree traversing techniques, the disadvantages of this method are the storage overhead and the high cost of updating and reorganizing the index [45], [7].

2.4.1.2 Thai word segmentation

As mentioned in the previous sections, the Thai language is a string of symbols without explicit word boundary delimiters. Due to this reason, Thai word segmentation is first performed to extract the indexing terms from the text documents before these indexing terms are indexed for retrieval. Thai word segmentation can be easy for Thai people, who understand the Thai language well, to use their own judgment to analyze the text in Thai contexts. In contrast, it is complicated for computer systems to perform the word segmentation successfully in different Thai contexts. In the following section, the segmentation problems will be described.

Segmentation problems

Despite much research, automatic Thai word segmentation still remains an unsolved problem. Unlike English, Thai does not use explicit symbols for word boundaries (as discussed in Section 2.3). In addition, the existence of compound words makes the segmentation task even harder, because there is usually more than one way to insert word separators, which can lead to segmentation ambiguity. From the literature review, there are two main problems for segmenting Thai texts: segmentation ambiguity and unknown words [8], [30].

- **Segmentation ambiguity**

Although many methods can be employed to segment Thai texts into linguistic units, the same text can be segmented in many different ways, leading to segmentation ambiguity. This ambiguity happens when two or more phrases or sentences are spelled alike but have different meanings or pronunciation.

Many Thai words are also compound words consisting of multi-syllables and two or more simple words, making it possible to have different segmentations for the same compound word. Figure 2.10 shows different ways of inserting word separators in terms of computer systems.

Thai and English texts	Different ways to insert word separators	
	Thai	English
‘ฉันมารอกราบพระสงฆ์’	ฉัน|มาร|อก|ราบ|พระสงฆ์	I|come|devil|breast|flat|monk
	ฉัน|มา|รอก|ราบ|พระสงฆ์	I|come|jack|flat|monk
‘I come to wait to worship a monk’	ฉัน|มา|รอ|กราบ|พระสงฆ์	I|come|wait|worship **(Correct segmentation)**
‘เรือโคลงเพราะโคลงเรือ’	เรือ|โคลง|เพราะ|โคลง|เรือ	Boat|roll|because|roll|boat
	เรือ|โคลง|เพราะ|โค|ลง|เรือ	Boat|roll|because|cow|get down|boat
‘A boat is rolled because a cow get down to the boat’	เรือ|โค|ลง|เพราะ|โคลง|เรือ	Boat|cow|get down|because|roll|boat
	เรือ|โค|ลง|เพราะ|โค|ลง|เรือ	Boat|cow|get down|because|cow|get down|boat **(Correct segmentation)**
‘คนตากลมนอนตากลม’	คน|ตาก|ลม|นอน|ตาก|ลม	Human|expose|wind|sleep|expose|wind
	คน|ตาก|ลม|นอน|ตา|กลม	Human|expose|wind|sleep|round|eye
‘A round eye human sleep and expose wind’	คน|ตา|กลม|นอน|ตาก|ลม	Human|eye|round|sleep|expose|wind
	คน|ตา|กลม|นอน|ตา|กลม	Human|eye|round|sleep|round|eye **(Correct segmentation)**

Figure 2.10. Different ways to insert word separators in Thai texts

Another problem is that most Thai words have a variety of meanings, which is known as polysemy. Polysemy refers to a set of words that are spelled alike but each belongs to a different grammatical category or meaning. For example, the word ‘ตา’ as shown in Figure 2.15 can mean ‘eyes’, ‘a grandfather’ or ‘a turn’. Also the word ‘ที่’, which occurs frequently in Thai texts, has a variety of meanings such as ‘place’ or ‘that’ and can belong to different grammatical categories, such as relative pronoun, locative preposition, quantification marker or partitive. The meanings and grammatical categories of these words depend on the contexts.

- **Unknown word problem**

Another difficulty of segmentation is the existence of unknown words in the texts. This is a serious problem that occurs when the dictionary based approach is used to perform segmentation. Thai texts usually contain unknown words such as person or place names. This makes Thai word segmentation more complicated because the Thai language does not use capital letters to indicate proper nouns as mentioned in Section 2.3. The unknown words [9] in the Thai language can be classified into five categories: proper noun, loan word, acronym, foreign word, mistype and official places. These words cause mistakes in segmenting Thai texts.

Thai word segmentation techniques

Due to the segmentation problems, passing human knowledge to computers is very important so that computers can understand the Thai language like humans do. As a result, there are various Thai word segmentation approaches proposed by numerous researchers since 1981 [5]. Previously proposed methods for Thai word segmentation can be classified into four main categories.

The following sections provide several Thai word segmentation approaches for solving Thai word segmentation problems.

- **Rule based approach**

Thai words are based on the relationships between the consonants, vowels, tone markers and symbols that can be formed by using basic Thai word formation rules. As the rule based system can incorporate Thai word formation rules, a rule based approach has been used for word segmentation. It also uses the rule of syllable combination for segmentation. Like many languages, Thai is one language that has basic word formation rules, and these rules can be expressed using the Backus-Naur Form (BNF). Figure 2.11 shows the BNF of Thai word formation rules using a character encoding method. The character encoding method is a process of using characters to represent the basic word formation rules and the combination of these characters is used to describe a word characteristic. In BNF, Thai characters can be classified into the five sets: consonant (C_i), vowel (V_i), tone marks (T), numeral (N), and special symbol (S). Based on these character sets and their positions, the syllabic boundary’s rules can be formulated to extract words from sentences.

$w = (V_i)\ C \mid C_d\ (V_2)\ (T)\ (C_f)$

Where $C = \{ c_i \mid c_i =$ consonant, $1 \leq \mid \leq 44\}$
$V_1 = \{ v_i \mid v_i =$ vowels are often placed at the beginning of syllable, เ, แ, ไ, ใ, โ }
$V_2 = \{ v_i \mid v_i =$ vowels which are always placed after consonants}
$V_2 = \{$ ะ, า, ิ, ี, ึ, ื, ุ , ู}
$C_d = \{ c_ic_j \mid c_i \in C$ and c_j are double consonants, $c_j =$ ง, ญ, ณ, น, ม, ย, ร, ล, ว, ฬ}
$C_f = \{ c_i \mid c_i =$ syllable-final consonants}
$T = (t_i \mid t_i$ is', ้, ๊, ๋ and ์ }

Figure 2.11. BNF of Thai word formation rules

Due to having word formation rules for the Thai language, there are a number of different approaches that have been proposed for segmenting Thai texts based on Thai word formation rules, such as Thai Character Cluster (TCC) [11] and Thai Syllable Separation Algorithms [16]. Some of this research still cannot fully resolve the problems of segmentation, as these approaches segment Thai texts into partial words rather than whole words. The following sections summarize some approaches with the problems of segmentation in this area.

The earliest technique for the rule based approach was introduced by Thairatananond in 1981[15]. This technique uses the rule of syllable combination in segmentation. The characters can be divided into five categories as shown in Table 2.1.

Table 2.1. Types of Thai characters

Tag	Types of Thai characters	Example
S	Character that can be the final consonant in a word	ก ข ฃ ค ฆ ง จ ช ซ ญ ฎ ฏ ฐ ฑ ฒ ณ ด ต ถ ท ธ น บ ป พ ฟ ภ ม ย ร ล ว ศ ษ ส ฬ อ
V	Vowel	ะ า ิ ี ึ ื ุ ู ั ำ เ แ โ ใ ไ ฤ ฦ
C	Character that cannot be the final consonant in a word	ฅ ฉ ผ ฝ ฌ ห ฮ
T	Tonal character	่ ้ ๊ ๋
G	Symbol	์ ๆ ฯ

For example:

สิ้น CTVS

ศิลป์ CVSSG

กวาง CCVS

The advantage of this technique is simple as the segmentation is easily realized by applying the rule of syllable combination. However, this technique is not flexible enough to handle Thai characters because some Thai characters can be both C and S, which sometimes makes the segmentation inaccurate.

In 1983, Charnyapornpong introduced a Thai Syllable Separation Algorithm [16]. This method was used in the early development of the Thai word segmentation system. For this method, the system checks the rules of Thai characters and spaces that can sometimes be used to represent the beginning of new paragraph, to specify the word boundary, and the word boundary is divided into two groups: front boundary recognition rule and tail boundary recognition rule. The rule for characters specifies probability of word segmentation in the position of that character. This method divides character into five groups as follow:

1. Non-spaced characters such as ั, ิ, ุ , ๊, ่ and ์
2. Leader characters such as เ, แ, โ, ใ and ไ
3. Follower characters such as ะ, า and ำ
4. The mark placed over the final consonant of a word in the Thai language to indicate that it is mute character, which is ์
5. Remaining characters

The disadvantage of this rule based approach is that its segmentation accuracy is low when compared to the segmentation results that are manually done by Thai experts, and it requires hand-crafted rules. The segmentation accuracy can be evaluated using measurements in the proportion of corrected segmented words and total segmented words extracted by segmentation techniques.

Another method for the rule based approach was introduced by Theeramunkong, Tanhermhong, Chinnan and Sornlertlamvanich [11]. They proposed the method to segment a text into Thai character clusters (TCCs) which are units smaller than a word but larger than a character. This is because some contiguous characters tend to be an inseparable unit in the Thai language, called a Thai character cluster (TCC). Unlike word segmentation, which is a difficult task, segmenting a text into TCCs is easily realized by applying a set of rules based on the type of character. Therefore, this method needs no dictionary.

The segmentation ability of this process is 100 percent, in the sense that it is not possible for these units to be divided into two or more units, which are substrings in two or more different words [5]. This process can be implemented without a dictionary, but uses a set of simple linguistic rules based on the types of Thai characters as shown in Figure 2.12.

Types of Thai characters	**Example**
Consonants	ก ข ฃ ค ฅ ฆ ง จ ฉ ช ซ ฌ ญ ฎ ฏ ฐ ฑ ฒ ณ ด ต ถ ท ธ น บ ป ผ ฝ พ ฟ ภ ม ย ร ล ว ศ ษ ส ห ฬ อ ฮ
Upper vowel	ื ั ็ ิ ี ึ ำ
Lower vowel	ุ ู
Front vowel	เ แ โ ใ ไ
Rear vowel	า ำ ะ ๅ ฯ

Figure 2.12. Types of Thai characters

In terms of a set of linguistic rules, the composition of TCC is unambiguous and can be defined by a set of rules. For example, a front vowel and the next character, a consonant, have to be grouped into the same unit. A tone mark is always located above a consonant and cannot be separated from the consonant. A rear vowel and the previous character have to be grouped into the same unit.

As the first step in the segmentation approach, a set of rules is applied to a group of close characters in the document to form TCCs. The rules of segmentation into TCCs can be represented in EBNF form as shown in Figure 2.13.

<TCC> →	‘ก็’ | ‘อึ’ | ‘หึ’ | <Cons> ‘รร’ ,<Cons> ‘ ์ ’ | <Cons> <BCons> <Cons> ‘ ์ ’ | <Cons> <TCC1> <Karan> | <FVowel><Cons> <TCC2> <Karan>
<TCC2> →	<Cons> ‘ าะ ’ | ‘ ึ ’ <BCons> | <UVowel>{<Tone>}<BCons> [‘ า ’|‘ ะ ’] | {<Tone>} [‘ า ’|‘ าะ ’|‘ ะ ’]
<TCC1> →	<DVowel> {<Tone>} | {<Tone>} ‘ ำ ’ | [‘ ี ’|‘ ื ’] {<Tone>}<BCons> | ‘ ั ’{<Tone> [‘ ุ ’|‘ ิ ’]} | ‘ ึ ’ <BCons> | <Tone> [<TVowel>|<DVowel>] {‘ ว ’}<BCons> | ‘ ิ ’ {<Tone>{<BCons>}} | ‘ ี ’ <Tone> | {<Tone>} <BVowel> | NULL
<Karan> →	<Cons>{<Cons>}{[<DVowel> | ‘ ิ ’]}‘ ์ ’ | NULL

Figure 2.13. All rules for TCC Segmentation

Figure 2.13 shows all rules used for TCC segmentation. Here, <FVowel>, <TVowel>, <UVowel>, and <DVowel> represent a front vowel, a rear vowel, an upper vowel, and a lower vowel, respectively. <Cons> is a consonant that can be the first consonant of a word, <BCons> is a consonant that can be the second consonant of a word, <Tone> is a tone mark, <Karan> is a special character named ‘karan’ that is the mute character.

Figure 2.14 shows a fragment of a text segmented into TCCs by the proposed method and its correct word segmentation. Here, a character '-' indicates a segmentation point. The corpus where characters are grouped into TCCs is called a TCC corpus.

String	การเก็บภาษีประเทศไทยและประเทศ
Meaning	Collecting tax in Thailand and other countries
Correct segmentation	การ-เก็บ-ภาษี-ประเทศ-ไทย-และ-ประเทศ
Thai character cluster (TCC)	กา-ร-เก็บ-ภา-ษี-ป-ระ-เท-ศ-ไท-ย-และ-ป-ระ-เท-ศ

Figure 2.14. Example of Thai character cluster compared with correct segmentation

Although, the rule based approach seems to be able to deal with the problems of Thai texts because it applies the set of Thai grammar rules to split the Thai texts into linguistic units, there are still many disadvantages to using this method. One of the disadvantages is that its segmentation accuracy is low when compared to human segmentation results. Besides this, it requires hand-crafted rules.

- **Dictionary based approach**

Another approach for Thai word segmentation is the dictionary based approach. This approach uses a set of all possible words in the dictionary, denoted as *Dic*, to match an input text for segmenting process. The input text is parsed and segmented into words that are appropriate for the indexing process. A number of methods were proposed to use electronic dictionaries to divide the text into individual words. The first research that works for Thai word segmentation was presented by Poowarawan in 1986 [13]. This research introduced the 'longest matching algorithm for the dictionary based approach' to perform word segmentation for Thai language. It was reported [13] that this method scans a whole document from left to right, and selects the longest match with the set of words in a dictionary entry at each point.

For example, in string *s* = 'เจมส์เป็นเด็กตากลมต้องการนำโคลงเรือ' (James is the round eye kid that wants to bring cows down to a ship), the longest matching algorithm searches string *s* against the dictionary, based on the longest matching vocabulary. In case the longest match is found, the words will be identified within word boundaries, otherwise the right character will be removed one character at a time until the longest match is found in the dictionary. This process is performed continually until the end of the string that is shown in Figure 2.15.

Remainder of string *s*	**Longest words**
เจมส์เป็นเด็กตากลมที่ต้องการนำโคลงเรือ	เจ
จมส์เป็นเด็กตากลมที่ต้องการนำโคลงเรือ	จม
เป็นเด็กตากลมที่ต้องการนำโคลงเรือ	เป็น
เด็กตากลมที่ต้องการนำโคลงเรือ	เด็ก
ตากลมที่ต้องการนำโคลงเรือ	ตาก
ลมที่ต้องการนำโคลงเรือ	ลม
ที่ต้องการนำโคลงเรือ	ที่
ต้องการนำโคลงเรือ	ต้องการ
นำโคลงเรือ	นำ
โคลงเรือ	โคลง
เรือ	เรือ

Figure 2.15. Result of longest match for string *s* 'เจมส์เป็นเด็กตากลมที่ต้องการนำโคลงเรือ'

Obviously, this algorithm will fail to find the correct segmentation in many cases because it selects the longest match within the dictionary, also called greedy characteristic [17]. For instance, according to the above, string *s* ‘เจมส์เป็นเด็กตากลมที่ต้องการนำโคลงเรือ’ (James is the round eye kid that wants to bring cows down to a ship) will be incorrectly segmented using the longest matching algorithm as เจ (vegetarian) จม (sink) เป็น (be) เด็ก (kid) ตาก (expose) ลม (wind) ที่ (place) ต้องการ (want) นำ (bring) โคลง (roll) เรือ (ship), while the correct segmentation, which cannot be found by this algorithm, is เจมส์ (James) เป็น (be) เด็ก (kid) ตา (eye) กลม (round) ที่ (that) ต้องการ (want) นำ (bring) โค (cows) ลง (down) เรือ (ship). When compared to the correct segmentation, it can be observed that the longest matching algorithm may not provide the segmentation of words that reflects the meaning of the context. For example, one part of the above string: ตากลม (round eyes) will be incorrectly segmented as: ตาก (expose) ลม (wind), while the correct one is ตา (eye) กลม (round).

Notice that the longest matching algorithm can segment a string of characters into words correctly in about 80 per cent of all cases [14], [10]. For example, ‘การนวดแผนไทยเป็นการกายภาพบำบัดเพื่อรักษาคนไข้อย่างหนึ่ง’ (Thai traditional massage is one kind of physical therapy for treatment of patients) can be segmented into ‘การนวด-แผน-ไทย-เป็น-การกายภาพ-บำบัด-เพื่อ-รักษา-คนไข้-อย่างหนึ่ง’ as shown in Figure 2.16, but the longest matching algorithm cannot extract the keywords such as ‘แผนไทย’ (Thai traditional) or

'กายภาพบำบัด' (physical therapy) from the string. Consequently, this algorithm loses the keywords due to its incorrect segmentation. This is one of the drawbacks that appears when the longest matching algorithm discriminates this kind of ambiguous text incorrectly [18].

String	การนวดแผนไทยเป็นการกายภาพบำบัดเพื่อรักษาคนไข้อย่างหนึ่ง
Segmented string Using longest match	การนวด-แผน-ไทย-เป็น-การกายภาพ-บำบัด-เพื่อ-รักษา-คนไข้-อย่างหนึ่ง
Translate word by word	Massage-plan-Thai-be-physical-assuage-for-treatment-patient-kind of
Correct segmentation	การนวดแผนไทย-เป็น-การกายภาพบำบัด-เพื่อ-รักษา-คนไข้-อย่างหนึ่ง
Meaning	Thai traditional massage is one kind of physical therapy for treatment of patients

Figure 2.16. Segmentation of: 'การนวดแผนไทยเป็นการกายภาพบำบัดเพื่อรักษาคนไข้อย่างหนึ่ง'

In order to solve this problem, the maximum matching algorithm was proposed. According to Sornlertlamvanich, he presented the maximum matching algorithm in 1993 [14]. This algorithm was proposed to solve the problem of the longest matching algorithm. This method is sometimes called the backtracking algorithm, which also uses matching against the dictionary. The maximum matching algorithm is different from the longest matching algorithm, although both techniques use matching against the dictionary. The longest matching algorithm maps written text from left to right, and gives first priority to the longest candidate found in the dictionary, while the maximum matching algorithm creates all possible word segmentation candidates and selects the one that contains the fewest words.

In the first step of the maximum matching algorithm, this method generates all possible segmentations for the text based on a word set until there are no more words that can be segmented. The algorithm will then select the segmentation path with the lowest number of segmented words [12], [3]. This method can be performed efficiently by using dynamic programming techniques, because the method actually finds the words that have maximum matches instead of using heuristics to guess. This method always outperforms the longest matching method [5]. Figure 2.17 demonstrates the procedure of the maximum matching algorithm in processing the sentence 'การนวดแผนไทยเป็นการกายภาพบำบัดเพื่อรักษาคนไข้อย่างหนึ่ง**'** **(**Thai traditional massage is one kind of physical therapy for treatment patients).

	All possible segmentations for the sentence	**Number of words**
1	การนวด-แผนไทย-เป็น-การกายภาพบำบัด-เพื่อ-รักษา-คนไข้-อย่างหนึ่ง	8
2	การนวด-แผนไทย-เป็นการ-กายภาพ-บำบัด-เพื่อ-รักษา-คนไข้-อย่างหนึ่ง	9
3	การนวด-แผนไทย-เป็นการ-กายภาพ-บำบัด-เพื่อ-รักษา-คน-ไข้-อย่าง-หนึ่ง	11
4	การนวด-แผน-ไทย-เป็นการ-กาย-ภาพ-บำบัด-เพื่อรักษา-คนไข้-อย่างหนึ่ง	10
5	การนวด-แผน-ไทย-เป็นการ-กายภาพ-บำบัด-เพื่อรักษา-คนไข้-อย่างหนึ่ง	9
6	การนวด-แผน-ไทย-เป็นการ-กายภาพบำบัด-เพื่อรักษา-คนไข้-อย่าง-หนึ่ง	9
7	การนวด-แผน-ไทย-เป็นการ-กาย-ภาพ-บำบัด-เพื่อ-รักษา-คนไข้-อย่างหนึ่ง	11
8	การนวด-แผน-ไทย-เป็นการ-กาย-ภาพ-บำบัด-เพื่อ-รักษา-คน-ไข้อย่าง-หนึ่ง	12
9	การ-นวด-แผน-ไทย-เป็น-การ-กาย-ภาพ-บำบัด-เพื่อ-รักษา-คน-ไข้-อย่าง-หนึ่ง	15
10	การ-นวด-แผน-ไทย-เป็น-การกายภาพบำบัด-เพื่อ-รักษา-คนไข้-อย่างหนึ่ง	10

Figure 2.17 Illustration of maximum matching algorithm procedure

From Figure 2.17, the algorithm can generate ten possible segmentations for the sentence and the lowest number of words is eight at the first segmentation path. This path is the result of using the maximum matching algorithm containing keywords such as 'แผนไทย' (Thai traditional) or 'กายภาพบำบัด' (physical therapy).

Nevertheless, when the alternatives have the same number of words, the algorithm cannot determine which one is the best candidate. In this case, some other heuristics have to be applied to make the final decision. The heuristic that is often used is the longest matching pattern-word at each point.

The main advantages of the dictionary based approach are simple and straightforward [12]. However, the performance of the dictionary based approach depends critically on the quality and size of word sets in the dictionary used during the segmentation process, although the success rate of using the dictionary based approach in segmentation is quite high [2]. This implies that different dictionaries can provide different results. The most often used Thai dictionary is the Thai Royal Institute word dictionary [Pochananukorm Chabab Rachabandictayasathan B.E.2525], which was compiled by Virach and consisted of 32,558 words. The disadvantage of the dictionary based approach is that this technique may not perform well when handling unknown words and proper nouns, because these words cannot be found in the dictionary and the Thai language does not use capital letters for proper nouns. So, success relies on the dictionary used for segmentation [48], [4]. At present, many new Thai words have been generated through the transliteration of foreign words. Consequently, the process of the dictionary based approach has become inefficient, because this approach requires the manual addition of new words in order to improve its performance [2]. Also, the second problem is parsing ambiguity. Ambiguity occurs when there is more than one way to segment a given character sequence as described in the segmentation problems section.

- **Hybrid approach**

As dictionary based and rule based approaches have their own advantages, a number of problems in Thai word segmentation have been solved by a combination of the two approaches, known as the Hybrid approach. According to Kawtrakul and his colleagues [32], they presented the n-gram Markov model with *n* equal to three. They also included a solution for spell checking and word filtering. The first step in word segmentation was based on word formation rules and a dictionary lookup. They follow by using a Markov model for the word filtering process, which solves word boundary and tagging ambiguities.

- **Machine learning based approach**

The recently proposed method is the machine learning based approach [2], [19]. The machine learning based approach can be divided into four main algorithms: Naïve Bayes, decision tree, Support Vector Machine, and Conditional Random Field. These algorithms will be described below.

The Naive Bayes algorithm was first proposed and used for text categorization tasks by D. Lewis in 1998 [49]. Naive Bayes is based on Bayes' Theorem in the probabilistic framework. The basic idea is to use the joint probabilities of words and categories to estimate probabilities of the categories that best describe the document. The Naïve Bayes algorithm makes the assumption of word independence, (i.e., the conditional probability of a word given a category, is assumed to be independent from the conditional probabilities of other words given that category.)

The decision tree algorithm was first proposed by J. R. Quinlan and the well-known version of the algorithm is called C4.5 [50]. A decision tree is a simple structure where non-leaf nodes represent the conditional tests of attributes or features and leaf nodes contain the class label.

Support Vector Machines is the machine learning algorithm introduced by Vapnik [51]. Support Vector Machines was first applied for text categorization tasks by Jaochim [52]. This algorithm is based on the Structural Risk Minimization principle with the error-bound analysis. The method is defined over a vector space where the problem is to find a decision surface.

Recent work was presented by Kruengkrai in 2006 [19]. He used the Conditional Random Field algorithm (CRF) for building a word segmentation model. The Conditional Random Field algorithm is a novel approach which has been shown to perform better than other machine learning algorithms for the task of labeling and segmenting sequence data [53], [54]. This work focused mainly on solving the problem of ambiguity in word segmentation. Two path selection schemes based on confidence estimation and Viterbi were proposed. The feature set used in their model required the part-of-speech (POS) tagged corpus. The POS tagged corpus is a corpus where texts are manually segmented into sentences and words, and each word is tagged with its POS [4], [55]. Therefore, if the POS tagging is inaccurate, the performance of the word segmentation could be affected.

For the Thai language, the conditional random field algorithm is a recent approach, which has been shown to perform better than other machine learning algorithms for the task of labeling and segmenting Thai texts [12], [19]. This technique uses the machine learning technique to learn from a Thai text corpus. As learning is the essential part of this approach, it is necessary to have an appropriate word-segmented Thai text corpus, called the ORCHID corpus. The ORCHID corpus in 2009 consisted of 113,404 manually tagged words since 1996 [12], [55]. The machine learning technique aims to address the drawbacks of the dictionary based approach. Using a tagged corpus in which word boundaries are explicitly marked with special annotations, the machine learning algorithm builds statistical models based on the features of the characters surrounding the boundaries. The most common features used for Thai word segmentation models are the identities and categories of characters within a defined n-gram of characters surrounding a word boundary candidate. Character types can be quite diagnostic when used for word segmentation. For example, certain leading vowels often appear at the beginning of a word, whereas tone marking characters can never begin a word. In machine learning approaches, a binary classification task is used to assign each character in the string. Each character in the Thai language can be tagged into one of two classes. The first class is the character that is placed at the beginning of the word, called word-beginning that can be labeled as class ‘B’. The second class is the character that is placed inside the word, called intra-word character that can be labeled as class ‘I’. Figure 2.18 shows an example of a string where each character is tagged with ‘B’ or ‘I’. By using the tagged corpus in which word boundaries are explicitly marked with a special character, the conditional random fields algorithm can be applied to train a model based on the features surrounding these boundaries [12], [2], [19]. For the Thai language, characters can be distinguished for segmentation tasks into ten different types as shown in Figure 2.19.

ห	n	B
ม	c	I
อ	c	I
ร	c	B
ั	t	I
ก	c	I
ษ	c	I
า	v	I
ค	c	B
น	c	I
ไ	w	I
ข	c	I
้	t	I

Figure 2.18. Example of string of characters tagged as word-beginning (B) or intra-word (I) characters

The set of character types is designed based on Thai linguistic knowledge. Machine learning algorithms can build classification models by extracting patterns of character types. For example, the conditional random field algorithm could learn that the character which is a vowel by type is most likely to begin a word, (i.e. class B). The final feature set for constructing a model is the n-gram of characters preceding and following the word boundary with their character types.

Tag	Type	Example
c	Characters that can be the final consonant in a word	ก ข ฃ ค ฆ ง จ ช ซ ญ ฎ ฏ ฐ ฑ ฒ ณ ด ต ถ ท ธ น บ ป พ ฟ ภ ม ย ร ล ว ศ ษ ส ฬ อ
n	Characters that cannot be the final consonant in a word	ฅ ฉ ผ ฝ ฌ ห ฮ
w	Vowels that can begin a word	เ แ โ ใ ไ ฤ ฦ
v	Vowels that cannot begin a word	ะ า ิ ี ึ ื ุ ู ั ำ
t	Tonal characters	่ ้ ๊ ๋
s	Symbols	์ ๆ ฯ
d	Digit characters	0-9
q	Quote characters	‘_‘ ‘_’
p	Space character inside a word	_
o	Other characters	A-Z

Figure 2.19. Character types for building a feature set used by machine learning approach

The main advantage of the machine learning approach is that it does not require the use of a dictionary. However, it still requires the use of an appropriate corpus. In this technique, the unknown word and ambiguity problems are handled in principle by extracting sufficiently rich contextual information from the n-gram, and by providing a sufficiently large set of training examples to enable accurate classification. Therefore, the weakness of the approach is that its performance depends critically on the characteristics of the document domain and the size of the training corpus. For example, if a model is constructed based on a corpus from a specific domain, it might not perform well on documents from other domains. In addition, the learning process is time consuming. The unknown word and ambiguity problems are handled by the algorithm which learns various character patterns inside a tagged corpus.

2.4.1.3 Thai stopword removal

Stopwords or noise words are those terms that frequently occur in text documents, but are not significant in representing the main content of the documents, and also they are generally not searched due to the fact that they are useless terms. Many languages such as the English language have their own stopwords in order to form grammatical constructions. In the English language, examples of such words are as follows: ‘a’, ‘about’, ‘an’, ‘are’, ‘as’, ‘at’, ‘be’, ‘by’, ‘for’, ‘from’, ‘how’, ‘in’, ‘is’, ‘of’, ‘on’, ‘or’, ‘that’, ‘the’, ‘these’, ‘this’, ‘to’, ‘was’, ‘what’, ‘when’, ‘where’, ‘who’, ‘will’, and ‘with’.

Using stopwords to index documents can be viewed as creating noise in the indexing terms, because these stopwords have no ability to identify the document. It may affect the document search by retrieving irrelevant documents and generating too much noise for the retrieval task. Thus, removing stopwords may decrease the noise in the information retrieval system.

Like the English language, the Thai language also has its own stopwords. The following table, Table 2.2, shows the list of Thai stopwords [10].

Table 2.2. Thai stopwords list from morphology

Thai Stopwords from Morphology				
ที่	ใน	ว่า	และ	จะ
ทาง	กล่าว	โดย	ซึ่ง	ต้อง
อยู่	ออก	นั้น	หรือ	เมื่อ
ใหม่	ก่อน	จึง	หาก	แก่
มี	ได้	ของ	ให้	เป็น
จำ	ก็	แต่	ยัง	ขึ้น

Thai Stopwords from Morphology				
ขณะ	เปิด	แห่ง	ร่วม	เพราะ
เช่น	ทุก	ไว้	บาง	เพียง
นี้	ไม่	จาก	ไป	มา
อย่าง	ทั้ง	เพื่อ	เช้า	แล้ว
ไร	กว่า	มาก	ด้าน	นอก
พร้อม	ใต้	ดู	อาจ	หลาย
ด้วย				

In fact, around 30 percent of the words in Thai text document collections are typically of little use in information retrieval [10]. Therefore, these stopwords are assumed not to carry any important information. They are usually ignored and not indexed in order to save storage space in the inverted index and increase retrieval speed. In stopwords removal, each word is checked against a pre-defined list of non-semantic bearing, high-frequency words, and then stopwords are removed from the index during the pre-processing step. However, the ignored stopwords are replaced by a placeholder so as to remember the offset at which stopwords occur. This technique enables searching for phrases that contain stopwords. One of the advantages of eliminating the stopwords is reducing the size of the inverted index. The size of the indexing structure can be reduced by up to 33 percent or more from a total index with the elimination of stopwords [10]. Although stopwords removal is able to improve the efficiency of indexing, it sometimes reduces the recall ability in cases where users prefer to exactly match the user's query with documents. For instance, consider a user who is looking for documents containing the phrase 'โดยลบและตัดคำเพื่อทำดัชนี' (stopwords removal and word segmentation for indexing). After performing word segmentation and stopwords removal, the indexer may keep only the terms 'ลบ' (removal), 'ตัด' (segmentation), 'คำ'

(word), ‘ทำดัชนี’ (indexing) in the inverted index. This makes retrieval and proper recognition of the documents which contain the phrase ‘โดยลบและตัดคำเพื่อทำดัชนี’ (stopwords removal and word segmentation for indexing) almost impossible. This is one important reason that can lead to the adoption of a full text search or other full text indexing methods in order to support exact matches for Thai texts [56].

Further discussions on the advantages and disadvantages of using language-dependent methods will be provided in Section 2.6.

2.4.2 Using language-independent method for Thai text indexing

In this section, the details of two approaches for indexing Thai text documents using language-independent techniques: an n-gram inverted index and suffix array techniques are reviewed.

2.4.2.1 An n-gram inverted index

Besides the word inverted index as described in Section 2.4.1, an n-gram inverted index is one of many indexing techniques that uses n-gram terms as an indexing term [22], [21], [10]. This method can be viewed as an n-gram based approach and was first introduced and tested as indexed terms by Adams in 1991 [57]. The n-gram inverted index is a language-independent approach, which does not require the use of language analysis, or a dictionary or corpus. In information retrieval systems, the n-gram inverted index is often used for Asian languages where extraction of words is not simple [58]. The n-gram inverted index has been acknowledged as a viable solution for indexing non-segmented languages such as Chinese, Japanese and Korean (CJK) [22], [21]. It is also used for other non-segmented text in the area of bioinformatics [25], [59]. This method has been experimented with in many related IR fields [23], [24]. There are a number of techniques implemented in Chinese, Japanese and Korean information retrieval systems. It is also used as a possible solution for Thai in the same way as Chinese, Japanese and Korean.

Although the Thai language is as linear as other Asian languages such as Chinese or Japanese and the same n-gram inverted index used in Chinese and Japanese can be used for Thai, the parameters for the n-gram method should be adjusted to be appropriate for the Thai language. Determining the dimensions of the gram term for using the n-gram inverted index should be considered so that they are appropriate for each application. Before the detail of the n-gram inverted index is described, the general process of the n-gram inverted indexing technique is first provided in Figure 2.20.

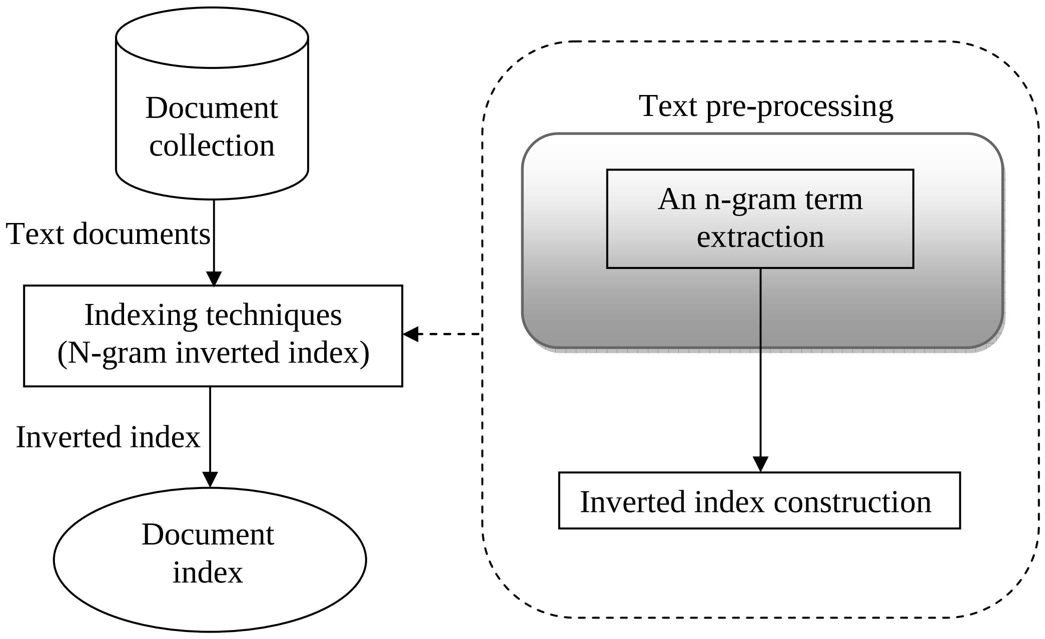

Document collection
Text documents
Indexing techniques (N-gram inverted index)
Inverted index
Document index
Text pre-processing
An n-gram term extraction
Inverted index construction

Figure 2.20 General process of indexing Thai text documents using n-gram inverted index

An n-gram term extraction

In order for the n-gram inverted index to be successful, n-gram term extraction is essential and it has to be done before indexing can be performed. Selection of the dimension of the gram term is important for these languages. For instance, it has been shown that the bi-gram term is effective for indexing Chinese text documents [60], [61], [62], [63]. Furthermore, most Chinese bi-gram terms do not lose the semantics of words. In Japanese, the dimension of the gram term had also been found to be equal to two [64]. In bioinformatics, CAFE [25] is a well known method which uses the n-gram base approach. It uses 9-gram terms for the genome sequence and 3-gram terms for the protein sequence as indexing terms.

Meanwhile the n-gram based approach with *n* equal to three and four characters seems to have the best parameters to achieve retrieval effectively for the Thai language [8], [10]. This is because the top 20 of the high frequencies' 3-gram and 4-gram terms are complete words in the Thai language, where Thai words have varying lengths. As a result, three and four are both used as the best parameters in the n-gram inverted index for the Thai language. In the Thai language, Jaruskulchai [8] showed that the more probable *n* for the Thai language should be greater than two. By selecting the *n* greater than two, one could increase the possibility of achieving the effective retrieval, since the minimum number of characters for Thai word appearance is two, with at least one of them being a consonant. Furthermore, each Thai character cannot represent a word or a meaning like Chinese or Japanese. In Thai, the smallest unit which can represent a word or a meaning is a syllable.

Due to the above reasons, there is no single parameter for n-gram that is best for all un-delimited texts and applications. The following paragraphs will describe the process of n-gram term extraction.

Assume that document *d* consists of a string of characters $a_1, a_2, ..., a_N$. An n-gram term is a substring of *n* overlap or non-overlap successive characters extracted from the string. Extracting a set of n-gram terms from the documents *d* can be done by using the 1-sliding technique [39]. That is, sliding a window of length *n* from a_1 to a_N and storing the characters located in the window. Therefore, the *i*th n-gram term extracted from document *d* is the substring $a_i, a_{i+1}, ..., a_{i+n}$. Figure 2.21 shows 1-gram, 2-gram, 3-gram, 4-gram, …, N-gram overlap sequence of the document *d* containing the string *s* ‘การประกอบการ’.

1-gram terms	ก, า, ร, ป, ร, ะ, ก, อ, บ, ก, า, ร
2-gram terms	กา, าร, รป, ปร, ระ, ะก, กอ, อบ, บก, กา, าร
3-gram terms	การ, ารป, รปร, ประ, ระก, ะกอ, กอบ, อบก, บกา, การ

:

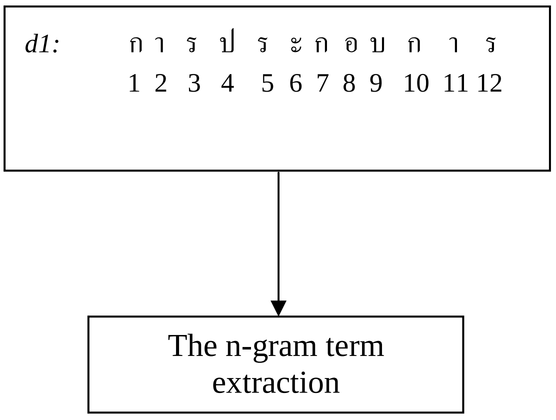
N-gram terms	การประกอบการ

Figure 2.21. Sets of 1-gram, 2-gram, 3-gram, …, N-gram overlap sequence of document *d* containing the string *s* ‘การประกอบการ’.

Construction of an n-gram inverted index

To construct the n-gram inverted index, the same technique used in the word inverted index construction (as described in Section 2.4.1.1) is employed. After Thai text documents are segmented into a series of indexing terms or n-gram terms using the n-gram term extraction, all tokenized indexing terms are then stored in alphabetical order in the inverted index.

From Figure 2.21, an example of building the n-gram inverted index—which is created on the document *d1* containing the string *s* ‘การประกอบการ’—is shown. In this example, 3-gram is used as one of the parameters, depicted in Figure 2.22 as most 3-gram terms are meaningful in the Thai language.



Inverted Index

3-gram terms	
Vocabulary	**Posting file**
กอบ :	<*d1*, 1, [7]>
การ :	<*d1*, 2, [1, 10]>
บกา :	<*d1*, 1, [9]>
ประ :	<*d1*, 1, [4]>
รปร :	<*d1*, 1, [3]>
ระก :	<*d1*, 1, [5]>
อบก :	<*d1*, 1, [8]>
ะกอ :	<*d1*, 1, [6]>
ารป :	<*d1*, 1, [2]>

An n-gram inverted index construction using the trie data structure

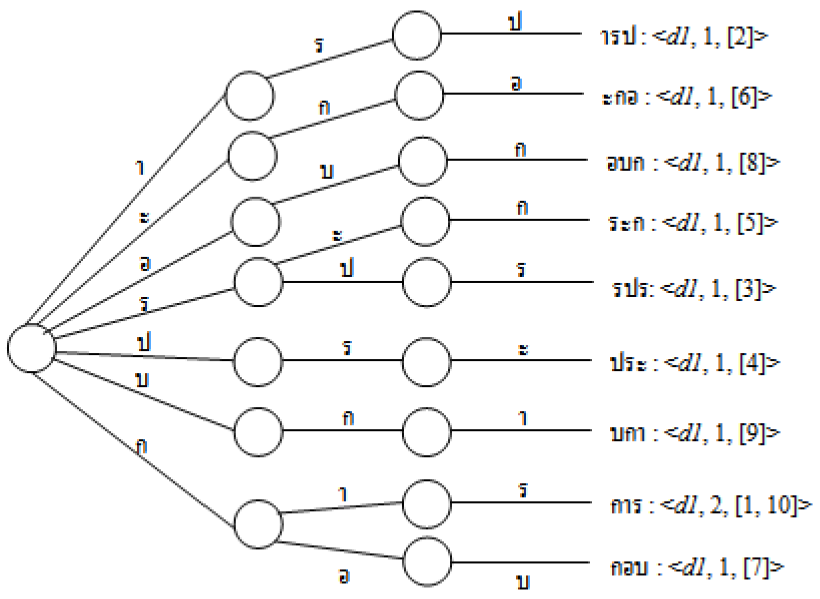


Figure. 2.22. Example of building n-gram inverted index for documents containing the string *s* 'การประกอบการ'

The advantage of the n-gram inverted index is language-independence [7], [26], [65]. Hence, it was one of the promising alternatives for indexing Thai text documents [10]. This n-gram inverted index can be used to search for the query where dictionary or language analysis may not be used. There are many applications of the n-gram based approach such as string searching, approximate string matching, and similar sequence matching in bioinformatics [66]. However, this technique suffers from a larger index size and poor retrieval time [26], [10], [39] when compared to the word inverted index. Like the word inverted index, the n-gram inverted index requires query processing and text pre-processing to extract n-gram terms before retrieval and indexing can be performed.

2.4.2.2 Suffix array approach

Another indexing approach is by using a data structure called suffix array. Within the suffix array scheme, Thai texts are viewed as a sequence of characters which can be structured by using an array. Suffix array approach can be viewed as a full text indexing technique. This technique does not require indexing term extraction like word inverted index and n-gram inverted index approaches. Suffix array approach is one of the more efficient methods for computing terms and document frequency for all substrings (i.e. keywords) from the text. This technique was proposed in 2001 [28] by Yamamoto and Church. The algorithm is based on suffix arrays [67] for computing *tf* (term frequency) and *df* (document frequency). Suffix array can also be used to solve substring problems. Term frequency (*tf*) is the standard notion of frequency in corpus-based natural language processing. It counts the number of times that a type (term-word-n-gram) appears in a text.

Suffix array construction

The suffix array data structure makes it convenient to compute the frequency and locations of a substring in a text. The lexicographical ordering technique is used to group all suffixes together in the suffix array, and can be found efficiently with a search algorithm. This technique constructs a suffix array that contains all suffixes that are sorted alphabetically. A suffix, also known as a semi-infinite string, is a string that starts at position *i* in the text and continues to the end of the text. Therefore, the constructed suffix array shows all possible substrings [28].

The constructed suffix arrays of a string can be used as an index to locate all occurrences of a substring within the string. Finding all occurrences of the substring is equivalent to finding every suffix that begins with the substring. This enables the algorithm to compute the term frequency using overlapping computation. As a result, suffix array can be used to search substrings efficiently. The following section depicts constructing the suffix array to compute term frequency and to retrieve the substring. Figure 2.23 shows an illustration of suffix array from string *s* = “การประกอบการ”

Let string *s* = ‘การประกอบการ’

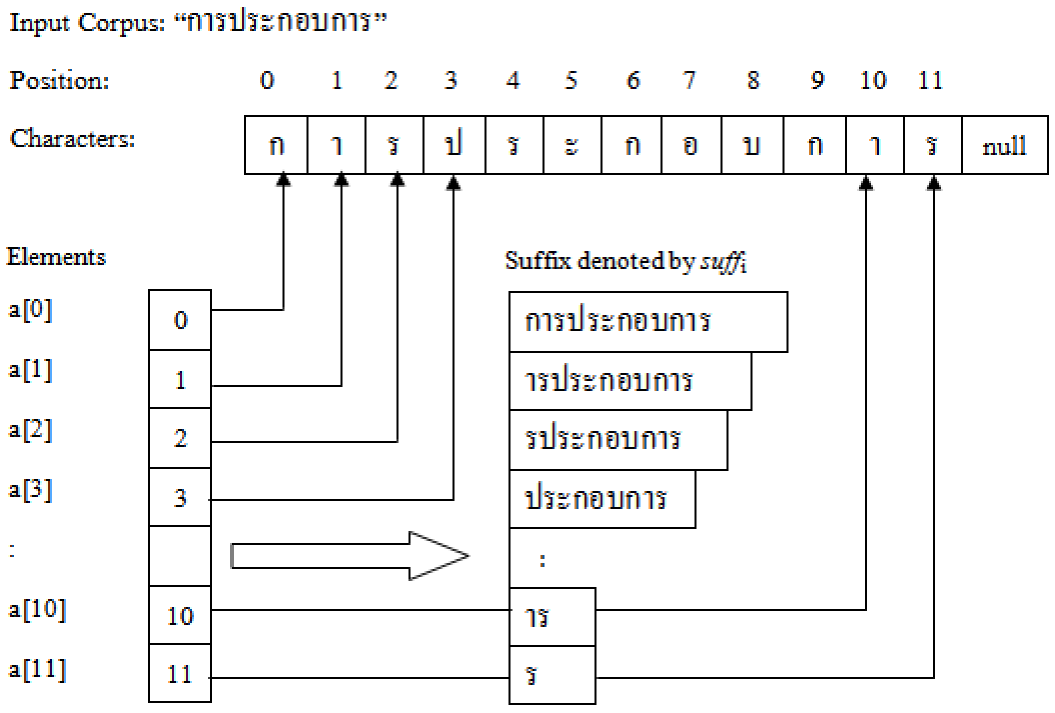
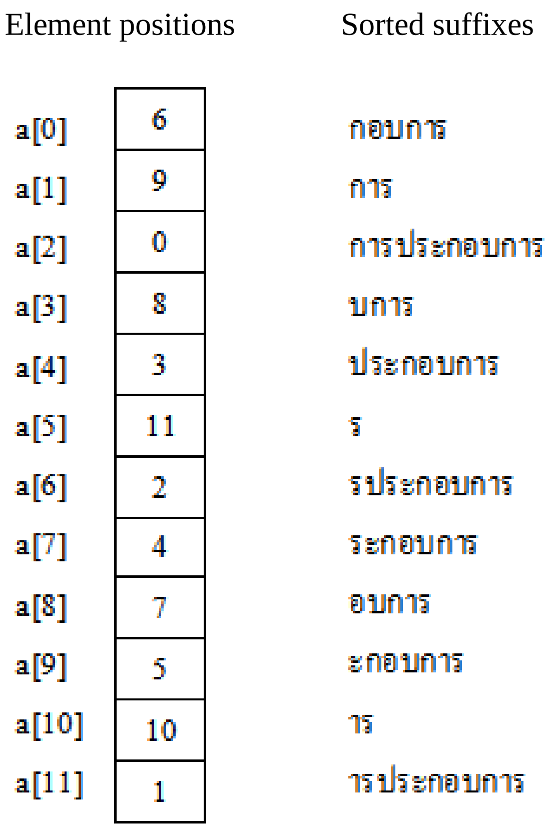


Figure 2.23. Illustration of suffix array from string *s* = ‘การประกอบการ’

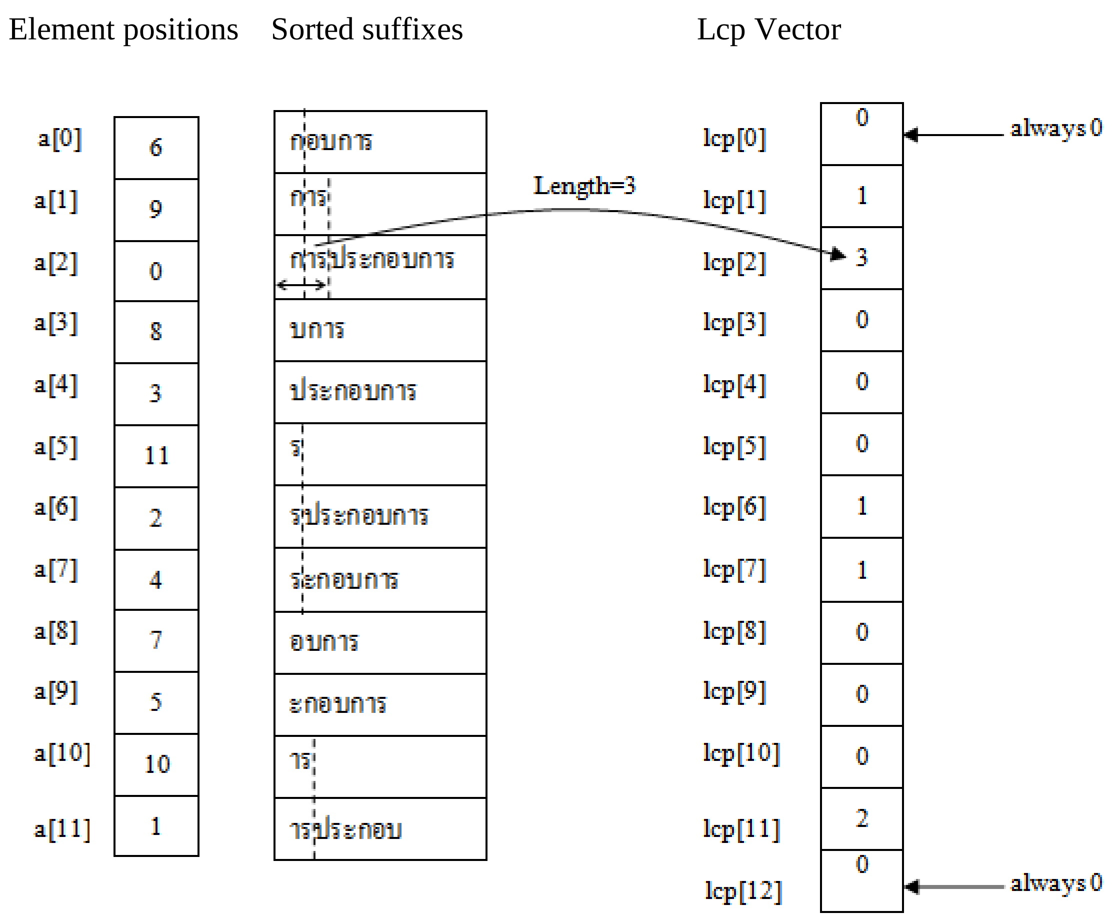
From Figure 2.23 the suffixes are enumerated by using suffix array, but elements in the suffix array have not been initialized and sorted. For each element in the suffix array, a[*i*] is an integer denoting a suffix, starting at position *i* in the text and extending to the end of the text. The elements in the suffix array are then sorted in alphabetical order for the next process as shown in Figure 2.24.



	Element positions	Sorted suffixes
a[0]	6	กอบการ
a[1]	9	การ
a[2]	0	การประกอบการ
a[3]	8	บการ
a[4]	3	ประกอบการ
a[5]	11	ร
a[6]	2	รประกอบการ
a[7]	4	ระกอบการ
a[8]	7	อบการ
a[9]	5	ะกอบการ
a[10]	10	าร
a[11]	1	ารประกอบการ

Figure 2.24. Illustration of suffix array from Figure 2.23, which has been sorted in alphabetical order

Additional to using suffix array, LCP (longest common prefixes) [28] was proposed as an algorithm that makes use of an auxiliary array to store long and repeated substrings from texts [68]. This algorithm is used to compute the term frequency using overlapping computation as depicted in Figure 2.25.



LCP-delimited	Terms	LCP-length	Term frequency (TF)
<0,2>	{“ก”}	1	3
<1,2>	{“การ”}	3	2
<5,7>	{“ร”}	1	3
<10.11>	{“าร”}	2	2

Figure 2.25. All longest common prefixes with their length and term frequency from suffix array

From Figure 2.25 it can be observed that term frequency can be computed on suffix array after sorting by using the longest common prefixes. The vertical dash lines represent the substrings, which appear multiple times in the text and the horizontal arrow shows the length of substrings. As a result, by using suffix array, the text documents can be indexed in order to support the search of the queries with their frequency and positions. Note that more detail of the suffix array approach will be provided in Chapter 3.

2.5 The retrieval process

At this point, the view of the retrieval process is detailed. The retrieval process is a field of study that helps users to find the required information from a large collection of text documents. The basic method of retrieval is to use the document index to retrieve documents that are likely to be relevant to a query. To describe the retrieval process, a general retrieval system is first illustrated in Figure 2.26.

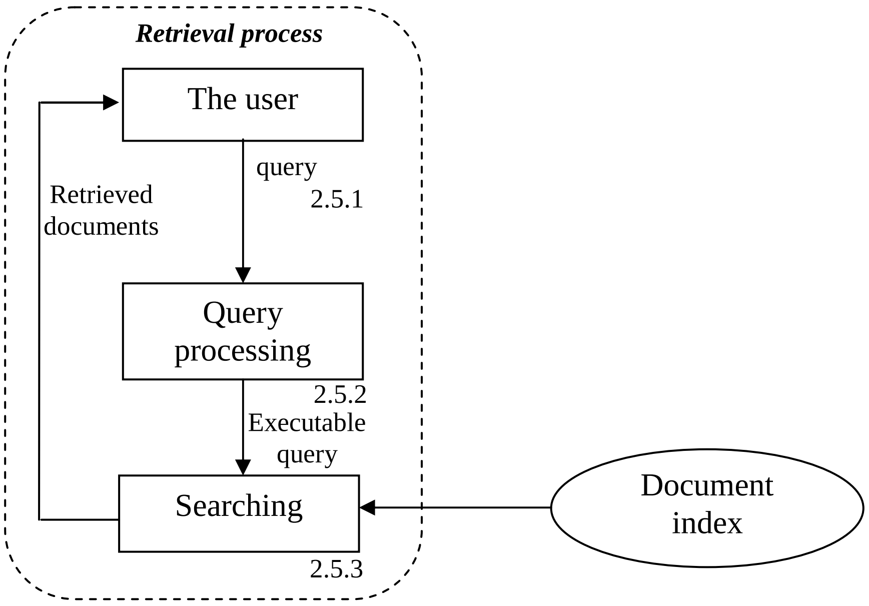


Figure 2.26. General retrieval system (numbers beside each box indicate sections that cover corresponding topic)

From Figure 2.26, it can be observed that there are several processes in the retrieval system. First of all, the user specifies a user need, which is usually called the query. This query is then parsed and transformed by the query operation. The query operation is a necessary process that is usually done before the searching process can be initiated.

The query operation transforms the original query to be appropriate and executable for the searching process. Once the original query is transformed, this executable query will then be passed forward to the searching process, in order to look up the relevant documents and obtain the retrieved documents from the document index.

For ease of better comprehension, the following sections provide more detail of each process including: a query, the query operation and searching.

2.5.1 A query

A query is a process of getting the information that the users need [7], [45], [69]. In the Thai information retrieval system, different kinds of queries are normally posed to the search engine [1]. Commonly used queries in the Thai information retrieval system can be divided into three forms as follows [7], [1], [69]:

- **Single word queries**

In the information retrieval system, a word is regarded as the most elementary query that can be formulated in text. Single word queries are normally defined in a simple way in searches [7], [69]. They generally produce the highest volume of searches, but the amount of returned results cannot satisfy users’ needs, also called the lowest amount of targeted traffic. This is because many users always start with single word queries, only to find that the results produced are not targeted for their specific need or intent. They then go back to refine their search, often multiple times, using various word combinations, until they find the best combination of words that gives them the results they need. For instance, if users are interested in the information about ‘ค้นคืน’ (retrieval), the information retrieval system will treat ‘ค้นคืน’ as a single word query to find the documents that contain this query. Furthermore, users can also use a set of single word queries to be more specific in finding the information they need. For example, if users intend to find out the information about 'การค้นคืนข้อมูล' (information retrieval), which has to be segmented into 'การค้นคืน-ข้อมูล' using Thai word segmentation, this 'การค้นคืน-ข้อมูล' query will be treated as 'การค้นคืน AND ข้อมูล'. The relevant documents that contain both single word queries will be retrieved for the users, although two single word queries appear to be far apart from each other in the text documents.

- **Phrase queries**

Phrase queries are mentioned in terms of using phrases. Phrase queries are queries consisting of a sequence of characters [7], [69]. In phrase queries, it is necessary for each returned relevant document to contain at least one instance of the exact phrase. The phrase queries are always enclosed with double quotes to indicate the characteristic phrase queries. For instance, it is possible for users to use the following phrase query including the double quotes, "การค้นคืนข้อมูลและเทคนิคการสืบค้น" or "information retrieval and search techniques" to retrieve documents that contain this exact phrase. The phrase queries are usually searched for by using full text scanning. The full text scanning method is based on string matching, which can be used as an option to search for exact phrases from text documents by scanning through all documents sequentially. The operation for such a method is to search the documents containing a given phrase query using matching string. These methods provide good accuracy, but have poor computational performance because the search time can be quite long depending on the document size [7]. Several approaches are proposed by researchers to develop full text scanning : Knuth, Morris and Pratt [70], Boyer and Moore [71], Sunday [72], Aho and Corasick [73], Wu and Manber [74], and Hollaar [75].

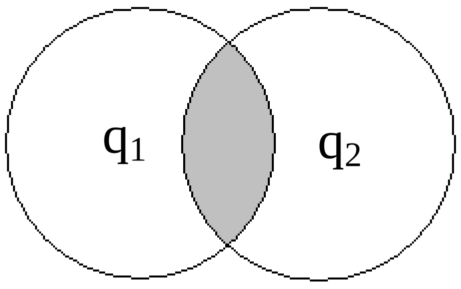
However, not all systems use phrase queries, even though they are very useful in most cases. Many information retrieval systems generally employ phrase queries in order to improve the ability to search for relevant documents. This is because a phrase may be significant in identifying the relevant documents. For instance, a user might want to find phrases from text documents with specific information that they need.

- **Boolean queries**

One standard model of a query developed from a term list is a Boolean query [7], [69]. The Boolean query combines queries using Boolean operators. The most commonly used Boolean operators are AND, OR, and NOT. The users can use these operators to construct complex queries. Each operator can be visually described by using Venn diagrams, as shown below.

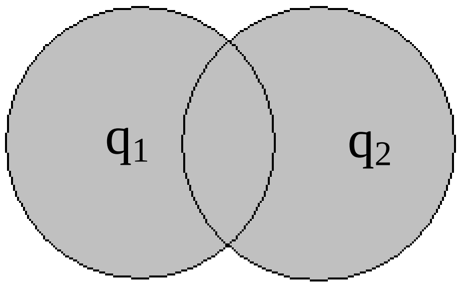
- **AND:** The query (q_1 AND q_2) retrieves all documents which contain both q_1 and q_2**.**

AND



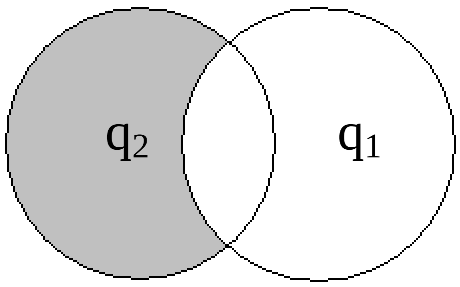
- **OR:** The query (q_1 OR q_2) retrieves all documents which contain q_1 or q_2**.**

OR



- **NOT:** The query (NOT q_1) retrieves all documents having no q_1**.** For this operator, a huge amount of text documents is always delivered, but it is probably not what the user wants.

NOT



Note that the construction of the query can be more complex, for example ((คำนวณ AND การค้นคืน) OR การสืบค้น) or ((compute AND retrieval) OR mining) in order to support the complex instructions.

In this thesis, only single word queries and phrase queries are used to evaluate the retrieval performance as will be demonstrated in Chapter 3. This is because these two types of queries are most commonly used to retrieve the relevant documents in the information retrieval system.

2.5.2 Query processing

Query processing is a module to transform queries to become appropriate for the retrieval system [7]. The query operations range from the simplest case to the most complex. In the simplest case, the raw query is just passed directly to the search engine without performing any processing. For complex cases, the query is processed before it is sent to the search engine by using some text pre-processing such as stopword removal, word segmentation or n-gram term extraction etcetera, which was discussed in the previous sections. In the most complex case, a query is transformed from a natural language query to an executable query suitable for computation.

2.5.3 Searching

The search process mentioned in this thesis is not referring to search algorithms but referring to simple indexing lookup algorithms. In the search process, the methodology used to search the relevant documents is dependent on the form of the document index. As discussed in the Thai text indexing technique section, the indexing techniques used for the Thai language can be divided into three approaches: the word inverted index, the n-gram inverted index and the suffix array approach. Therefore, searches on the document index can be categorized into two main types: search using the inverted index and search using the suffix array. In following section, the search using the inverted index is described. Meanwhile, the search using the suffix array will be described in Chapter 3.

2.5.3.1 Search using inverted index

Among the Thai text indexing techniques, the word inverted index and the n-gram inverted index are the techniques that use construction of the inverted index as the document index. In the search using the word inverted index and n-gram inverted index, it is necessary at all times to perform query processing before the search process can be initiated. Query processing is one part of query operation. To search on the word inverted index, it is necessary to apply Thai word segmentation and stopword removal to the query. The query is usually transformed into a set of single word queries by using the Thai word segmentation technique, and then stopwords are removed from this set of single word queries before searching. Meanwhile, it is necessary to apply the n-gram term extraction to the query before searching on the n-gram inverted index. Unfortunately, phrase query searches are not applicable for the word inverted index and n-gram inverted index since these two techniques do not contain phrases as the indexing terms.

For query searches using the inverted index, there are generally three steps: vocabulary search, retrieval of occurrences and manipulation of occurrences.

- **Vocabulary search:** The search first verifies that the query has valid terms by searching the vocabulary on the inverted index. Notice that phrase queries are segmented into a set of single word queries for Thai information retrieval. This set of single words is searched to see whether it is part of the vocabulary on the inverted index.

- **Retrieval of occurrences:** When the query is found in the vocabulary, the list of positions of the query found is retrieved from the posting file. Therefore, it is a good idea to separate the vocabulary and posting file into different places, because it is possible that the data will not fit in main memory due to large text collections [45], [7].

- **Manipulation of occurrences:** Finally, manipulation of occurrences is processed in order to solve the complex queries using Boolean operation.

2.6 Limitations of Thai text indexing techniques

While a number of indexing techniques have been proposed for the Thai language in order to enhance the performance of Thai information retrieval (as described in Section 2.4) this review also revealed several underlying limitations of these techniques. Thus, the motivation of this research is to address those limitations. In this section, the discussion on advantages and disadvantages of Thai text indexing techniques described in Section 2.4 is provided. In order to compare the Thai text indexing techniques, Table 2.3 points out the advantages and disadvantages of three indexing techniques: the word inverted index, the n-gram inverted index and the suffix array approach. For further information, these three techniques are also compared and discussed in term of indexing efficiency and retrieval performance in Section 3.5.

Table 2.3. Advantages and disadvantages of Thai text indexing techniques

Thai text indexing techniques	Advantages	Disadvantages
The word inverted index	• Requires less storage space for indexing when compared to the n-gram inverted index and the suffix array approach	• Requires word segmentation as text pre-processing before indexing • Requires long computation time for indexing when compared to the n-gram inverted index and the suffix array approach • Requires additional space for storing a dictionary or corpus or manually hand crafted rules. • Can only support one language or application depending on the dictionary or corpus or language knowledge used • Requires word segmentation to perform query processing before searching
The n-gram inverted index	• Language-independent technique • Supports any language or application	• Requires n-gram term extraction to perform text pre-processing before indexing • Requires n-gram term extraction to perform query processing before searching • Requires more storage space for indexing when compared to the word inverted index technique
The suffix array approach	• Language-independent technique • Does not require text pre-processing and query processing before indexing and searching • Supports any language or application	• Requires more storage space for indexing when compared to the word inverted index and the n-gram inverted index

Table 2.3 gives an overview of the limitations of the different Thai text indexing techniques. These limitations are discussed in detail in the following sections, categorized by the type of the technique.

- **The word inverted index technique**

The word inverted index can be regarded as the more widely used indexing technique in Thai information retrieval as described in Section 2.4.1. The main advantage of this technique is that it requires less space for indexing and storing the indexing terms when compared to the n-gram inverted index and suffix array approaches, as presented in the experiment in Section 3.5.1. However, indexing Thai text documents using the word inverted index has shown several limitations. The main limitation of the word inverted index is that the process of constructing the word inverted index is very time consuming. This limitation is caused by the need for word segmentation to perform text pre-processing in extracting the indexing terms before the inverted index can be constructed (as described in Section 2.4.1.2). Most word segmentation approaches require complex language analysis and thus require long computation time, and sometimes require dictionaries or corpora that are costly to maintain (as described in Section 2.4.1.2). Beside this, the word inverted index also requires additional storage space for storing a dictionary or corpus or manually hand crafted rules to perform word segmentation.

Another limitation of the word inverted index is that it is language-dependent. The word inverted index needs knowledge of the language to extract the indexing terms before indexing can be performed.

From a search point of view, the limitation of the word inverted index is that this technique requires query processing before the searching process can be performed (as described in Section 2.5.3.1 and will be shown in Section 3.5.2). To search the word inverted index, it is necessary to apply Thai word segmentation to the query before it is sent to the search process to look up the relevant documents. This will be illustrated in more detail in Section 3.5.2. Note that the word segmentation technique applied to the query has to be the same word segmentation technique used to extract the indexing terms from the text documents.

- **The n-gram inverted index**

According to Table 2.3, the main advantage of the n-gram inverted index is that this technique supports any language or application due to its being a language-independent technique. Due to this advantage, the n-gram inverted index has been one of the most often used indexing techniques for many Asian documents, and it has also been used in analyzing genome sequences in bioinformatics as described in Section 2.4.2.1. However, some limitations of the n-gram inverted index still exist. Its first disadvantage is that the n-gram inverted index requires the indexing term extraction using the n-gram term extraction method before the inverted index can be constructed (as described in Section 2.4.2.1). Determining the dimensions of the gram term is essential, so that they are appropriate for each application as described in Section 2.4.2.1. Additionally, the n-gram inverted index has limitations in terms of storage space. The n-gram inverted index requires larger space for storing indexing terms when compared to the word inverted index, because the number of indexing terms extracted by the n-gram inverted index is usually more than the number of indexing terms extracted by the word inverted index. This has been shown in the experiment presented in Section 3.5.1.

Furthermore, from a search point of view, a limitation of the n-gram inverted index is that it requires query processing in extracting the n-gram terms from the query before searching can be performed (as described in Section 2.5.3.1 and will be shown in Section 3.5.2).

- **The suffix array approach**

The suffix array approach is one of the language-independent techniques, which do not require the use of a dictionary or corpus or grammatical knowledge of a language. Due to being language-independent, the main advantage of the suffix array approach is that it is applicable for any language or application, for example many Asian languages or genome sequences in bioinformatics as highlighted in Section 2.4.2.2. The suffix array also does not require text pre-processing and query processing before indexing and searching can be performed. However, the main limitation of this approach is that it requires more storage space for indexing when compared to the word inverted index and the n-gram inverted index. This will be shown in the experiment presented in Section 3.5.1. One of the drawbacks is that storage space, in terms of the index size of the suffix array technique, could be critical. Since the size of electronically stored information in the Thai language has grown exponentially, the method of suffix array may not be practical for use in some applications.

By investigating the above limitations, the motivation of this research is to propose an efficient indexing technique for extracting indexing terms and constructing an index for Thai text documents. This proposed technique should address several limitations of existing Thai text indexing techniques. Firstly, in order to address the limitations of the word inverted index in terms of construction time, the proposed technique should be language-independent, which does not require the use of a dictionary or corpus or grammatical knowledge of a language. Thus, the proposed technique could be used to improve performance in terms of construction time over the word inverted index techniques, as this proposed technique does not require text pre-processing tasks in extracting the indexing terms before indexing can be performed. Additionally, the proposed technique could be applicable for any language or application due to its being language-independent. That is the main advantage of language-independent techniques. Secondly, the proposed technique should not require query processing before searching can be performed. Finally, to address the critical limitations of the suffix array approach in terms of index size, the proposed technique should focus on improving the space required in the language-independent technique.

2.7 Conclusion

In conclusion, research relating to Thai text indexing has been surveyed. In Thai text indexing, the methodologies can be categorized into two main techniques: language-dependent methods and language-independent methods. For language-dependent methods, the word inverted index technique was discussed. For language-independent methods, the n-gram inverted index and suffix array approaches were described. The disadvantages of the existing indexing techniques were examined. In the word invert index, word segmentation is required to extract the indexing terms before the inverted index can be constructed. However, most word segmentation approaches require complex language analysis and long computation time, and sometimes dictionaries or corpora that are costly maintain. Success of the word inverted index relies on the accuracy of word segmentation. Another of the drawbacks of the word inverted index is that it requires query processing using word segmentation before searching can be performed. The word inverted index also needs knowledge of an individual language in terms of extracting indexing terms, due to being language-dependent. While the n-gram inverted index is language-independent, it still requires indexing term extraction using the n-gram term extraction method before the inverted index can be constructed. Although the n-gram inverted index can be applied to many Asian languages and other sequence patterns due to its being language-independent, determining the appropriate dimensions of the gram term is problematic. This method also requires more space for storing indexing terms when compared to the word inverted index. Furthermore, like the word inverted index, the n-gram inverted index also requires query processing and text pre-processing to extract the n-gram terms before searching and indexing can be performed. Regarding the suffix array approach, this refers to a language-independent technique that can be applied to any language and other sequence patterns. However, one of its drawbacks is that this method obviously requires a large amount of storage space for indexing because it generates and keeps all suffixes from text documents during the indexing process. Although the suffix array approach does not require text pre-processing and query processing in terms of extracting the indexing terms before the suffix array can be constructed and searched, one of the drawbacks in terms of index size seems to be very critical. This makes the suffix array approach impractical at times to be used in the Thai environment, as the amount of the Thai language that is electronically stored has grown exponentially.

Chapter 3

Frequent Max Substring Technique

3.1 Introduction

In text indexing, the primary purpose of an index is to enhance performance in finding relevant documents which contain a query. As reviewed in Chapter 2, a number of indexing techniques have been proposed for the Thai language, such as the word inverted index, the n-gram inverted index or the suffix array method. Despite this, the search of relevant documents still varies in the way indexing is performed. Besides Thai text indexing techniques, there have been some alternative indexing methods for indexing Thai text documents, in order to facilitate fast and accurate searching. Common approaches are language-independent methods and they are based on string matching for the query within documents. The advantage of these approaches is that they do not require language analysis, text pre-processing and query processing.

Of the methods that do not use indexing, the oldest and conceptually simplest way was to scan through all documents in a collection sequentially to find documents that contain the query and to determine which documents satisfy the information need. This method is often referred to as 'the full text search' [70]. This is a tried and true method that is as old as collections of information-bearing items [69]. This technique does not need additional space for indexing, and it requires minimal effort for insertions and updates [56], [7], [71]. However, the main disadvantage of the full text search is its poor computational performance, because the search time may be quite long depending on the total size of the stored documents. However, if the total size of the documents is small, and is unlikely to be searched very often, the full text search seems to be an efficient strategy for searching documents which contain the query. Unfortunately, the number of stored documents in today’s collections is always very large. When the number of documents is large and is expected to be searched frequently, the full text search may not be practical. Hence, the problem of full text search can lead to index based approaches that are divided into two tasks: indexing and searching.

Indexing refers to scanning the text of documents and building a list of indexing terms via a useful data structure, in order to facilitate search and retrieval of documents. The searching stage will look up the query in the index rather than the text of the documents. The index based approaches appear to address the drawbacks of the full text search and become time and cost effective for query searching. The index based approaches would speed up searching response and improve the accessibility of the documents for multiple times.

A number of different index based approaches have been proposed. One of them is a substring indexing based on suffixes. This technique builds a set of substrings as the indexing terms. In substring indexing based on suffixes, suffix trie, suffix tree and suffix array are employed as alternative ways to construct the substring index. Suffix trie, suffix tree and suffix array are versatile data structures which are fundamental to string processing [47]. They are usually faster for string searching or substring enumeration than the full text search, but they are harder to build and maintain than the full text search [47], [71]. These data structures can be used to index any text as they see the text as a single string. Despite this, suffix trie, suffix tree and suffix array suffer from the requirement for larger space to construct the index and to keep pointers as they are used to enumerate the complete set of substrings from the string. A frequent substring indexing method was proposed to address one of the drawbacks of substring indexing, in terms of reducing the space requirement. The frequent substring indexing method was introduced by Vilo in 1998 [76]. This technique is based on suffix trie and it only indexes frequently occurring substrings in a string to avoid enumerating the complete set of substrings. Frequent substring indexing builds the index that contains only the frequent substrings as indexing terms. This is because indexing terms occurring less frequently in the string could usually be assumed to be insignificant [77], [69]. Although Vilo's method has been proposed for indexing only frequent substrings in order to reduce the space requirement, large storage space is still required for extracting and storing all frequent substrings that appear in the string. However, it is not necessary to index all frequent substrings if some technique can retain a smaller number of indexing terms that can contain all frequent substrings.

One of the motivations of this research discussed in this thesis is to create a space efficient Thai text indexing technique. This chapter aims to present the proposed technique, called the frequent max substring technique. The frequent max substring technique is used to extract indexing terms, known as the frequent max substrings, from Thai text documents. The set of frequent max substrings are able to contain all frequent substrings without information loss and further scan of the string. In order to extract the frequent max substrings, a new data structure is required. The proposed data structure, called the frequent suffix trie or FST structure, is employed to ensure exhaustive enumeration of substrings, to support extracting frequent max substrings.

In order to describe the frequent max substring technique clearly, substring indexing and frequent substring indexing are first examined in the next sections, as they are used as a basic methodology to design the frequent max substring technique. The remaining sections of this chapter are then organized as follows. Section 3.2 discusses substring indexing based on suffixes that include suffix trie, suffix tree and suffix array, as well as string-related common terminology. Section 3.3 describes frequent substring terminologies and Vilo's method for indexing frequent substrings. Section 3.4 introduces the concept of the frequent max substring and presents the proposed technique, the frequent max substring technique. The experimental studies, comparison results and discussion on indexing Thai text documents in terms of indexing efficiency and retrieval performance are presented in Section 3.5.

3.2 Substring indexing based on suffixes

The basic approach to substring indexing is based on a trie data structure [47], [78], [79]. The trie is a tree-based data structure for fast string matching [70], [80], [81]. The 'trie' comes from the word 're**trie**val' and was first introduced by Fredkin in 1960 [82]. The trie is designed for search in a large text or string [83]. The trie structure is built from an input string and the leaf node of each path containing the symbol '$' which represents the end of the string. There are five kinds of tries: non-compact trie [84], compact trie [85], compact Patricia trie [86], suffix trie and suffix tree [47]. These data structures are particularly useful for any application that requires string matching. Of these five types of tries, only two types, suffix trie and suffix tree [47], are used for enumeration of all possible substrings from an input string, because they basically construct over all suffixes of the string along the paths on the tries. In addition, the suffix array [67], [28], [81] is another data structure that provides the same functionality as the suffix trie and the suffix tree. The suffix array is designed for string processing applications. As a result, the suffix trie, suffix tree and suffix array can be used as an index to quickly locate every occurrence of a substring within the string. These data structures have also been used in bioinformatics with DNA sequences and some Asian languages such as the Thai language [1], [47].

In order to understand substring indexing based on suffixes, common terminologies are firstly provided followed by details of the suffix trie, suffix tree and suffix array.

Common definition:

String: Let $\sum$ be a finite set of characters. The size of $\sum$ is the number of unique characters in $\sum$, denoted $|\sum|$. A string over $\sum$ is any finite sequence of characters from $\sum$. The length of the string *s*, denoted $|s|$, is the number of characters in *s*. If $|s| = 0$, we call the string the empty string, denoted by λ. The set of all possible strings over $\sum$ is denoted $\sum^*$. For non-empty string $s = a_1\ a_2\ ....\ a_n$, where $a_i \in \sum$, individual characters in *s* are identified by their positions within *s*, thus character a_i at the position *i* on string s is also denoted by $s[i]$.

For example, let string *s* = ‘positivelives’ over $\sum$ = {p, o, s, i, t, v, e, l}. It can be seen that a_1 is p, a_2 is o, a_3 is s, a_4 is i, a_5 is t, a_6 is i, a_7 is v, a_8 is e, a_9 is l, a_{10} is i, a_{11} is v, a_{12} is e and a_n is s. The length of the string *s, or* $|s|$, is 13.

Substring: Let *s* and *x* be two strings, *x* is called a substring of *s* if $x \neq \lambda$ and $s = yxz$ for some strings *y* and *z*. Note that substring *x* may occur more than once within string *s*. For each occurrence of *x* in *s*, *x* is said to occur at position *j* if the last character of that occurrence is at position *j* of string *s*. A given occurrence of substring *x* in *s* is also denoted by $s[i, j]$ if the first and the last characters of the substring are at positions *i* and *j* of string *s* respectively.

The notation $x \subseteq s$ is used to denote that *x* is a substring of *s*. If $x \subseteq s$ and $x \neq s$, then *x* is said to be a *proper* substring of *s*, denoted $x \subset s$.

For example, let string *s* = ‘positivelives’. String *x* = ‘i’ is a *proper* substring of *s* that has length 1, and this substring has multiple occurrences in *s*. The occurrences of the substring in *s* are *s*[*4, 4*], *s*[*6, 6*], and *s*[*10, 10*]. If x = ‘ive’, *x* is a *proper* substring of *s* of length 3 and this substring has multiple occurrences in *s*, which are *s*[6, *8*], and *s*[*10, 12*].

Substring set: Let *s* be a string. The substring set for *s*, denoted *SS*(*s*), is defined to be the set of all substrings of the given string *s*. Note $\lambda \in SS(s)$.

For example, let string *s* = ‘positivelives’.

SS(*s*) is { λ , p, o, s, i, t, v, e, l,

po, os, si, it, ti, iv, ve, el, li, es,

pos, osi, sit, iti, tiv, ive, vel, eli, liv, ves,

posi, osit, siti, itiv, tive, ivel, veli, eliv, live, ives,

posit, ositi, sitiv, itive, tivel, iveli, veliv, elive, lives,

positi, ositiv, sitive, itivel, tiveli, iveliv, velive, elives,

positiv, ositive, sitivel, itiveli, tiveliv, ivelive, velives,

positive, ositivel, sitiveli, itiveliv, tivelive, ivelives,

positivel, ositiveli, sitiveliv, itivelive, tivelives,

positiveli, ositiveliv, sitivelive, itivelives,

positiveliv, ositivelive, sitivelives,

positivelive, ositivelives,

positivelives}

Superstring: If *x* and *y* are two strings over the same $\sum$ and *x* is a substring of *y*, then *y* is called a superstring of *x*, denoted $y \supseteq x$. Furthermore, if $x \neq y$, *y* is called a *proper* superstring of *x*, denoted $y \supset x$.

Suffix: Let *s* and *z* be two strings, *z* is called a suffix of *s* if $z \neq \lambda$ and z is a substring of string *s* starting at position *i* and ending at the last position *n* of string *s*, denoted by $suff_i$

Suffix Set: Let *s* be a string. A suffix set for *s*, denoted *suff*(*s*), is defined to be a set of all suffixes of the string *s* [87].

Prefix: Let *s* and *y* be two strings, *y* is called a prefix of *s* if $y \neq \lambda$ and y is a substring of string *s* starting at the first position and ending at the position *i* of string *s*, denoted by $pref_i$

Prefix Set: Let *s* be a string. A prefix set for *s*, denoted *pref(s*), is defined to be the set of all prefixes of the string *s* [87].

Note that, to construct the suffix array, the suffix trie and the suffix tree, \$ is normally added to the end of the string to signal the string termination. Note that the terminating symbol \$ must not be a character of $\sum$ from which string *s* was formed. For any string *s*, the string *s*\$ is the actual input to the suffix array, the suffix trie and the suffix tree algorithms. Figure 3.1 shows all suffixes of the resulting string *s* ‘positivelives\$’.

Let string *s* = p o s i t i v e l i v e s $

Positions = 1 2 3 4 5 6 7 8 9 10 11 12 13 14

Enumerate all suffixes of string *s*

1. positivelives$
2. ositivelives$
3. sitivelives$
4. itivelives$
5. tivelives$
6. ivelives$
7. velives$
8. elives$
9. lives$
10. ives$
11. ves$
12. es$
13. s$
14. $

Figure 3.1. All suffixes of string *s* ‘positivelives$’

3.2.1 Suffix trie

Suffix trie is regarded as one kind of trie data structure built over all suffixes of the string. This data structure is a simplified but more resource demanding version of the well-known data structure suffix tree [78], [88], [89], [47]. The suffix trie is a versatile data structure for many string processing applications. When the suffix trie for a string has been constructed, it means all substrings of the string have been indexed and it can be used to quickly search the query or any substrings of the string using string matching. The suffix trie has the following properties:

Let *s* be a string of length *n*, the suffix trie for *s* has the following properties:

1. The tree has *n* leaves, labeled *1* to *n*, each corresponding to one suffix of string *s*
2. Each edge in the tree is labeled with a character from a finite set of characters

3. Every edge label has a length of exactly one
4. The concatenation of edge labels from the root to the leaf labeled *i* is $suff_{\mathrm{i}}$
5. The labels of the edges connecting a node with its children start with different characters

For string *s*, the suffix trie contains all its suffixes. Let *s* be a string of length *n* including the termination character \$. The processes of constructing suffix trie will start with an empty tree and iteratively insert suffixes one by one. The resulting tree will finally be the suffix trie when all suffixes are inserted into the tree as illustrated in Figure 3.2.

According to McCreight [88], the algorithm inserts suffixes in the order $suff_1$, $suff_2$, $suff_3$,…. $suff_{\mathrm{n}}$. Let T_{i} denote the suffix trie after $suff_{\mathrm{i}}$ is inserted. T_1 is the tree consisting of a single leaf labeled 1 that is connected to the root node by edges with the label substring $s[1, n]$. In iteration *i* of the algorithm, $suff_{\mathrm{i}}$ is inserted into $T_{\mathrm{i-1}}$ to form T_{i}. The way to do this is by starting from the root and following the unique path matching characters in $suff_{\mathrm{i}}$ one by one until no more matches are possible. If the traversal does not end at an internal node, create an internal node and another branch there. Then, attach a leaf labeled *i* to this internal node and use the unmatched portion of $suff_{\mathrm{i}}$ for the edge label. The run time for inserting $suff_{\mathrm{i}}$ is proportional to $|suff_{\mathrm{i}}| = n – i + 1$. The total run time for constructing the suffix trie is therefore $\sum_{i=1}^{n}(n-i+1) = O(n^2)$ [47], [88], [89]. Considering the space issue of the suffix trie, analysis of space usage in constructing the suffix trie is straightforward. For the string of length *n,* the suffix trie has *n* depth, and at most $n – 1$ internal nodes at any depth of the suffix trie. Hence, storage for the nodes in each depth requires $O(n)$ space complexity. As the suffix trie is the tree of *n* depth, the total space for constructing the suffix trie is therefore *O(n^2)* space complexity [47], [88].

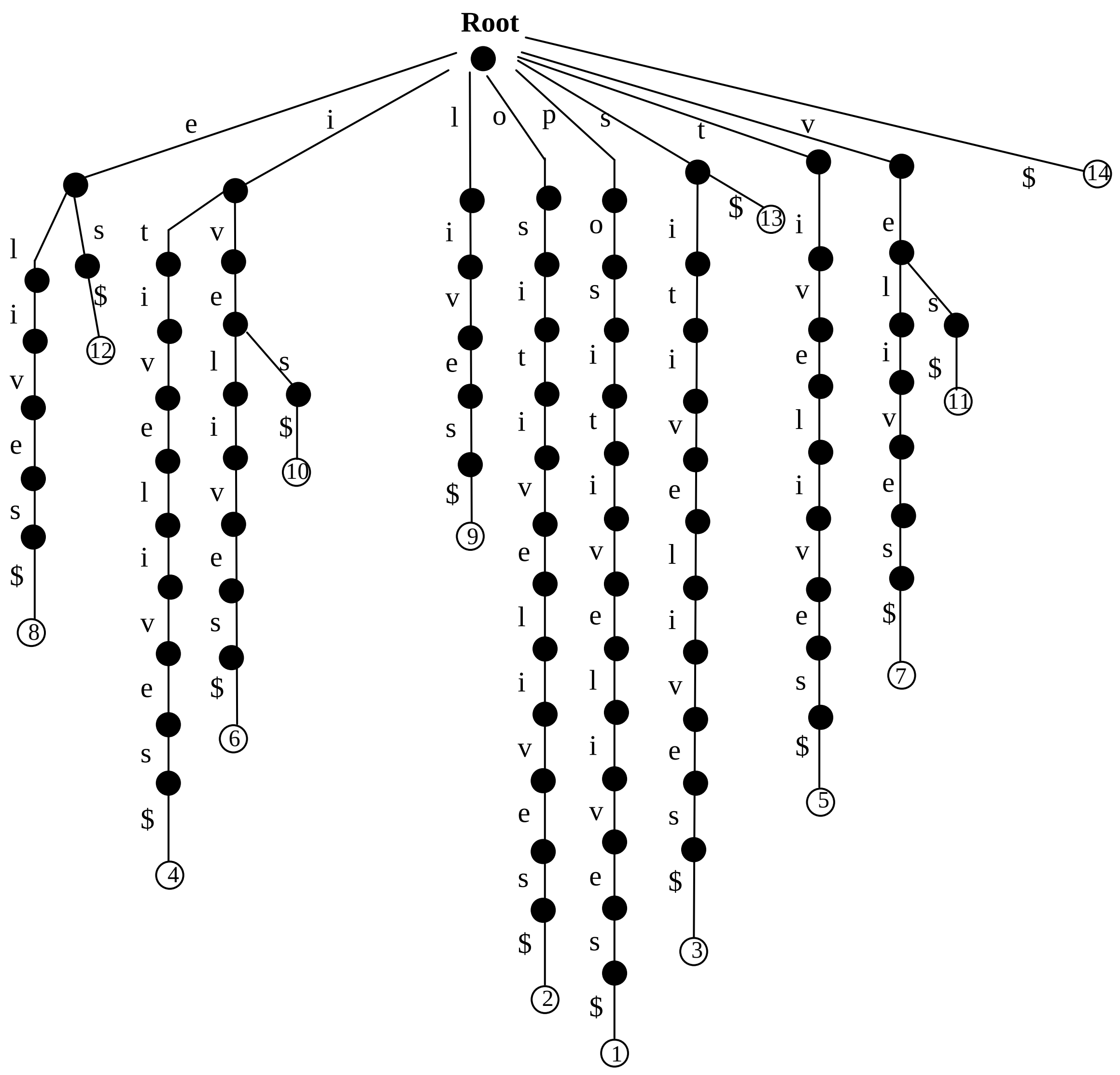


Figure 3.2. Suffix trie of string *s* = 'positivelives$'

To search a query using the suffix trie, a string matching algorithm is used to find a query (substring) within the string via the suffix trie. The string matching algorithm is an important class of string algorithms, which are a traditional area of study in computer science providing the generic implementation of string-related algorithms [47].

String matching problem*:* Given a query P and a string s, the string matching problem is to find all occurrences of P in s. Let $|P| = m$, is the length of the query P and $|s| = n$, is the length of the string s, typically $m \leq n$. In the text retrieval system, the string s could be a text document and P could represent the query. Thus, it is beneficial to index the string s so that the query can be answered as quickly as possible.

Considering the string matching problem, it can be shown that the query can be searched in $O(m + k)$ time using the suffix trie, where k is the number of occurrences of query P in the string s and m is the length of query P. Suppose query P occurs in the string s starting from position i, then query P is a prefix of $suff_i$ in the string s. It follows that query P matches the path from root to leaf labeled i in the suffix trie. This algorithm works by starting from the root of the suffix trie and follow the path matching characters in query P, until query P is completely matched or a mismatch happens. When the mismatch occurs, it means that query P is not contained in string s. In contrast, if query P is completely matched, then each leaf in the subtree below the matching position represents an occurrence of query P. The positions can be enumerated by traversing the subtree below. The matching position is k, this takes $O(k)$ time complexity. If only one occurrence is of interest, the suffix trie can be processed in $O(n)$ time complexity, so that each internal node contains the label of one of the leaves in its subtree. Thus, the problem of whether query P occurs in string s, or the problem of finding one occurrence can be answered in $O(m)$ time complexity [47], [88].

3.2.2 Suffix tree

The suffix tree is another kind of trie data structure that can be used to index all substrings of the string. The suffix tree represents all the suffixes of that string in the same manner as the suffix trie. The concept of the suffix tree was first introduced as a position tree by Weiner in 1973 [78] and the paper subsequently characterized it as the 'Algorithm of the Year 1973' [70]. The construction was greatly simplified by McCreight in 1976 [88] and also by Ukkonen in 1995 [89], [90]. Ukkonen provided the first linear-time online-construction of suffix trees, now known as Ukkonen's algorithm. Giegerich [91] provided wotd-algorithm or the write-only top-down algorithm in 1999, which was used for generating the suffix tree and suffix trie in the breadth-first order, level by level. Based on the suffix trie, the suffix tree provides the same functionality as the suffix trie but requires less space. The suffix tree differs from the suffix trie in that every edge label can have a length of more than one and every internal node has to have at least two children. Therefore, the suffix tree has the same properties as the suffix trie except Property 3, which is not true for the suffix tree. Because of this, the suffix tree generally has less internal nodes than the suffix trie, resulting in less space requirement. As each internal node has at least two children, and n-leaf suffix tree has at most $n - 1$ internal nodes, where n is the length of string s. Because of Property 5, the maximum number of children per node is bounded by $|\sum| + 1$. Except for the edge labels, the size of the suffix tree can be $O(n)$ space complexity by counting only the number of nodes in the suffix tree [47], [79], [88]. The number of nodes in the suffix tree for the string s is linear in $|s|$ or n, because there are n leaves and less than n internal nodes. As there are exactly n leaves and every internal node is a branching node, there are altogether $O(n)$ nodes in the suffix tree. However, if the edge labels in the suffix tree are considered, the suffix tree requires $O(n^2)$ space, the same as the suffix trie [47], [88]. This is because the number of edge labels in the suffix tree and the suffix trie are equivalent. So, the space complexity of the required suffix tree is $O(n^2)$ [47], [89], [90]. The suffix tree is constructed with the same process as used in constructing the suffix trie. Hence the total run time for constructing the suffix tree is $O(n^2)$ [47]. Figure 3.3 shows the constructed suffix tree from the string *s* ‘positivelives$’.

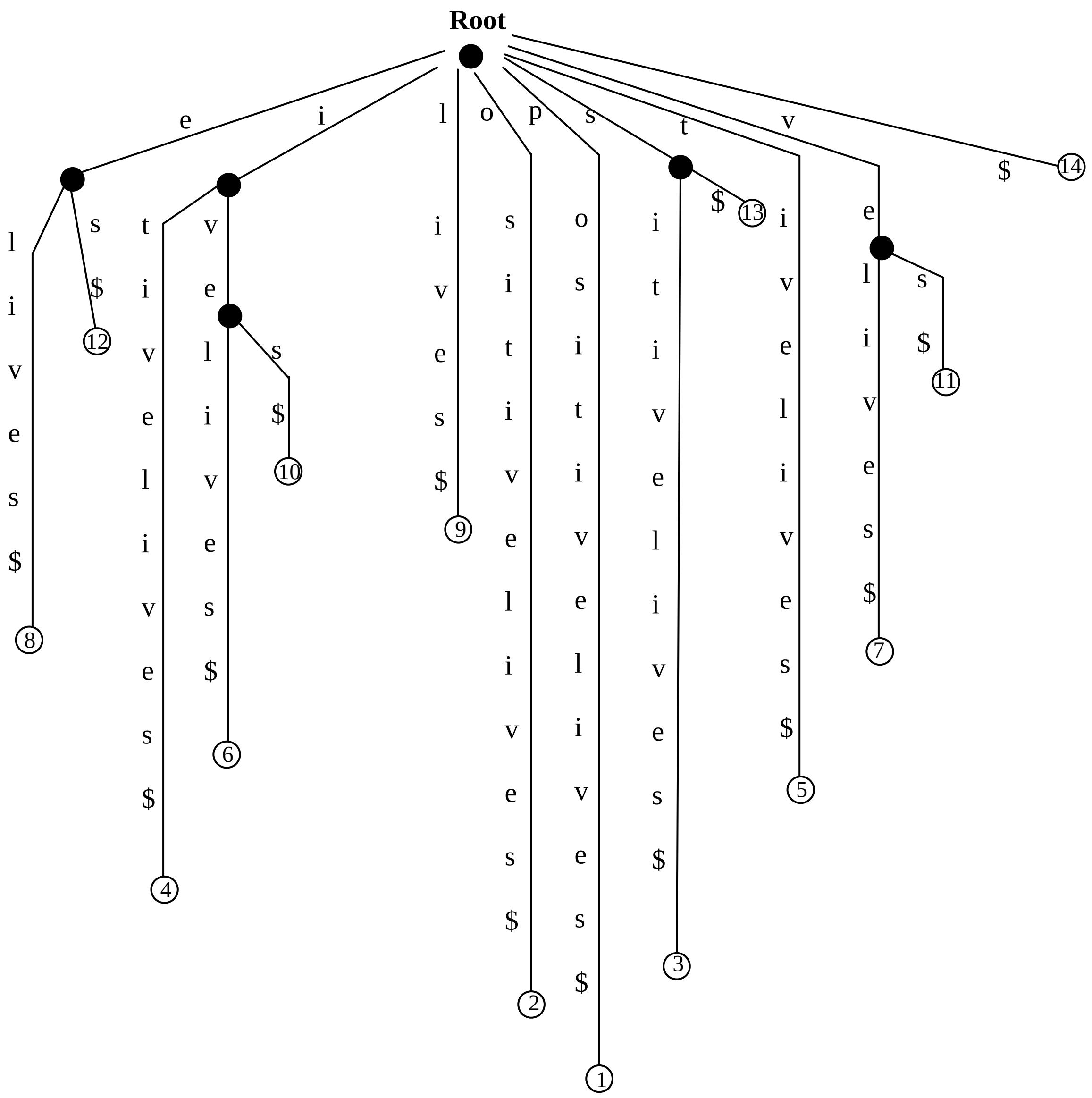


Figure 3.3. Example of suffix tree of string *s* ‘positivelives$’

In order to allow a linear space representation of the suffix tree, each edge label is represented by a pair of integers denoting the starting and ending positions of the substring describing the edge label. If the edge label corresponds to a repeat substring, the indices corresponding to any occurrence of the substring may be used. The suffix tree of the string *s* ‘positivelives$’ represented by a pair of integers is shown in Figure 3.4.

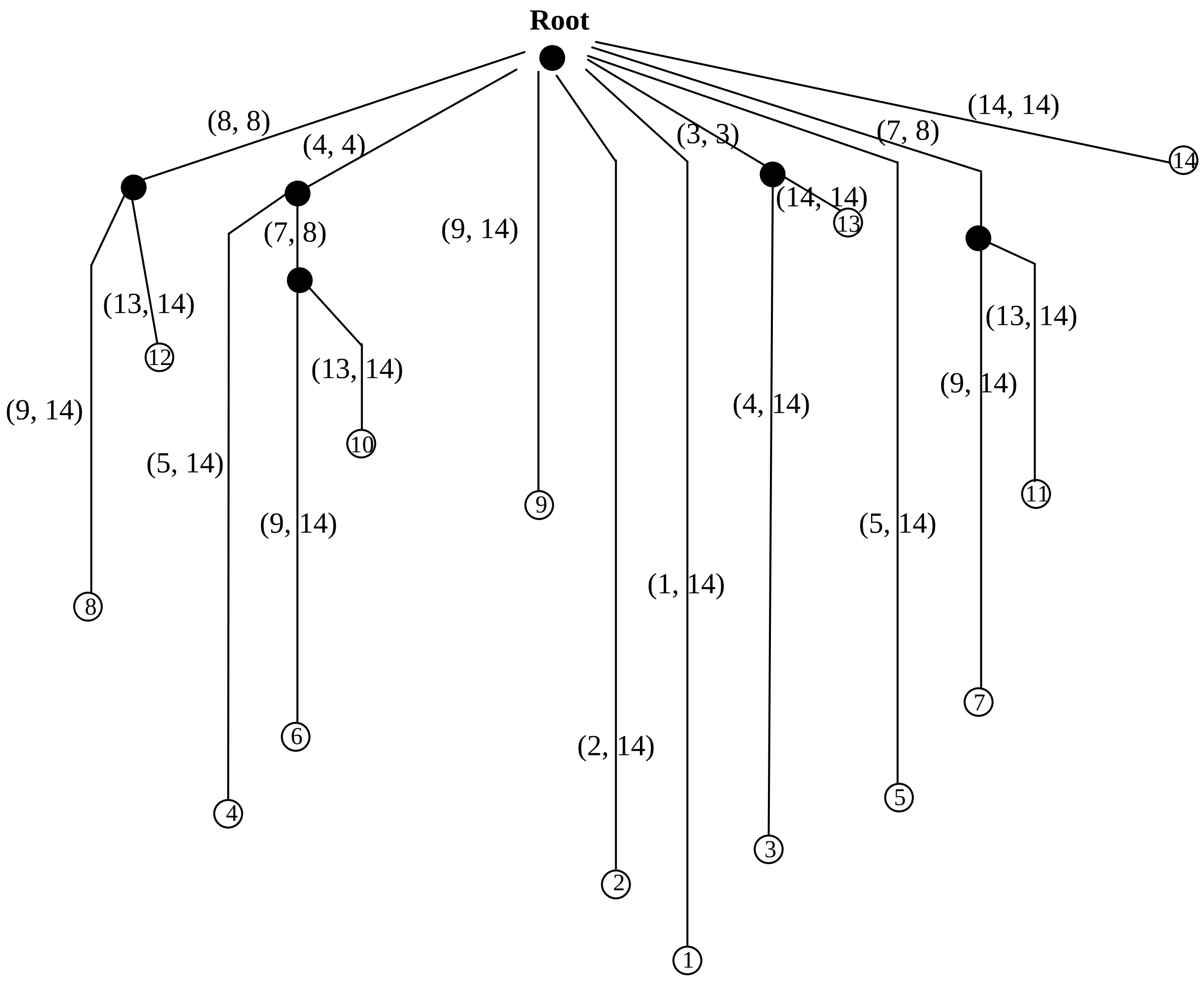


Figure 3.4. Suffix tree of string *s* ‘positivelives$’represented by a pair of integers denoting starting and ending positions

After building the substring index using the suffix tree, the occurrences of the query on the string can be found quickly by using the same technique used for finding the occurrences of the query on the suffix trie. Therefore, the query search can be processed in $O(m + k)$ time using the suffix tree [83], where k is the number of occurrences of the query in the string and m is the length of the query.

3.2.3 Suffix array

Suffix array [67] is another versatile data structure in string processing applications that provides essentially the same functionality as the suffix trie and suffix tree. This data structure is able to answer complex queries more efficiently. Within the suffix array, the input string is viewed as a sequence of characters, which can be stored in an array. The suffix array data structure makes it easy to compute the frequency and locations of a substring in the string. The lexicographical ordering technique is used to group all suffixes together in the suffix array. All the suffixes and pointers of the string are stored in suffix arrays in lexicographical order, so as to speed up retrieval. The pointers are used as the identifiers of suffixes in the string. Besides this, the position of each suffix in the string needs to be stored. The memory address of the suffix string needs to be stored as well. Once the suffix array is constructed, it can be used as an index to locate all occurrences of a substring within the string. Finding all occurrences of the substring is equivalent to finding prefixes of suffixes. This enables the algorithm to compute the term frequency using overlapping computation as described in Chapter 2.

In the process of constructing the suffix array, the character occurrences are first recorded in an index array, then suffixes in the index array are sorted in alphabetical order by using sorting algorithms such as Quicksort, bucket sort, selection sort, insertion sort, and so on [67], [47]. These sorting algorithms perform indexing with different time complexity. For instance, suffix array construction can be done in $O(n^2)$ time complexity by using the Quicksort algorithm [47]. Quicksort is known as one of the fastest sorting algorithms that use a recursive ‘divide and conquer’ technique to sort the data. In the use of the Quicksort algorithm, the suffix array is split into two partitions: the lower part and the upper part. To do this, a pivot element is chosen as a median, then the suffixes which are smaller than the pivot are moved to the lower part, and the suffixes which are larger than the pivot are moved to the upper part. Once the suffix array is partitioned, these partitions will be sorted separately using Quicksort. Because the Quicksort algorithm is applied on both partitions, both will in turn be split into partitions of their own. The suffix array will keep splitting in this manner until the partitions have only one element in them. Once no more partitions can be split, the partitions are actually sorted and they make up the sorted array. The process of constructing suffix arrays is illustrated in Figure 3.5 and Figure 3.6. However, the space required for the suffix array is of significant concern, especially for very large strings. As the suffix array contains all suffixes of the string, the catch is that constructing and storing the suffix array in this manner can require $O(n^2)$ space complexity since the average length of the n suffixes is $n/2$ [28], [47], [67], [81].

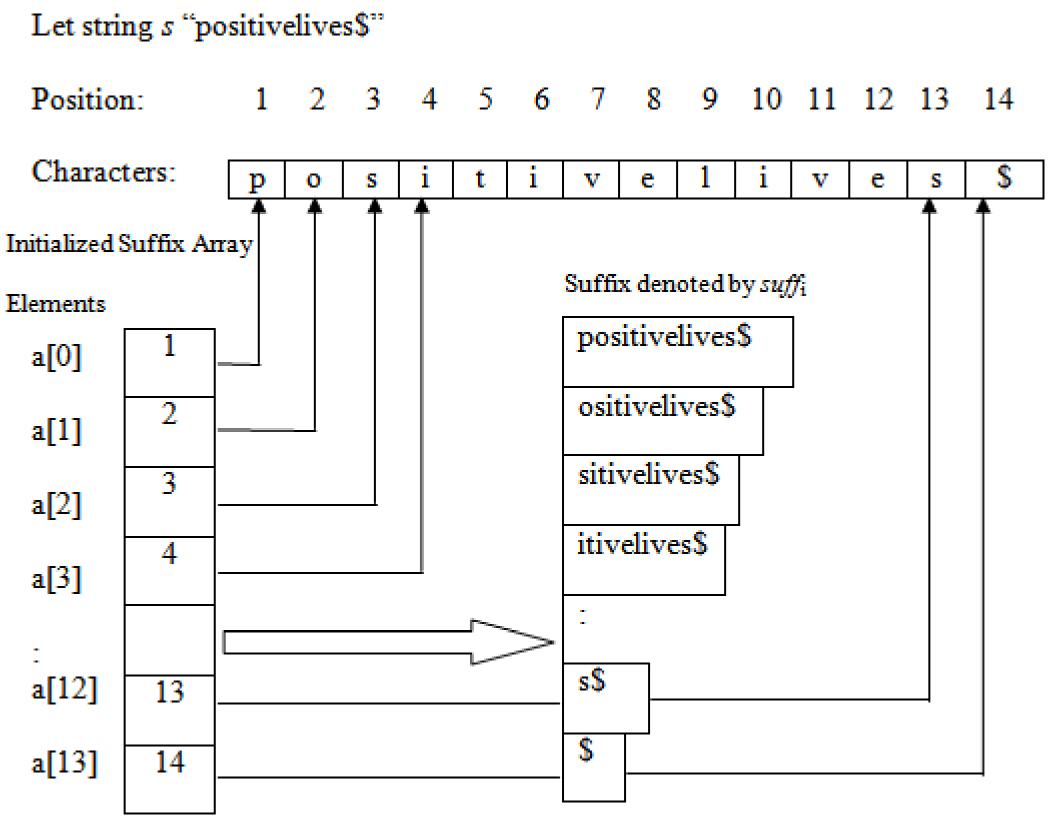


Figure 3.5. Illustration of suffix array from string *s* = 'positivelives$'

In Figure 3.5, the suffixes are enumerated by using a suffix array, but elements in the suffix array have not been initialized and sorted. Each element in the suffix array, *a*[*i*], is an integer denoting a suffix in the suffix array. Suffixes in the suffix array will then be sorted in alphabetical order by using a sorting algorithm as shown in Figure 3.6.

Element positions		Elements or Sorted Suffixes
a[0]	8	elives
a[1]	12	es
a[2]	4	itivelives
a[3]	6	ivelives
a[4]	10	ives
a[5]	9	lives
a[6]	2	ositivelives
a[7]	1	positivelives
a[8]	13	s
a[9]	3	sitivelives
a[10]	5	tivelives
a[11]	7	velives
a[12]	11	ves

Figure 3.6. Illustration of suffix array from Figure 3.5, sorted in alphabetical order

After the suffix array has been constructed, the search of a query or a substring can be performed quickly by using string matching. For string matching, the suffix array allows a binary search to efficiently search the query in the suffix array. With the binary search, the query can be found on the suffix array in *O(m log n)* [67], which can be derived in the following way. Consider the problem of string matching when the suffix array of the string is constructed. As before, it is necessary to find all suffixes that contain the query *P*. As the suffix array is a lexicographically sorted order of the suffixes of string *s*, all such suffixes will appear in consecutive positions in it. The sorted order in the suffix array allows easy identification of these suffixes using the binary search. The binary search works by comparing the query *P* to the middle element of the suffix array. The comparison will determine whether the middle element contains the query *P* as a prefix. If the query *P* is contained as the prefix of the middle element in the suffix array, then the search stops and returns the position of the element. In contrast, if the query *P* is not contained in the middle element, the comparison will then be made to determine whether the query *P* is less than or greater than the middle element. Depending whether *P* is less than or greater than the middle element, the search will repeat the same process only to the top or bottom subset of the suffix array. According to the above search operation, the binary search in the suffix array takes *O(log n)* comparisons. In each comparison, query *P* is compared with a suffix to determine their lexicographic order. This requires comparing at most the length of the query, |*P*|, or *m* characters. Thus, the time complexity of a search operation is *O(m log n)* in finding the occurrence of the query [67].

3.2.4 The comparison of suffix trie, suffix tree and suffix array

In order to compare the suffix trie, suffix tree and suffix array, the details of these three data structures are summarized in Table 3.1 followed by a description below.

Table 3.1. Comparison of suffix trie, suffix tree and suffix array

Data structure	Time complexity (construction)	Space complexity (construction)	Number of nodes	Searching time
Suffix trie	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(m + k)$
Suffix tree	$O(n^2)$	$O(n^2)$	$O(n)$	$O(m + k)$
Suffix array	$O(n^2)$ (Case of quick sort)	$O(n^2)$	N/A	$O(mlogn)$

The time and space complexity of constructing the substring index via the above three data structures are similar. Construction of the suffix trie has $O(n^2)$ time and space complexity where n is the length of the string, and the corresponding tree has $O(n^2)$ nodes, while constructing the suffix tree also takes $O(n^2)$ time and space but the corresponding tree has $O(n)$ nodes as described in Section 3.2.1 and 3.2.2. The suffix trie and suffix tree require equivalent time and space complexity because both data structures have equivalent edge labels, which spell out the complete set of substrings on the string. Depth-first traversal is usually used to enumerate all substrings of the string over suffix trie and suffix tree data structures. Meanwhile the suffix array requires construction time depending on the sorting algorithm used as described in Section 3.2.3. The space complexity of the suffix array construction is $O(n^2)$ as this technique works on sorting all suffixes of the string. The space complexity for storing the sorted suffix array is also $O(n^2)$ because this data structure contains all sorted suffixes of the string. From a search point of view, the time complexity of a search operation used by the suffix array is $O(mlogn)$ by using the binary search, where m is the length of the query and n is the length of the string as described in Section 3.2.3. Meanwhile, the suffix trie and suffix tree require $O(m + k)$ time complexity of the search operation, where m is the length of the query and k is the number of occurrences of the query in the string.

3.3 Frequent substring indexing with Vilo’s method

In 1998, Jaak Vilo presented an algorithm for discovering frequent substrings in a string [76]. This algorithm aims at finding substrings that occur frequently in the string (at or above the given frequency threshold value). This is achieved by constructing a pattern trie, which is based on the suffix trie, while maintaining information about the occurrences of each substring. The algorithm constructs only a subtree of the suffix trie that corresponds to frequent substrings of the string, to avoid enumerating the complete set of substrings and in order to reduce the space requirement. It builds the pattern trie for the input string in the breadth-first order, level by level, and creates a list of occurrences for each frequent substring in the string. Frequent substrings are constructed incrementally by expanding prefixes of the substrings that occur at least at the frequency threshold value. Only substrings that occur in the string and occur at least at the frequency threshold value are generated and analyzed. This algorithm has been successfully used for analyzing the full genome of yeast and for predicting certain regulatory elements, and it has also been used for string matching in bioinformatics where the string is a DNA sequence [76].

However, Vilo’s technique has not been used for Thai text indexing, as this approach was proposed for string matching in bioinformatics where strings of DNA sequences are searched and matched. Despite this, DNA sequences can be regarded as long contiguous strings with a specific alphabet, for example {A,C,G,T} in the genome [92], which is similar to the Thai language. This suggests that Vilo’s technique may be used as a language-independent technique for indexing Thai text documents. Therefore, Vilo’s technique will be applied for indexing Thai text documents, in order to compare it to the proposed frequent max substring technique in Section 3.5.

Before discussing Vilo's algorithm in detail, frequent substring terminology is firstly provided as follows, in order to understand the characteristics of the frequent substring.

Frequent substring terminologies*:*

Substring frequency: Let s and x be two strings and x is a substring of s. The substring frequency of x in s is defined as the number of different occurrences of x in s. The notation $f_s(x)$ is used to denote the frequency of substring x in string s.

For example, let string s = 'positivelives', and x = 'iv', then $f_s(x) = 2$.

Substring frequency set: Let s be a string. The substring frequency set for s, denoted $SFS(s)$, is defined as a set of all substring-frequency pairs, where each substring-frequency pair consists of a unique substring x of s and the substring frequency of x in s. Formally,

$$SFS(s) = \{ <x, f> \mid \text{ where } x \subseteq s \text{ and } f = f_s(x) \} \tag{1}$$

For example, let string s = 'positivelives'

$SFS(s)$ = {<p, 1>, <o, 1>, <s, 2>, <i, 3>, <t, 1>, <v, 2>, <e, 2>, <l, 1>,<po, 1>,
<os, 1>, <si, 1>, <it, 1>, <ti, 1>, <iv, 2>, <ve, 2>, <el, 1>, <li, 1>,
<es, 1>,<pos, 1>, <osi, 1>, <sit, 1>, <iti, 1>, <tiv, 1>, <ive, 2>, <vel, 1>,
<eli, 1>, <liv, 1>, <ves, 1>, <posi, 1>, <osit, 1>, <siti, 1>, <itiv, 1>,
<tive, 1>, <ivel, 1>, <veli, 1>, <eliv, 1>, <live, 1>, <ives, 1>, <posit, 1>,
<ositi, 1>, <sitiv, 1>, <itive, 1>, <tivel, 1>, <iveli, 1>, <veliv, 1>,
<elive, 1>, <lives, 1>, <positi, 1>, <ositiv, 1>, <sitive, 1>, <itivel, 1>,
<tiveli, 1>, <iveliv, 1>, <velive, 1>, <elives, 1>, <ositiv, 1>,
<ositive, 1>, <sitivel, 1>, <itiveli, 1>, <tiveliv, 1>, <ivelive, 1>,

<velives, 1>, <positive, 1>, <ositivel, 1>, <sitiveli, 1>, <itiveliv, 1>, <tivelive, 1>, <ivelives, 1>, <positivel, 1>, <ositiveli, 1>, <sitiveliv, 1>, <itivelive, 1>, <tivelives, 1>, <positiveli, 1>, <ositiveliv, 1>, <sitivelive, 1>, <itivelives, 1>, <positiveliv, 1>, <ositivelive, 1>, <sitivelives, 1>, <positivelive, 1>, <ositivelives, 1>, <positivelives, 1>}

Frequent substring: Let x and s be two strings and x is a substring of s. For a given frequency threshold value θ, $\theta > 0$, x is called a frequent substring of s at threshold θ if $f_s(x) \geq \theta$.

As a special case, any substring x of s is a frequent substring of s at $\theta = 1$.

Frequent substring set: Let s be a string and θ is a given threshold value, the frequent substring set of s at threshold θ, denoted $FSS(s, \theta)$, is the set of all frequent substrings of s at threshold θ, i.e., $FSS(s, \theta) = \{ x \mid x \subseteq s \text{ and } f_s(x) \geq \theta \}$.

For example, from (1),

$FSS(s, 1) = SFS(s)$

$FSS(s, 2)$ = {<s, 2>, <i, 3>, <v, 2>, <e, 2>, <iv, 2>, <ve, 2>, <ive, 2>}

FSS(s, 3) = {<i, 3>}

3.3.1 Frequent substring indexing based on Vilo's technique

It is possible to use the suffix trie and suffix tree to enumerate and index frequent substrings. Firstly, the suffix trie or suffix tree needs to be completely constructed, then the constructed tree is traversed in the depth-first order to collect the number of occurrences of each substring. Finally, the substrings that occur at least at the given frequency threshold value θ on the string are extracted from the suffix trie or suffix tree.

The weakness of the above processes is that the space required for storing the full suffix trie or suffix tree can be large. Even with efficient implementation, the size of the suffix tree or suffix trie is on average 10 – 15 times the size of the string [76]. To reduce the space requirement for indexing the frequent substrings, Vilo introduced his algorithm and data structure, which will be described in the following sections.

Vilo's technique is interested only in substrings that occur at least at threshold θ times in the string. It seems that it is not necessary to construct the subtrees with less than threshold θ leaves. As a result, Vilo's algorithm only builds the part that contains frequent substrings. The algorithm is based on the suffix trie data structure. The construction procedure is inspired by the lazy algorithm [79] for generating a suffix trie. The algorithm is a generalization of the *wotd* (write-only top-down) suffix trie construction algorithm, to find the frequent substrings of a string. The resulting trie contains all frequent substrings. The nodes of the trie are labeled with the substrings. Labels on the path from the root to an internal node form the substring associated with that node. Thus each internal node represents a substring of the string and each terminal node represents a suffix of the string. The trie is called the pattern trie in Vilo's algorithm.

At each node, an occurrence list is maintained that contains the position of each occurrence of the substring corresponding to the node. The trie is generated starting from the root. The root corresponds to the empty pattern λ the occurrence list of which contains all character positions of the string. The trie is extended by generating the nodes in the trie in a systematic way. At each step, the children of some of the current leaf nodes are generated and inserted into the trie to make new leaf nodes. For a node *N* with associated substring ABC, every legal extension ABCD is generated by inserting a new child with label D under the node *N*. The occurrence list of ABCD is computed from the occurrence list of ABC by checking for each occurrence of ABC in the string to see if it can be extended to an occurrence of ABCD.

Each node *N* in the trie can be identified by the substring *x* that is the sequence of labels along the path from the root to the node *N*. This node *N* can be denoted by *N*(*x*). Hence, *N*(ABC) is the node identified by substring ABC, and *N*(*x*D) is the child of *N*(*x*) with character label D that equals *N*(ABCD). Every node in the trie contains additional information about its relation to other nodes in the tree. The dot-notation will be used to represent subfields—for example *N.parent*, *N.child*, *N.char* and *N.sibling*. The substring *x* is formed by the character labels *N.char* along the path from the root to the node *N*(*x*), *N*(*x*D).*char* = a_i that is the character label D where $a_i \in \sum$, and *N*(*x*D).*parent* = *N*(*x*). Given the node *N*, *N.child*(a_i) is used to denote the child *P* of node *N* so that *P.char* = a_i. A sibling of node *N* can be identified by the shorthand notation *N.sibling*(a_i), where *N.sibling*(a_i) is actually *N.parent.child*(a_i). Note that *N.sibling*(a_i) is the same as *N* if *N.char* = a_i. To keep the information about the occurrences of each substring, the lists of character positions of the string where the substring occurs are used. The occurrence list of substring *x* is stored in the node *N*(*x*) and denoted by *N*(*x*).*pos*. In addition, the frequency of substring *x*, $f_s(x)$, can be calculated from the number of substring positions.

Vilo’s algorithm starts by building the suffix trie for the input string *s* in a systematic order, for example in the breadth-first order, level by level. For each node *N*(*x*) create the list of positions *N*(*x*).*pos* containing each location of the string *s* where *x* occurs. To represent the occurrence that ends at character position *j* of the string *s*, a pointer is used to position *j*+1. To create the children of node *N*(*x*), find characters $a_i \in \sum$ for which the substring xa_i occurs in at least at θ different locations of the string *s*. This corresponds to counting which characters of ∑ occur at least threshold θ at the positions $N(x).pos$ of the string *s*. This can be done by one traversal of the position list $N(x).pos$ and creating simultaneously all the position lists for every character occurring at these positions in the string *s*. Only these nodes $N(xa_i)$ are inserted into the trie, for which the character a_i occurs at least threshold θ at positions $N(x).pos$.

Algorithm: Frequent substring generation:

Input: *String s of length n, and the frequency threshold θ for occurrences of substrings*

Output: *Suffix trie containing substrings that occur at least at θ in string s*

Method:

```
1.  Root ← new node; Root.char ← $\lambda$
2.  Root.pos ← (1,2,….., |s|)
3.  Enqueue(Q, Root)
4.  While N ← dequeue(Q)
5.          // Group the positions according to character in s
6.          foreach character $a_i \in \sum$
7.            Set($a_i$) ← Ǿ
8.          foreach character position of $a_i \in$ N.pos
9.            Add position + 1
10.         // Insert new child nodes for substrings that are sufficiently frequent
11.         foreach charater $a_i \in \sum$ that occur at least at $\theta$ in s, such that | Set($a_i$) $\geq \theta$
12.           N.child($a_i$) ← new node P with label P.char = $a_i$
13.           P.pos← Set($a_i$)
14.           enqueue(Q, P)
15.         delete N.pos
16. return Root
```

The trie is constructed by systematically extending the leaf nodes. Thus, the position lists are needed only for the leaves during the trie construction. An example of trie construction, in discovering frequent substrings from the string by using the above algorithm, is depicted in Figure 3.7.

Let string *s* = 'positivelives$' and θ =2

String *s* = p o s i t i v e l i v e s $

.pos = 1 2 3 4 5 6 7 8 9 10 11 12 13 14

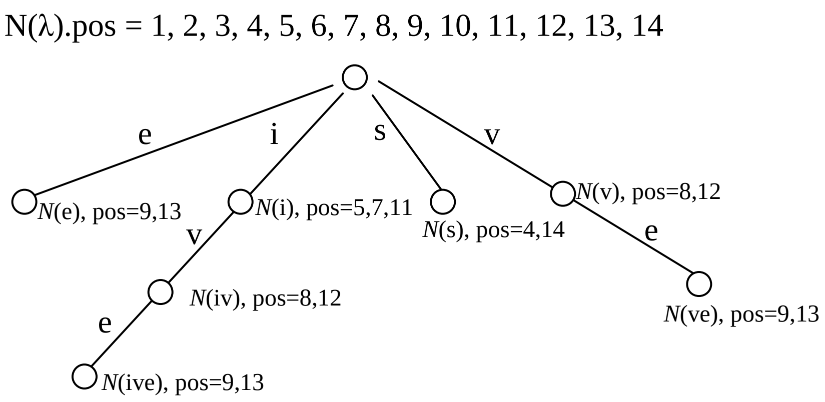


Figure 3.7. Discovering frequent substrings of string *s* = 'positivelives$' having at least two occurrences in string *s*. Nodes generated into trie represent substrings λ, e, i, s, v, iv, ve and ive

In Vilo's algorithm, each node in the trie represents a unique substring and contains the list of positions in the string where the substring occurs. From Figure 3.7, the algorithm generated six frequent substrings from string *s* as shown in Table 3.2.

Table 3.2. All frequent substrings with number of occurrences

Substring	Number of occurrences
e	2
i	3
s	2
v	2
iv	2
ve	2
ive	2

The strength of Vilo's algorithm is that this technique requires less storage space and construction time than the suffix tree and suffix trie for indexing the frequent substrings when $\theta > 1$. This is because the algorithm constructs only subtrees of the suffix trie that correspond to the frequent substrings to avoid enumerating all substrings.

In this part, an analysis of the time and space requirement of Vilo's algorithm is pesented. The total time used by Vilo's algorithm is proportional to the number of all frequent substrings [93], and for each frequent substring is proportional to the number occurrences where it traverses the string *s*. The space used by Vilo's algorithm is proportional to the number of all frequent substrings plus the position lists of the leaves and their parents. In case of $\theta = 1$, the time and space requirements of Vilo's algorithm for generating substrings of the *n* characters string *s* is $O(n^2)$, which is equal to the time and space requirement of the suffix tree and suffix trie as will be proofed on following.

> **Proof*:*** At every depth *l* of the trie, the work is proportional to the total size of the position lists of all nodes at that depth, i.e. $O(n)$. Hence, if $\theta = 1$, then Vilo's algorithm constructs the trie of depth *n*. Therefore, the time and space requirement for Vilo's algorithm is $O(n^2)$.

By raising the θ value, Vilo's algorithm can reduce the space requirement and speed up the construction time in practice. This is because the number of frequent substrings will be less when θ increases. When $\theta > 1$, the size of the trie is optimal in the sense that only the nodes $N(\alpha)$ corresponding to substrings α that occur at least θ, $\theta > 1$, in the string *s*, are inserted into the trie. As a result, the total size of all position lists of the nodes at any depth of the trie is $O(n)$. However, the size of the trie is not known before the trie is built. The worst-case space complexity of the algorithm therefore depends on the depth of the trie which is *O(nd),* where *d* is the depth of the trie. In the case of time complexity, the working time used for constructing the trie is also *O(nd).* This is because the working time is done at any depth *l* in the trie is *O(n)*. Hence, the overall time complexity is *O(nd)*, where *d* is the depth of the trie.

To determine the occurrence of a query, Vilo's technique uses the same technique that finds occurrences of the query on the suffix trie. However, Vilo's technique is faster than the conventional suffix trie. Once the frequent substring index has been constructed using Vilo's algorithm, the problem of whether the query occurs in the string can be answered in $O(m)$ time, where m is the length of the query. The search can be done by only one traversing of Vilo's trie to check whether the query appears on the trie. If it does, the number of occurrences and positions of the query on the string can be retrieved from the node corresponding to the query.

The comparison between Vilo's method and the suffix based method is explored as follows. As mentioned in previous paragraphs, the overall time and space complexity of Vilo's method is *O(nd)*, where *d* is the depth of the trie. Meanwhile, analysis of substring indexing based on suffixes is straightforward. The suffix trie and suffix tree are the tree of *n* depth, where *n* is the length of the string. These suffix tries and suffix trees are used for enumerating and indexing the complete set of substrings. Therefore, the construction of the suffix tree and suffix trie takes $O(n^2)$ time and space as shown in Table 3.1. Consequently, it is shown that Vilo's method has lower time and space complexity than the suffix trie and suffix tree in constructing the index, as Vilo's method constructs only the subtrees that correspond to the frequent substrings. In the case of time complexity for searching, Vilo's method takes only $O(m)$ time complexity.

This is because Vilo's method makes only one traversing on the trie to find the query that matches the substring on the trie and derives information about the occurrence of the substring from the node. Meanwhile, the suffix trie and suffix tree take $O(m + k)$ time complexity for searching, where k is the number of occurrences of the query in the string and m is the length of the query. These structures make the depth-first traversal to collect the number of occurrences of the query that match the substring on the tree. One traversal can be done in $O(m)$ time. The positions that can be enumerated by traversing the subtree is k, this takes $O(k)$ time complexity. As a result, the query can be searched in $O(m + k)$ time complexity on the suffix trie and the suffix tree.

Although Vilo's method was proposed for indexing only frequent substrings in order to reduce the space requirement, large storage space is still required for extracting and storing all frequent substrings. When it is directly applied to Thai text indexing [94], it tends to extract numerous insignificant or small indexing terms. This is because Vilo's technique finds the complete set of frequent substrings. However, it may not be necessary to extract all frequent substrings from Thai text documents if the frequent max substrings, which contain all frequent substrings, can be extracted from the text documents. This is one of the motivations of the proposed frequent max substring technique. To address the drawbacks of Vilo's technique, the proposed technique is used to extract indexing terms, known as frequent max substrings, from Thai text documents. The set of frequent max substrings are able to contain all frequent substrings without information loss and further scanning of the string. One of the strengths of this proposed method is that it retains a relatively smaller number of indexing terms without sacrificing its effectiveness in information retrieval when compared to Vilo's method. This is because each frequent max substring can potentially represent multiple frequent substrings that occur in the text documents. Thus, using the proposed technique for indexing text documents could provide better searching time as the search is performed on a smaller number of indexing terms extracted from the proposed technique. The proposed technique also uses the proposed data structure, called the frequent suffix trie. The proposed data structure has a more advantageous function than the conventional suffix trie and the pattern trie introduced by Vilo. Under the proposed data structure, the indexing terms can be extracted, together with their occurrence information and frequency, while the conventional suffix trie and the pattern trie extract only the indexing terms without frequency information [47], [76]. This means the proposed data structure has successfully been used to support extracting and containing of the indexing terms. In the next section, details of the proposed technique and the proposed data structure are described.

3.4 Frequent max substring technique

In this thesis, the frequent max substring technique is proposed to extract indexing terms from Thai text documents. Due to the rapid growth in the number of Thai electronic documents, the proposed technique may lead to significant savings in terms of storage space for storing indexing terms by extracting and indexing only the frequent max substrings. The proposed technique has also offered beneficial contributions in more efficient computation than language-dependent techniques. This is because the frequent max substring technique is proposed as a language-independent technique that does not rely on the use of a dictionary or corpus or complex language analysis for pre-processing. Unlike the word inverted index and n-gram inverted index, the proposed technique does not require query processing before the search can be performed.

The frequent max substring technique uses the analysis of frequent max substring sets to extract indexing terms as long and frequently-occurring substrings, called frequent max substrings, rather than individual words from Thai text documents. The proposed technique is different from Vilo's technique in that it focuses on extracting the frequent max substring set, thereby requiring less storage space for storing indexing terms than Vilo's technique, which extracts all frequent substrings as indexing terms. The frequent max substrings refer to all substrings which appear frequently at or above a given frequency threshold value, and have the maximum length of substrings in the given text. It is assumed that frequent max substrings are likely to be the terms of interest in a document. In the proposed technique, the length of substrings and term frequency are applied to reduce the number of substrings, which will be described in the algorithm section (Section 3.4.2). This technique uses two reduction rules: a) the reduction rule using a given frequency threshold value, $\boldsymbol{\theta}$, to check extracting termination, and b) the reduction rule using superstring definition to reduce the number of extracted substrings. The algorithm also uses a heap data structure to support computation while a queue data structure was used in Vilo's method [76]. In the proposed algorithm, the heap data structure is used to extract frequent max substrings, as this data structure is commonly used as a priority queue data structure that can satisfy some conditions. It will be described in more detail in Section 3.4.2.

The proposed method is used to extract the frequent max substring set without context consideration and is interested in substrings that occur frequently in Thai text documents, in order to reduce the number of insignificant indexing terms from the index. This is because indexing terms occurring less frequently in text documents could usually be assumed to be insignificant in defining subject matter [77], [69]. In this thesis, the new data structure, called the frequent suffix trie structure, FST structure, is also proposed to support extracting frequent max substrings as this proposed data structure can represents substrings, together with their occurrence and frequency information. The FST structure can be used to represent all substrings with their frequencies and list of positions on the string, meanwhile the suffix trie is the conventional data structure used to enumerate all substrings without their frequencies and list of positions. In addition, the proposed data structure is also different from the pattern trie introduced by Vilo, because the pattern trie enumerates substrings with their positions, but it does not keep the number of occurrences of substrings. In order to support the extraction of frequent max substrings, the FST structure is employed to ensure exhaustive enumeration of substrings. The frequent max substrings are then used as indexing terms, together with their number of occurrences and positions, to form the index. In the following sections, frequent max substring definitions are firstly provided, before details of the frequent suffix trie structure and the algorithm for extracting and indexing frequent max substrings will be described.

Frequent max substring definitions*:*

Max substring: Let x and s be two strings and x is a substring of s. The substring x is said to be a max substring of s if it satisfies the following condition: for any substring y of s, if $y \supset x$, then $f_s(y) < f_s(x)$.

Max substring set: Let s be a string. The max substring set of s, denoted $MSS(s)$, is defined to be the set of all max substrings of s.

For example, from (1),

$MSS(s)$ = {<s, 2>, <i, 3>, <ive, 2>, <positivelives, 1>} (2)

Frequent max substring: Let x and s be two strings. For a given frequency threshold value θ, $\theta > 0$, x is called as a frequent max substring of s if x is a max substring of s and $f_s(x) \geq \theta$.

Frequent max substring set: Let s be a string and θ be a given frequency threshold value, the frequent max substring set of s at threshold θ, denoted $FMAX(s, \theta)$, is the set of all frequent max substrings of s the substring frequency of which is at or above θ, i.e. $FMAX(s, \theta) = \{ x \mid x \text{ is a max substring of } s \text{ and } f_s(x) \geq \theta \}$.

For example, from (2),

$FMAX(s, 2)$ = {<s, 2>, <i, 3>, <ive, 2>}

3.4.1 Frequent suffix trie structure or FST structure

In the frequent max substring technique, only subtrees that correspond to frequent max substrings are constructed. The frequent suffix trie structure is the data structure employed to extract the frequent max substring set and to create the index at the same time. The basic concept of the frequent max substring technique is to enumerate substrings and record their frequencies and positions. Such substrings are then selected based on the given frequency threshold value. Therefore, it is necessary to have some efficient enumeration method that can be used to generate all substrings and their frequencies correctly from the strings. Consequently, the concept of the frequent suffix trie structure is proposed. The frequent suffix trie structure is similar to the suffix trie structure as both data structures are used to enumerate all substrings from the string by constructing all suffixes over the tree. Despite this, the frequent suffix trie structure additionally provides the list of positions and frequency information of all substrings. The frequent suffix trie structure has the following properties:

1. Let *suff*$_i$ = *s*[*i*, *n*] be the suffix of *s* starting at *i*th position and end at position *n*. A set of all suffixes of an *n*-length string *s* or *s*[*i, n*]*;* where *1* ≤ *i* ≤ *n*, is a set of substrings of string *s* that starts at position *i* and ends at position *n* [95].
2. The frequent suffix trie structure of *n*-length string *s* is a tree structure that represents all suffixes of string *s* starting with the root node and ending with *n* leaf nodes. Also '*$*' is appended at the end of string *s*. The terminating symbol'*$*' is added to show the end of string *s* and to make all suffixes of string *s* different from each other. Therefore, all suffixes of string *s* contain '*$*' at the *n* different ends of the frequent suffix trie structure. The frequent suffix trie structure also shows all substrings with their frequencies and position lists of any substrings of string *s*.
3. An edge is labeled with a symbol or a character that is an element of the character set. Every edge label has a length of exactly one. The labels of the edges connecting a current node with its children start with different characters. The concatenation of edge labels from the root to the leaf labeled *i* is *suff*$_i$
4. A node is used to represent a substring with frequency and list of positions (or *.pos*). The position is the end position of each substring of string *s*. Each depth of node leads to increased length of substrings and connecting to child substrings, child nodes. All leaf nodes keep suffixes with their frequencies and list positions of suffixes. All leaf nodes are also labeled with *i*, where *i* is the starting position of each suffix.
5. In the frequent suffix trie structure, the frequency of parent substrings, parent nodes, is always not less than the frequency of child substrings in the same path.
6. Child substrings are always a proper superstring of prior extracted substrings in the same path. The substrings can also be a proper superstring of smaller substrings in different paths.

3.4.1.1 The frequent suffix trie construction

The frequent suffix trie structure can be created in three steps. The following example shows the frequent suffix trie structure representing all substrings with their frequencies and list of positions of string *s* = ‘positivelives’.

Let string *s* = ‘positivelives’

1) Append ‘*$*’ to the string and define the position of each character in the string

Let string *s* =	p	o	s	i	t	i	v	e	l	i	v	e	s	$
Positions =	1	2	3	4	5	6	7	8	9	10	11	12	13	14

2) Enumerate all suffixes of the string

1. positivelives$
2. ositivelives$
3. sitivelives$
4. itivelives$
5. tivelives$
6. ivelives$
7. velives$
8. elives$
9. lives$
10. ives$
11. ves$
12. es$
13. s$
14. $

3) All suffixes are used to create a frequent suffix trie structure as shown in Figure 3.8.

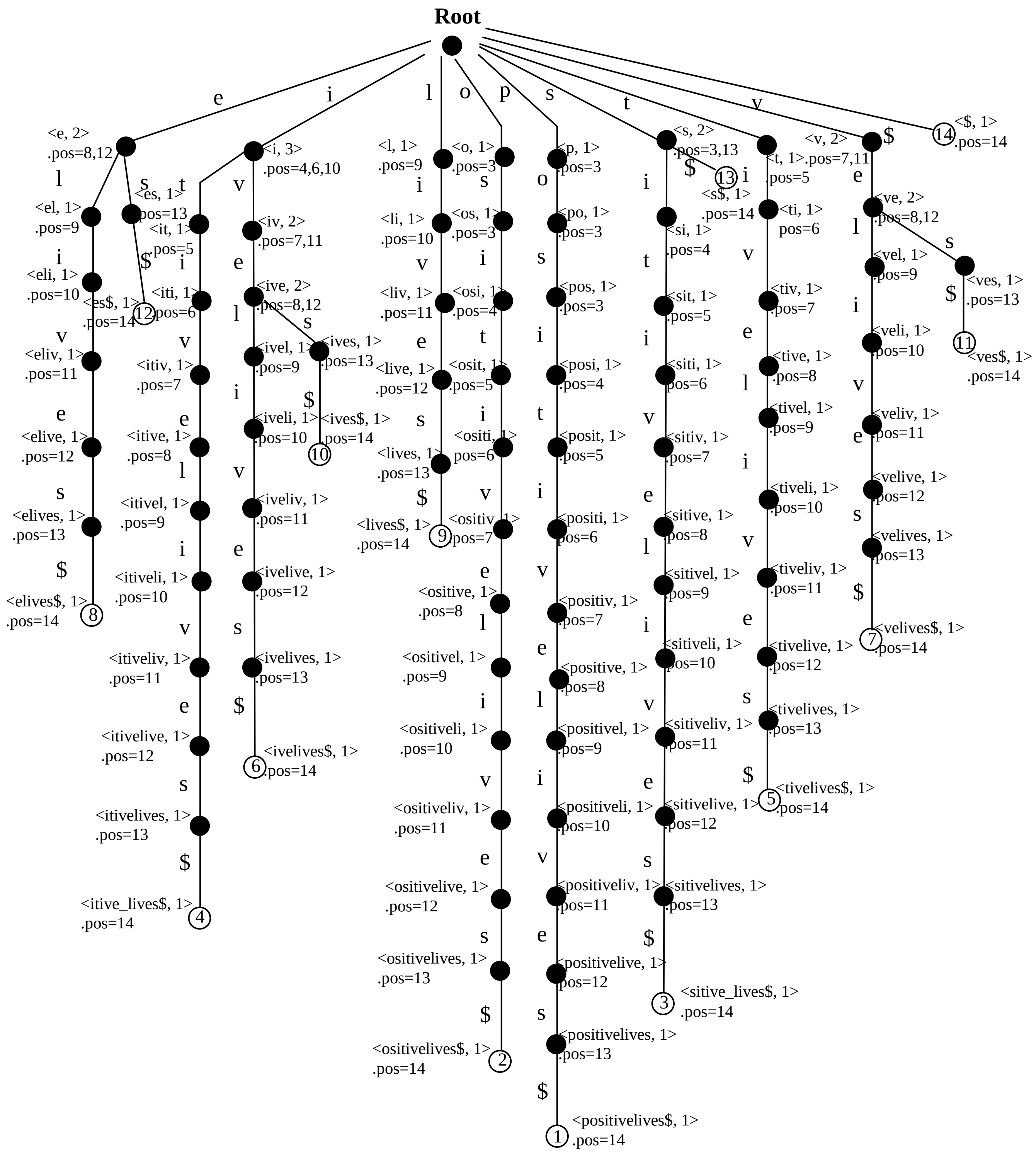


Figure 3.8. Frequent suffix trie structure for string *s* = 'positivelives$'

3.4.2 Algorithms

From Figure 3.8, the frequent suffix trie structure can be used as the data structure to extract frequent max substrings by using the following steps:

Step 1: Enumerate *SFS(s)* of string *s* using the FST structure.

Step 2: Extract frequent substring set or *FSS(s, θ)*

Step 3: Extract frequent max substring set or *FMAX(s, θ)*

From the above algorithm, the frequent max substring technique must first enumerate substring frequency sets or *SFS(s)*. If $|s| = n$, then $|SFS(s)| = O(n^2)$, because *SFS(s)* consists of *1*-length substrings to *n*-length substrings. A large amount of memory has to be used to keep numerous substrings in *SFS(s)*. Thus, a resolution to reduce memory is the reduction of the number of substring extractions in order to find the frequent max substrings, using two reduction rules:

a) Reduction rule using the given frequency threshold value, *θ,* to check extracting termination condition

b) Reduction rule using superstring definition

Rule a: Reduction rule using the given frequency threshold value, *θ* ,to check extracting termination condition.

From the property (5) of the frequent suffix trie structure, the frequency of a parent substring (node) is always not less than that of its child substring (node), because the parent substring is distributed to the child substring. Therefore, the enumeration of the child substring is terminated when the frequency of its parent substring is less than *θ* in the same path, and also its parent substring is deleted from the index.

Let x and y be two substrings on the same path on the frequent suffix trie structure, if y is a proper superstring of x, then the frequency of y or $f_s(y)$ is always less than or equal to the frequency of x or $f_s(x)$, i.e.

$$y \supset x \Longrightarrow f_s(y) \leq f_s(x)$$

Rule b: Reduction rule using superstring definition

From the property (6) of the frequent suffix trie structure, substring enumeration can be reduced by considering superstring definition. For example, let x be the substring of s that has length 1 and y is substring of s that has length 2, and also y is proper superstring of x. If the frequency of x is equal to the frequency of y where x and y are substrings in different paths, substring enumeration would be stopped in the x path, denoted as:

$$y \supset x \text{ and } f_s(y) = f_s(x) \Longrightarrow \text{Stop generating } x \text{ path}$$

In order to improve the algorithm, the frequent max substring technique is proposed. This technique uses the above two reduction rules to reduce storage requirements and the number of computations. The heap structure was employed to support computation for the proposed technique [47]. The heap structure is commonly used as a priority queue data structure that satisfies the heap property. The operations commonly performed with a heap are delete-max or delete-min. In the frequent max substring technique, min-heap is used as the data structure to support extracting frequent max substrings, using the operations insert and delete-min regarding the position of substrings (.pos). Therefore, it can insert, update and delete substrings on the min-heap structure in order to efficiently extract frequent max substrings. In the following section, extracting *FMAX(s, θ)* is shown as the following steps.

1. Enumerate the *1*-length substrings with their frequencies, and then select frequent substrings that occur at least at the given frequency threshold value. Substrings, frequency and position transactions (.pos) are kept in the min-heap structure and sorted by order of occurrence in the string. The position transaction (.pos) of substrings is used as a key to arrange substrings into the min-heap structure. Min position transactions mean more priority substrings are first inserted into the min-heap structure, because the prior substrings can be more frequent max substrings than later substrings.

2. Enumerate the child substrings of the first priority substring in the min-heap structure to process, and select only frequent child substrings. The min-heap structure will then be updated by using a deletion rule [95]. When each frequent child substring is enumerated every time, the deletion rule will be used to check, being a superstring of the frequent child substring, in order to remove existing substrings in the min-heap structure, which is a substring of the frequent child substring at the same frequency. If the frequency of existing substrings in the min-heap structure is equal to the frequency of superstrings, the existing substrings will be deleted from the min-heap structure and frequent child substrings are inserted into min-heap instead by considering two rules: (1) a substring is inserted into the min-heap structure, ordered by the occurring position on the string, (2) if the first position of the substring is equal to the first position of an existing substring in min-heap, a substring is inserted in the last position in the same group. The processed

substrings are deleted from min-heap. The other substrings are processed until the min-heap structure is empty.

3. Extract the frequent max substrings by selecting substrings having no superstring, from substrings in min-heap.

The next example shows the process of the algorithm using the min-heap structure and two reduction rules to reduce the storage requirement and the number of computations for frequent max substrings.

Let string *s* = ‘positivelives’

And the given frequency threshold value or $\theta = 2$

String *s*	=	p	o	s	i	t	i	v	e	l	i	v	e	s	$
Positions (.pos)	=	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Min-heap structure

Firstly, all substrings with a length of 1 are extracted, together with their frequencies and list of positions. The frequencies of these substrings are then checked in order to select only the frequent substrings with a length of 1. These frequent substrings are finally kept in the min-heap structure for further processes.

s, 2 .pos=3, 13	i, 3 .pos=4, 6, 10	v, 2 .pos=7, 11	e, 2 .pos=8, 12

<s, 2> is removed from the min-heap structure, and then child substrings of <s, 2> are extracted using the position transaction of <s, 2>. Child substrings consisting of<si, 1> and <s$, 1>. <si, 1> and <s$, 1> are not kept in the min-heap structure because their frequencies are less than the given frequency threshold value.

i, 3 .pos=4, 6, 10	v, 2 .pos=7, 11	e, 2 .pos=8, 12	

<i, 3> is removed from the min-heap structure, and then its child substrings are extracted using its position transaction. The child substrings of <i, 3> consist of <it, 1> and <iv, 2>. The existing substring in min-heap, <v, 2>, is deleted from the min-heap structure because <iv, 2> is a proper superstring of <v, 2> at the same frequency, and then <iv, 2> is kept in the min-heap structure instead of using the insertion rule because its frequency is equal to the frequency threshold value. The min-heap structure does not keep <it, 1> because its frequency is less than the given frequency threshold value.

iv, 2 .pos=7, 11	e, 2 .pos=8, 12		

Min-heap has <iv, 2> removed, and then its child substrings are extracted using its position transaction. The child substrings of <iv, 2> consisting of <ive, 2>. <e, 2> are deleted from min-heap because <ive, 2> is a proper superstring of <e, 2> at the same frequency, and then <ive, 2> is kept in min-heap instead of using the insertion rule, because its frequency is equal to the given frequency threshold value.

ive, 2 .pos=8, 12			

Min-heap has <ive, 2> removed, and then its child substrings are extracted using its position transaction. It consists of <ivel, 1> and <ives, 1>. They are not kept in min-heap because their frequencies are less than the given frequency threshold value.

The algorithm will stop when the min-heap structure is empty. That means all substrings in min-heap were detected and processed completely.

From the process, the frequent suffix trie structure can be shown as Figure 3.9.

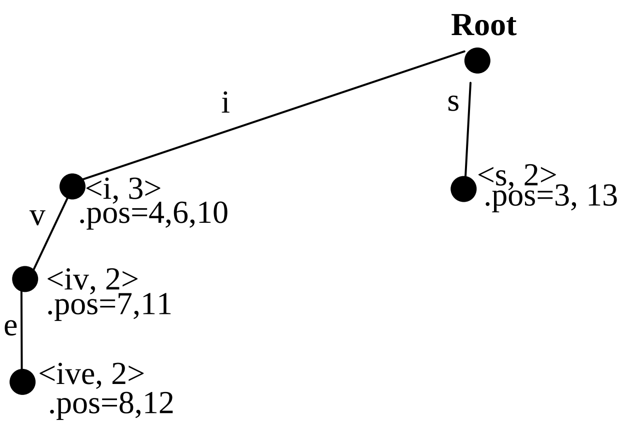


Figure 3.9. Frequent suffix trie structure using proposed algorithm

Figure 3.9 shows the FST structure using the proposed algorithm. The result is *FMAX*(*s*, *θ*) = {<s, 2>, <i, 3>, <ive, 2>}. From observation, the frequent max substring technique providea lesser number of indexing terms, but all possible frequent substrings can be derived from the set of indexing terms that is extracted by frequent max substring techniques without information loss.

The next example shows the frequent suffix trie structure representing all substrings with their frequencies and list of positions of the Thai language.

Let strings *s* = ‘การประกอบการ’

1) Append ‘$’ to the string and define the position of each character in the string

String *s* : ก า ร ป ร ะ ก อ บ ก า ร $

Position (.pos) : 1 2 3 4 5 6 7 8 9 10 11 12 13

2) Enumerate all suffixes of the string

1. การประกอบการ$
2. ารประกอบการ$
3. รประกอบการ$
4. ประกอบการ$
5. ระกอบการ$
6. ะกอบการ$
7. กอบการ$
8. อบการ$
9. บการ$
10. การ$
11. าร$
12. ร$
13. $

3) All suffixes are used to create the frequent suffix trie structure as shown in Figure 3.10.

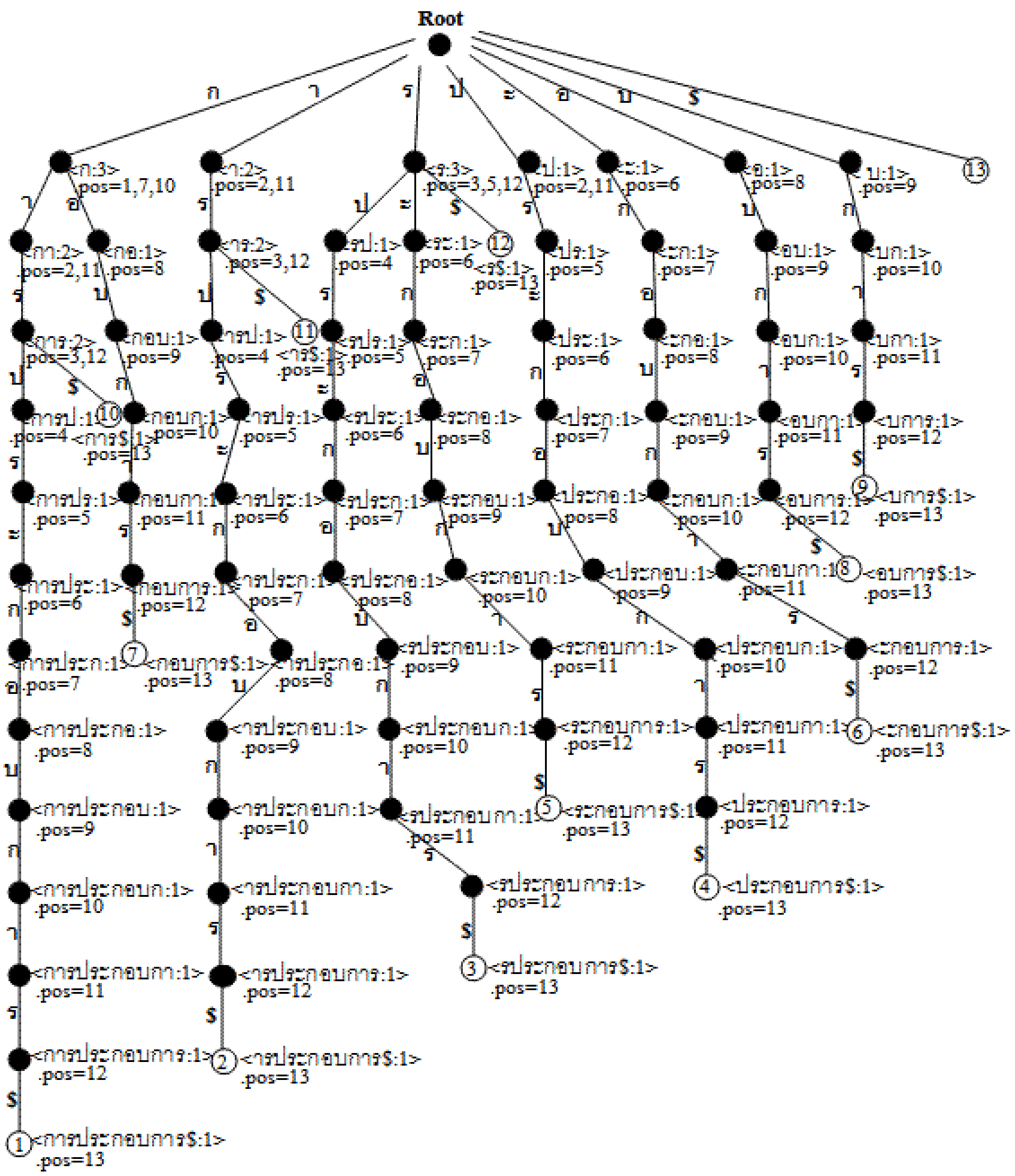


Figure 3.10. Frequent suffix trie structure for string *s* = 'การประกอบการ$'

The next example shows the process of the proposed algorithm using the min-heap structure and two reduction rules to reduce the storage requirement and the number of computations of frequent max substrings for the Thai language.

Let strings *s* = ‘การประกอบการ’

And the given frequency threshold value or *θ* = 2

String *s* :	ก า ร ป ร ะ ก อ บ ก า ร $
Position (.pos) :	1 2 3 4 5 6 7 8 9 10 11 12 13

Min-heap structure

Firstly, all substrings with a length of 1 are extracted together with their frequencies. The frequencies of these substrings are then checked in order to select only the frequent substrings with a length of 1. These frequent substrings are finally kept in min-heap for further processes.

ก, 3 .pos=1, 7, 10	า, 2 .pos=2, 11	ร, 3 .pos=3, 5, 12

<ก, 3> is removed from min-heap, and then its child substrings are extracted using its position transaction of <ก, 3>. Child substrings consist of <กา, 2> and <กอ, 1>. The min-heap structure has <า, 2> deleted, because <กา, 2> is a proper substring of <า, 2> at the same frequency and <กอ, 1> is not kept in the min-heap structure because its frequency is less than the given frequency threshold value. The min-heap structure

keeps <กา, 2> using the insertion rule, because its frequency is equal to the given frequency threshold value.

กา, 2 .pos=2, 11	ร, 3 .pos=3, 5, 12	

The min-heap structure has <กา, 2> removed, and then its child substrings are extracted using its position transaction. The child substrings of <กา, 2> consist of <การ, 2>. The min-heap structure keeps <การ, 2> using the insertion rule because its frequency is equal to the given frequency threshold value.

การ, 2 .pos=3, 12	ร, 3 .pos=3, 5,12	

The min-heap structure has <การ, 2> removed, and then its child substrings are extracted using its position transaction. They consist of <การป, 1> and <การ$, 1>. They are not kept in the min-heap structure because their frequencies are less than the given frequency threshold value.

ร, 3 .pos=3, 5, 12	

Min-heap has <ร, 3> removed, and then its child substrings are extracted using its position transaction. They consist of <รป, 1>, <ระ, 1> and <ร$, 1>. They are not kept in the min-heap structure because their frequencies are less than the given frequency threshold value.

The algorithm finishes when the min-heap structure is empty. That means all substrings in the min-heap structure were detected and processed completely.

From the process, the frequent suffix trie structure can be shown as Figure 3.11.

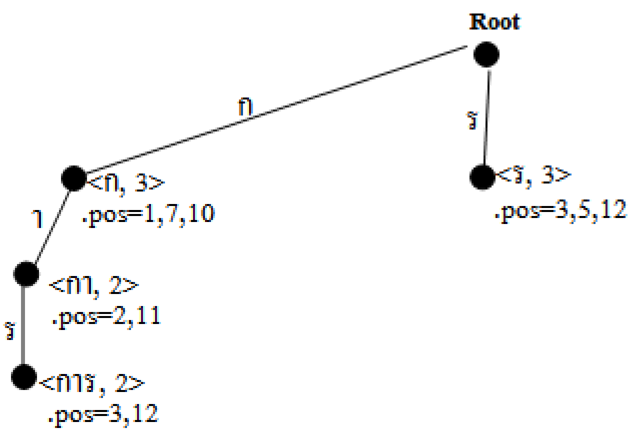


Figure 3.11. Frequent suffix trie structure using proposed algorithm for string *s* = ‘การประกอบการ$’

Figure 3.11 shows the FST structure using the proposed algorithm. The result is *FMAX(s, θ)* = {<ก, 3>, <ร, 3>, <การ, 2>}.

However, it is always possible to have more than one text document in the document collection. In order to construct the index for multiple documents, the frequent max substring technique is applied to the documents one by one in order to extract indexing terms and construct the document index. A merging technique is then applied to integrate document indices into one index.

For instance, after the first document has been processed, all indexing terms extracted from the first documents are stored in the index. The frequent max substring technique moves to the second document in order to extract the indexing terms and construct the index using the same process applied to the first document. Finally, the second document index is then merged to the first document index.

Assuming that there are *n* documents in the text document collection, this process goes through *n* pass. Once the *n* documents have been processed, all indexing terms, together with their occurrence information and frequency from the *n* documents, would have been contained in the index.

Note that the overlapping indexing terms from different documents are kept in the same record, using document indicating ID. Therefore, the document ID of the text documents may need to be stored in the index.

Figure 3.12 illustrates an overview of the general process on how to construct the index for multiple documents.

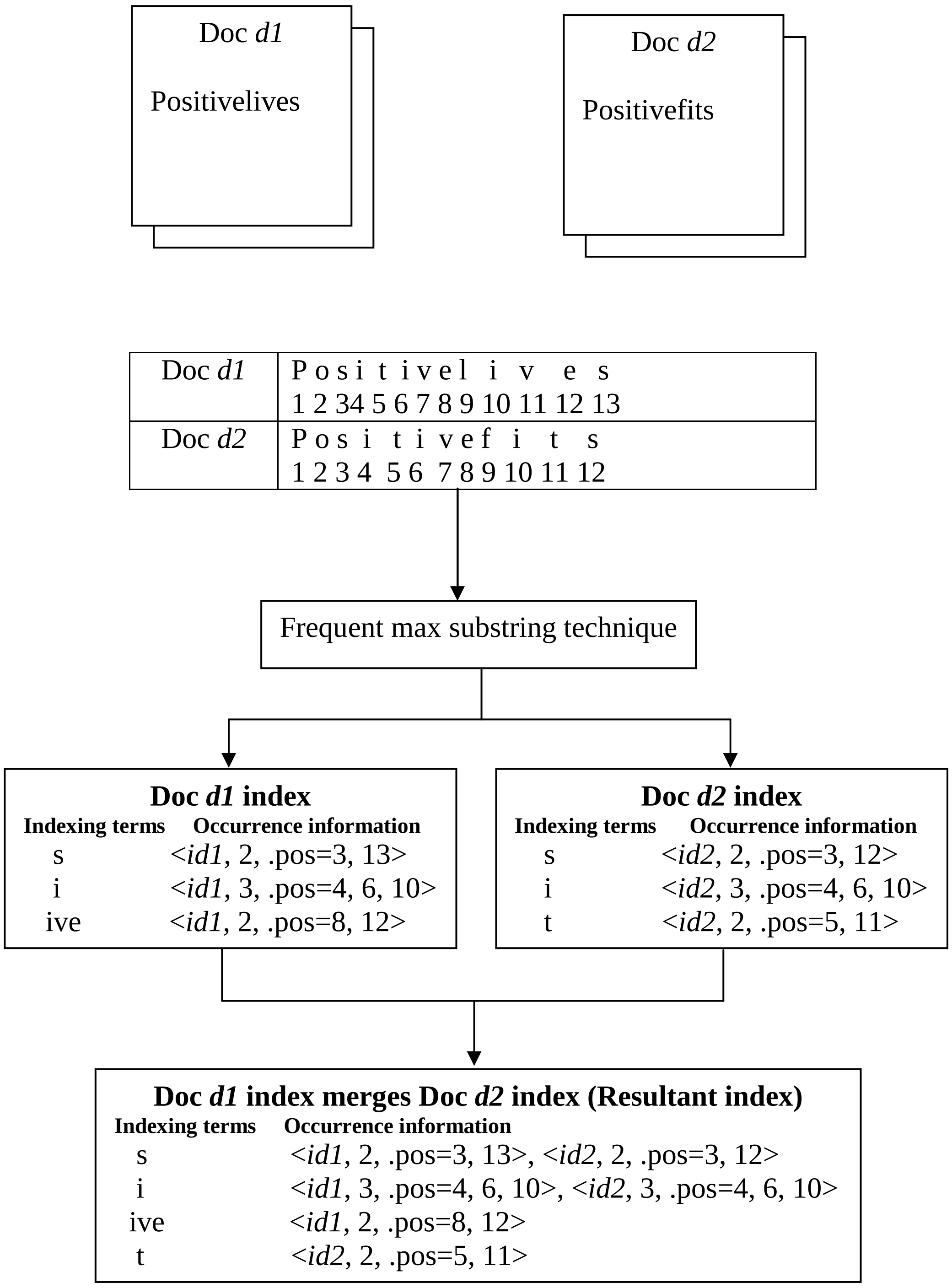

Doc *d1*
Positivelives
Doc *d2*
Positivefits
Doc *d1*
P o s i t i v e l i v e s
1 2 34 5 6 7 8 9 10 11 12 13
Doc *d2*
P o s i t i v e f i t s
1 2 3 4 5 6 7 8 9 10 11 12
Frequent max substring technique
Doc *d1* index
Indexing terms **Occurrence information**
s <*id1*, 2, .pos=3, 13>
i <*id1*, 3, .pos=4, 6, 10>
ive <*id1*, 2, .pos=8, 12>
Doc *d2* index
Indexing terms **Occurrence information**
s <*id2*, 2, .pos=3, 12>
i <*id2*, 3, .pos=4, 6, 10>
t <*id2*, 2, .pos=5, 11>
Doc *d1* index merges Doc *d2* index (Resultant index)
Indexing terms **Occurrence information**
s <*id1*, 2, .pos=3, 13>, <*id2*, 2, .pos=3, 12>
i <*id1*, 3, .pos=4, 6, 10>, <*id2*, 3, .pos=4, 6, 10>
ive <*id1*, 2, .pos=8, 12>
t <*id2*, 2, .pos=5, 11>

Figure 3.12. Example of indexing multiple documents using frequent max substring technique

The majority of the indexing techniques, such as the word inverted index and the n-gram inverted index techniques as described in Chapter 2, also used the same process when they are applied to index multiple documents [45], [1], [7]. This idea can also be applied to substring indexing and frequent substring indexing. However, the problem of substring indexing is that the data structures used in this technique do not keep occurrence information and frequency. Meanwhile the problem of frequent substring indexing is that the data structure used in this technique keeps only the occurrence information without frequency.

In the process of searching the query via the frequent max substring technique, the wildcard search is adopted for the search operation after the frequent max substrings are extracted. A wildcard is a character that may be used in the query to represent one or more characters. The most commonly used wildcard is an asterisk (‘*’). The asterisk can be used as a placeholder for any sequence of characters. For example, searching for the query ‘*per*’ would yield results which contain such indexing terms as ‘per’, ‘operation’, ‘permanent’ or ‘expert’. In the frequent max substring technique, the search first works by matching the query with the element in the set of indexing terms. If the query exactly matches the element in the set of indexing terms, then the search stops and returns the frequency and the position of the element. In contrast, if the query cannot be exactly matched with the element in the set of indexing terms, the query is formed in wildcard format such as ‘*retrieve*’. Then, this extended query is compared to the elements in the set of indexing terms, in order to find indexing terms that contain the extended query as their substrings. When all indexing terms that contain the extended query are retrieved, the frequency and positions will also be retrieved from the information that is collected about the occurrences, together with the indexing terms.

The strength of the proposed frequent max substring technique is that this technique improves significantly in terms of storage space over the suffix based method and Vilo’s method. The proposed technique retains a relatively smaller number of indexing terms without sacrificing its effectiveness in information retrieval. The details will be presented in Section 3.5. This is because each frequent max substring can potentially represent multiple frequent substrings that occur in the strings. This makes the proposed technique more storage efficient than substring indexing and frequent substring indexing. The proposed technique is also more efficient in computation when compared to language-dependent techniques, as the frequent max substring technique does not require pre-processing in segmenting the indexing terms like the word inverted index technique does. In addition, the frequent max substring technique also does not require pre-processing and query processing before indexing and searching can be performed, which provides more benefits than the word inverted index and n-gram inverted index techniques.

To demonstrate the strengths and weaknesses between the proposed technique and other techniques, experimental studies and comparison results on indexing Thai text documents are presented in the next sections.

3.5 Experimental studies

One of the basic requirements for evaluation is that results from different indexing techniques can be compared in terms of indexing efficiency and retrieval performance [96]. In this section, an experiment for indexing Thai text documents is shown. The objective of this experimental study is to demonstrate an experiment result for indexing Thai text documents using five different approaches, in order to compare indexing efficiency and retrieval performance. Suffix array, word inverted index, n-gram inverted index, Vilo's technique and the proposed frequent max substring techniques are investigated. From literature reviews, only a few papers work on the comparison of suffix array, word inverted index and n-gram inverted index techniques for indexing Thai text documents [1], [10]. In addition, Vilo's technique has never been applied and proved for indexing Thai text documents, although this technique was proposed to work on non-segmented texts like the genome sequence. Consequently, and while no one has made this comparison before, five indexing techniques will be investigated and compared in term of indexing Thai text documents in this thesis. The experimental study is divided into two main sections: evaluation of indexing and evaluation of retrieval performance, as will be described in Sections 3.5.2 and 3.5.3 respectively. In the evaluation of indexing, five indexing approaches are compared and evaluated in terms of the number of indexing terms, index size and indexing time. Meanwhile, the retrieval performance of five indexing approaches is evaluated by using standard precision and recall in the evaluation of retrieval performance section [7]. To make this comparison fair, the frequency threshold θ is set to 2 for the five indexing techniques in this experiment, which could ignore all words or substrings occurring only once in the texts, because one occurrence of words or substrings in the texts could usually not be taken as significant in defining the subject matter [77], [69] in Thai text.

3.5.1 Text collection

Unlike in English, standard data sets in Thai are not yet available for evaluating Thai text processing or indexing techniques [29]. However, in order to observe the performance of indexing techniques, a set of 50 Thai web pages with a size of 3.56 Mb is used as the text collection for the evaluation. In this experiment, 50 is chosen as the number of test web pages or documents in the text collection, because the number of documents normally used to evaluate Thai text processing techniques ranges from 10 to

80 documents or 1.25 Mb to 5.75 Mb [29], [48]. All Thai web pages used are Thai news websites, which consist of different content: sports, travel, education and political news as shown in Appendix A. The documents have varying sizes and lengths as also shown in Appendix B. The set of documents contains 103,287 characters, and average document length is 2,065 characters or 430 words per document. The basic statistics for the text collection are shown in Table 3.3.

Table 3.3. Basic statistics for Thai text collection

	No. of Docs	No. of Chars	No. of Words	Avg. Chars/Docs	Avg. Words/Docs
Sports news	15	24,727	5,447	1,648.46	636.13
Travel news	15	31,098	6,177	2,073.2	411.8
Political news	15	38,017	8,050	2,534.46	536.66
Education news	5	9,445	1,807	1,889	361.4

3.5.2 Evaluation of indexing

In indexing efficiency evaluation, the methodology used to compare the five indexing approaches is based on space and time efficiency for indexing Thai text documents. In space efficiency, the number of indexing terms which were enumerated by using five different approaches is compared, and index sizes used by the five approaches are also evaluated. The computational complexities used by the five indexing techniques are calculated in order to compare the time efficiency for indexing Thai text documents. In the n-gram inverted index, 3 is chosen to be the n, as most 3-gram indexing terms are meaningful in the Thai language. In the following sections, the comparison among five different indexing approaches is presented.

3.5.2.1 Space efficiency

In order to compare the five different indexing techniques for Thai text documents, the number of indexing terms which were extracted is first compared. The five indexing approaches used in the comparison are: suffix array, word inverted index, n-gram inverted index, Vilo's technique and the proposed frequent max substring techniques. In order to show the results clearly, the results from the five techniques are separated and shown in Figures 3.13, 3.14 and 3.15. The group of suffix array and the proposed frequent max substring technique is shown in Figure 3.13 and the group of Vilo's technique and the proposed frequent max substring technique is shown in Figure 3.14. Meanwhile, the group of word inverted index, 3-gram inverted index and the proposed frequent max substring technique is shown in Figure 3.15 as these three methods provided similar results and the results occur far from the first two groups. (For more data refer to Appendix C).

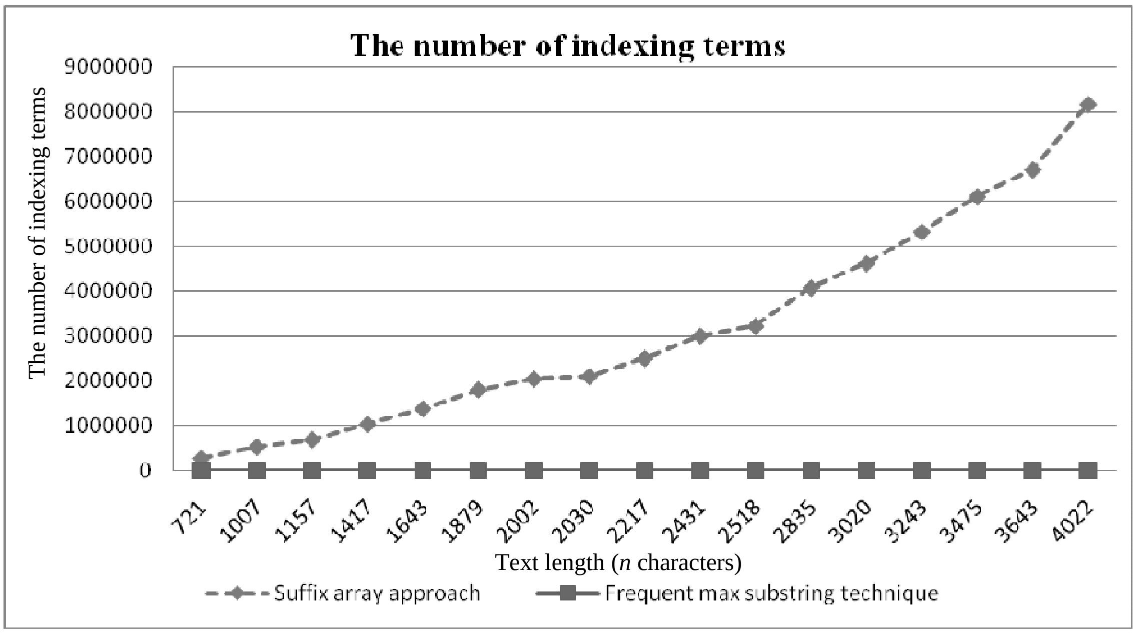


Figure 3.13. Graph of number of indexing terms extracted from two techniques: suffix array and proposed frequent max substring technique

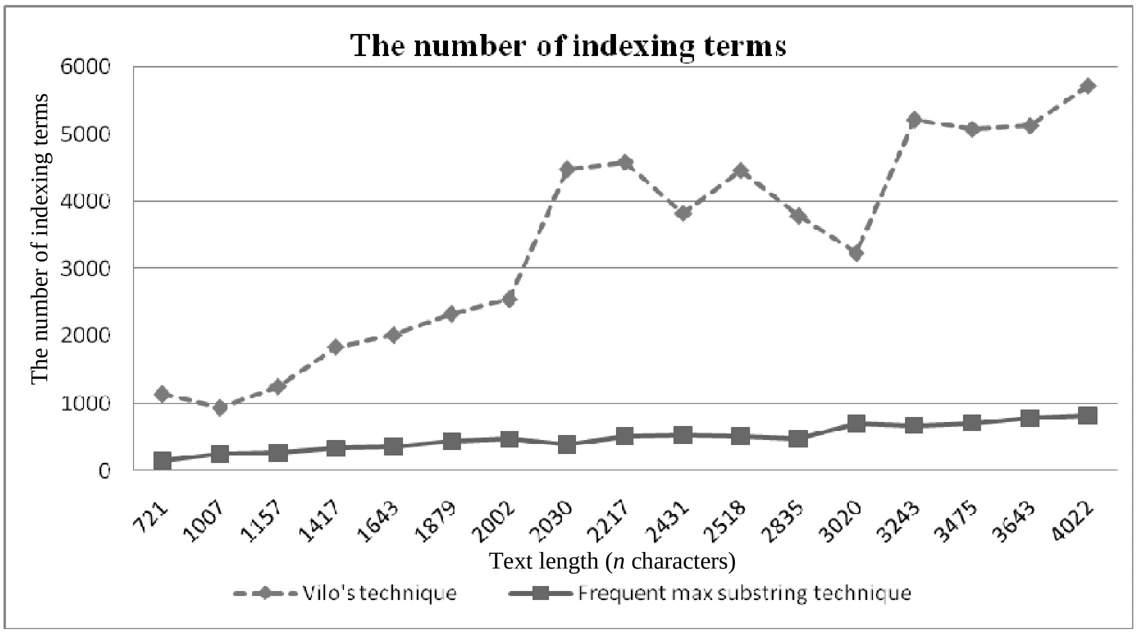


Figure 3.14. Graph of number of indexing terms extracted from two techniques: Vilo's technique and proposed frequent max substring technique

From Figure 3.13, it can be observed that the frequent max substring technique extracted a much smaller number of indexing terms when compared to suffix array. The frequent max substring technique also extracted a smaller number of indexing terms when compared to Vilo's technique as observed from Figure 3.14.

From observation, these three algorithms: suffix array, Vilo's technique and the frequent max substring technique can be used to retrieve all frequent substrings which occur at least at the given frequency threshold value, as described in previous sections. The indexing terms extracted by using these three techniques may or may not be meaningful as these are language-independent techniques. From the results crossing over Figure 3.13 and 3.14, the suffix array approach provides a much greater number of indexing terms than Vilo's technique and the proposed frequent max substring technique. The number of indexing terms which were extracted by the suffix array approach is the number of all substrings from the text documents. This is because the suffix array basically enumerates the complete set of the substrings from the given string before finding the number of occurrences of each substring. While for Vilo’s technique, the number of indexing terms generated is dependant on the size of the longest substrings that occur at least at the given frequency threshold value. In addition, the proposed frequent max substring technique provides a smaller number of indexing terms than the suffix array technique and Vilo’s technique. In the proposed frequent max substring technique, the algorithm enumerates only substrings that correspond to the frequent max substrings that contain all frequent substrings. Therefore, it uses less storage space for storing and extracting indexing terms. This could suggest that the proposed frequent max substring technique is more storage efficient when compared to suffix array and Vilo’s techniques.

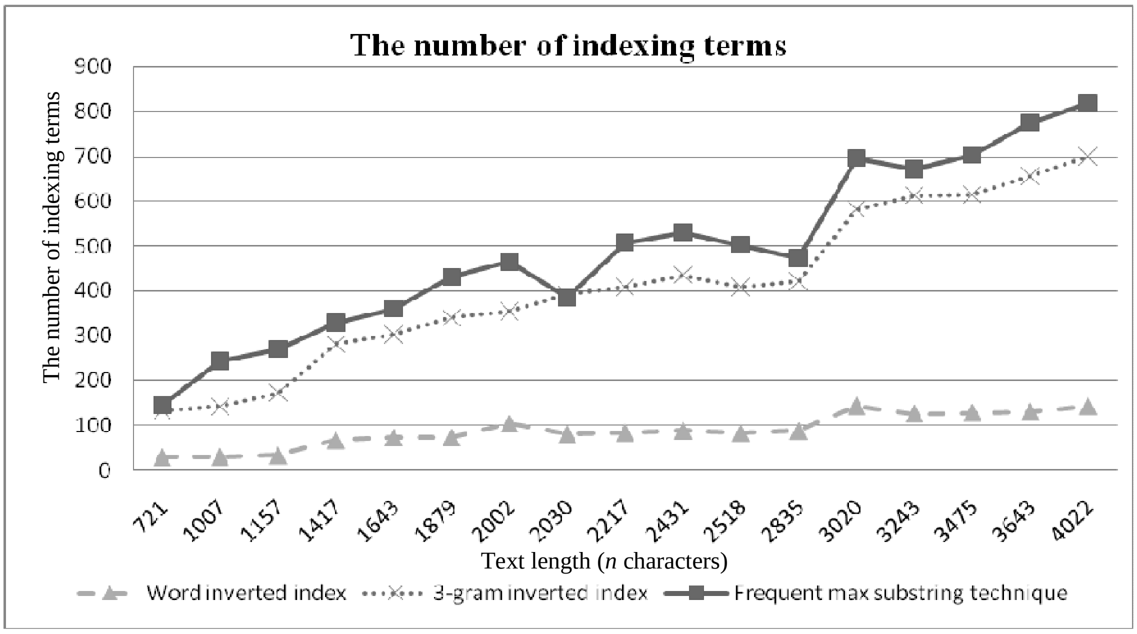


Figure 3.15. Graph of number of indexing terms extracted from three techniques: word inverted index, 3-gram inverted index and proposed frequent max substring technique

In the word inverted index method, all indexing terms are meaningful because this technique usually relies on language analysis or on the use of a dictionary. However, one of the drawbacks of this technique is that this method requires query processing and pre-processing in terms of segmentation, before searching and indexing can be performed. However, the ambiguous context of the Thai language is one of the major causes in degrading the efficiency of the parser. Additionally, the word inverted index technique sometimes loses the frequent substrings, as this technique uses word segmentation in extracting indexing terms as described in Chapter 2.

In the 3-gram inverted index approach, the indexing terms extracted by using this technique may or may not be meaningful. Despite this, using the 3-gram inverted index yields a smaller number of indexing terms than the proposed frequent max substring technique. Whereas, the disadvantage of the 3-gram inverted index is that it requires query processing before searching can be performed and query processing may lose the meaning of indexing terms. The *n*-gram inverted index also requires pre-processing to generate *n*-gram before indexing can be performed.

From Figure 3.15, it can be observed that the proposed frequent max substring technique provides slightly greater numbers of indexing terms than the word inverted index and the 3-gram inverted index techniques. It is worth noting that this proposed technique does not require any query processing or pre-processing before searching and indexing can be performed. Furthermore, most of the indexing terms extracted by the proposed frequent max substring technique are meaningful, as these indexing terms occur frequently in the text documents and most of them are noun phrases as will be described further in Section 3.5.3. However, the occurrence of indexing terms in text documents and the given frequency threshold value are the main factors impacting the number of indexing terms. The results show that the number of indexing terms increases when the text documents contain numerous small frequent substrings and the lengths of the frequent max substrings are short. In addition, the number of indexing terms also decreases when the given frequency threshold value increases.

For another experiment, a comparison of index size from five approaches is shown in Figures 3.16, 3.17 and 3.18, respectively (For more data refer to Appendix D). The comparison of index size refers to the storage space used by the five different approaches to index and store indexing terms and their pointers. Each technique requires a different index size, although they are performed using the same data collection. This is because the indexing terms which are extracted by the five approaches have varying lengths and pointers. Therefore, the comparisons between the number of indexing terms and index size are different in terms of memory. In this experiment, bytes are used as a measuring unit for comparing the index size used in the five different approaches.

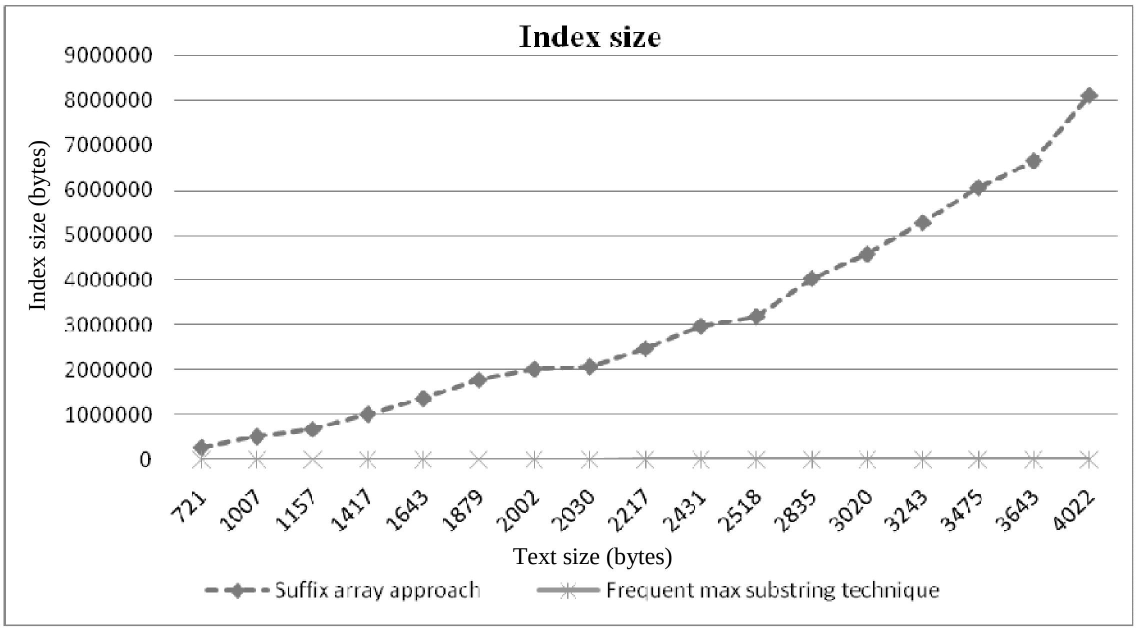


Figure 3.16. Comparison of index size from two techniques: suffix array and proposed frequent max substring technique

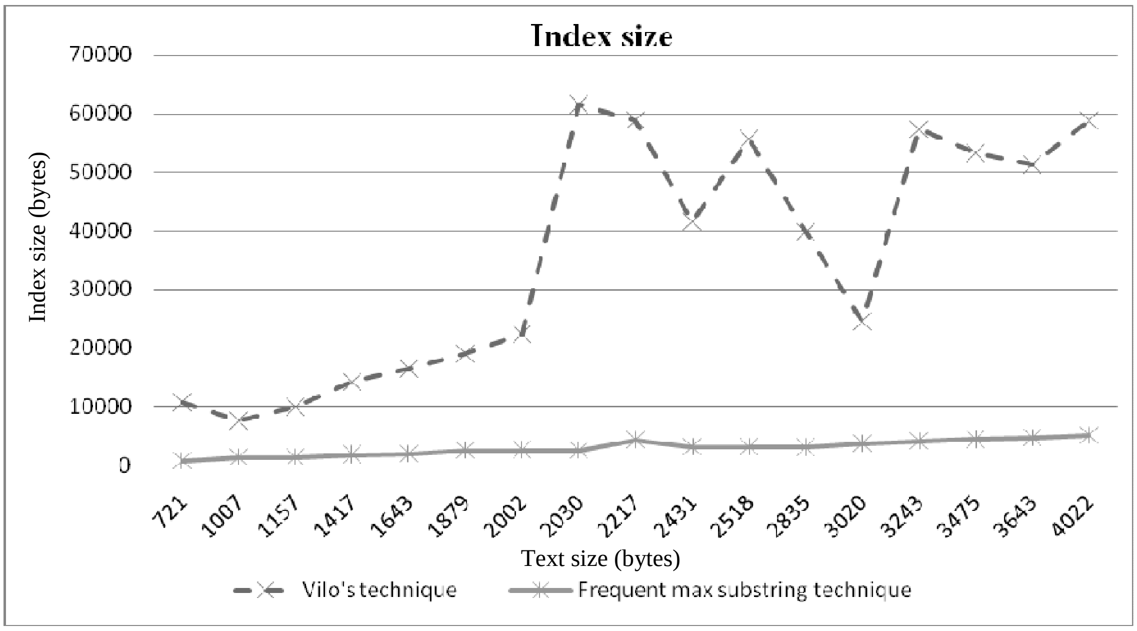


Figure 3.17. Comparison of index size from two techniques: Vilo's technique and proposed frequent max substring technique

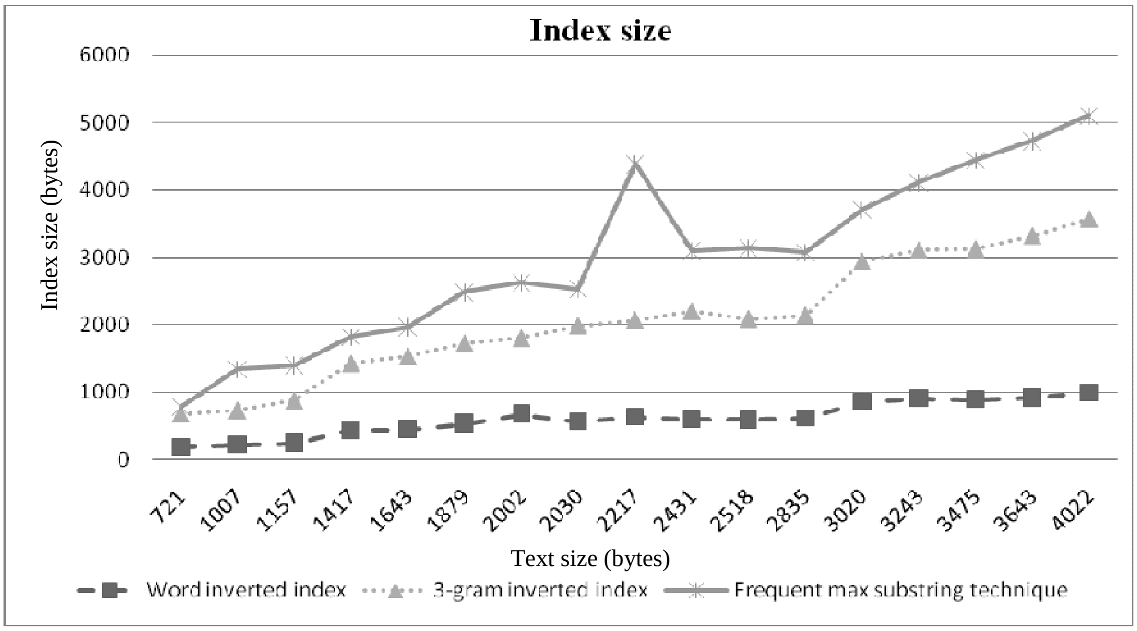


Figure 3.18. Comparison of index size from three techniques: word inverted index, 3-gram inverted index and proposed frequent max substring technique

From the evaluation results as shown in Figure 3.16, the suffix array approach obviously requires much more space to store indexing terms when compared to the proposed frequent max substring and other techniques. This is because this algorithm constructs a suffix array that contains all suffixes from the string, sorted alphabetically starting at position *i* in the string and continuing to the end of the string. As a result, one of the drawbacks in terms of index size in this technique seems to be very critical. Since the size of electronically stored information in the Thai language has grown exponentially, the method of suffix array is no longer practical for use in some applications, especially for a large collection of text documents, because the suffix array uses high storage space for containing all suffixes of the string, even when the string is short [1]. Meanwhile, Vilo's technique also requires more storage space when compared to the frequent max substring and other techniques, but less than the suffix array approach as shown in Figure 3.17. Although the given frequency threshold value can be used to reduce the index size, Vilo's technique still generates all possible frequent substrings from the given Thai text documents, which requires more storage space.

In the word inverted index, 3-gram inverted index and proposed frequent max substring techniques, index sizes used by these three approaches are quite similar, as shown in Figure 3.18. These three techniques also yield much smaller index sizes than the suffix array and Vilo's approaches. The proposed frequent max substring technique stores only frequent max substrings, thus this technique requires a small index size. However, the word inverted index and 3-gram inverted index approaches also require less index size than the proposed frequent max substring technique as these two techniques store the indexing terms in word and small substring levels. Although the proposed frequent max substring technique requires more index size, the benefit of this technique is that it does not need additional space to store a dictionary, corpus or manually hand crafted rules and does not require pre-processing to generate word and *n*-gram when compared to the word inverted index and n-gram inverted index, respectively.

3.5.2.2 Time efficiency

In this section, a comparative study of the five approaches in terms of indexing time is performed. The time used to index the text documents is calculated in process rounds dependent on the text lengths or *n* characters. Time complexities used in these five approaches are also examined in this section. Figures 3.19, 3.20, 3.21 and 3.22 show the experimental results in terms of indexing time (For more data refer to Appendix E).

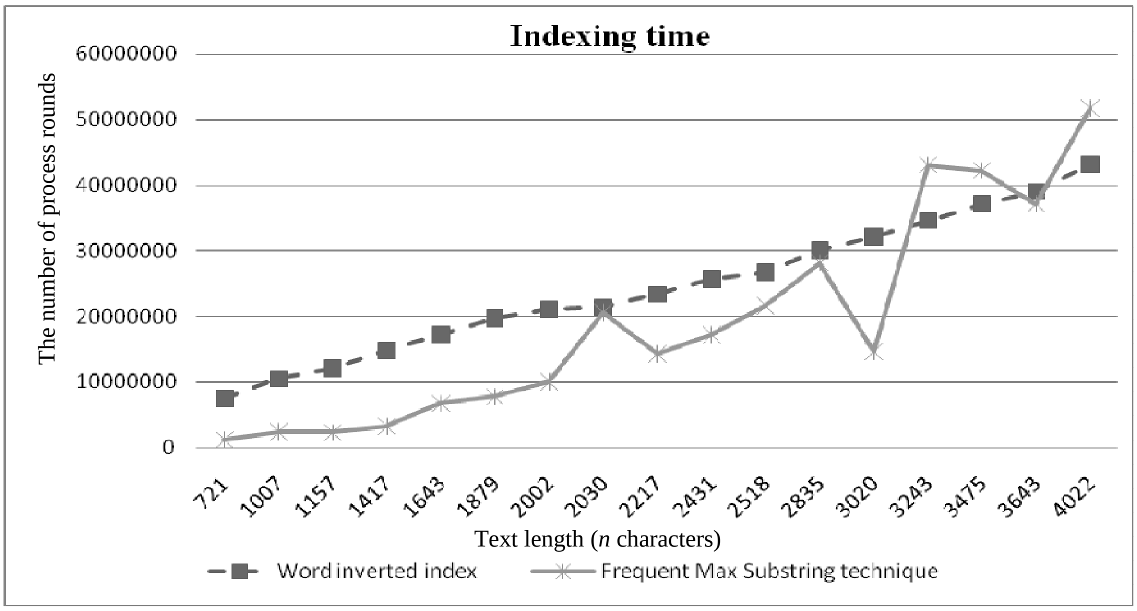


Figure 3.19. Comparison of indexing time of two techniques: word inverted index and proposed frequent max substring technique

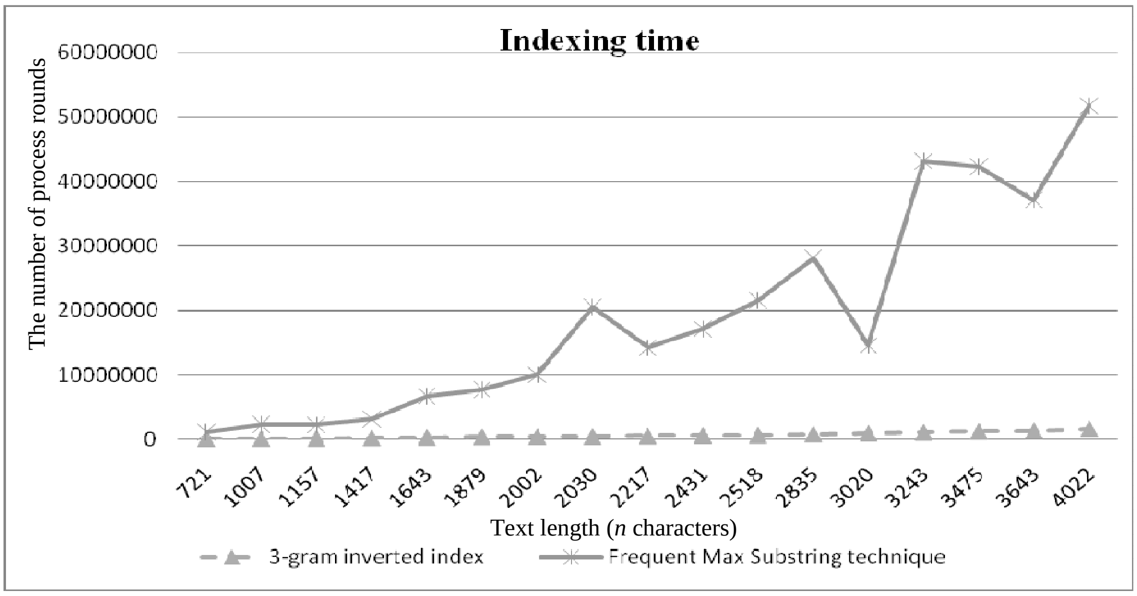


Figure 3.20. Comparison of indexing time of two techniques: 3-gram inverted index and proposed frequent max substring technique

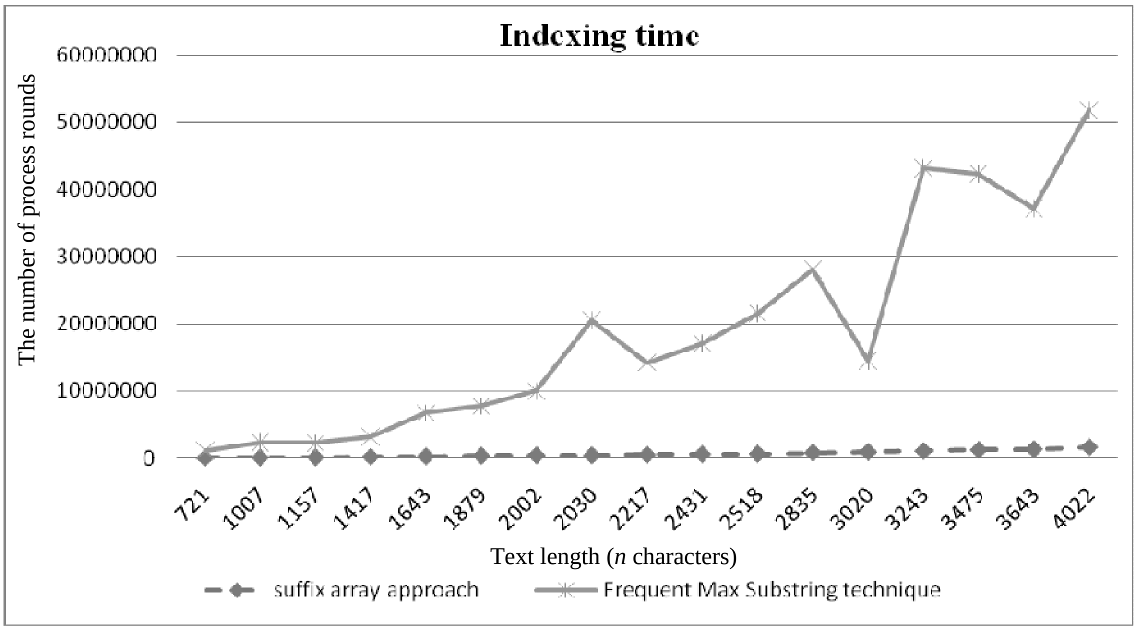


Figure 3.21. Comparison of indexing time of two techniques: suffix array and proposed frequent max substring technique

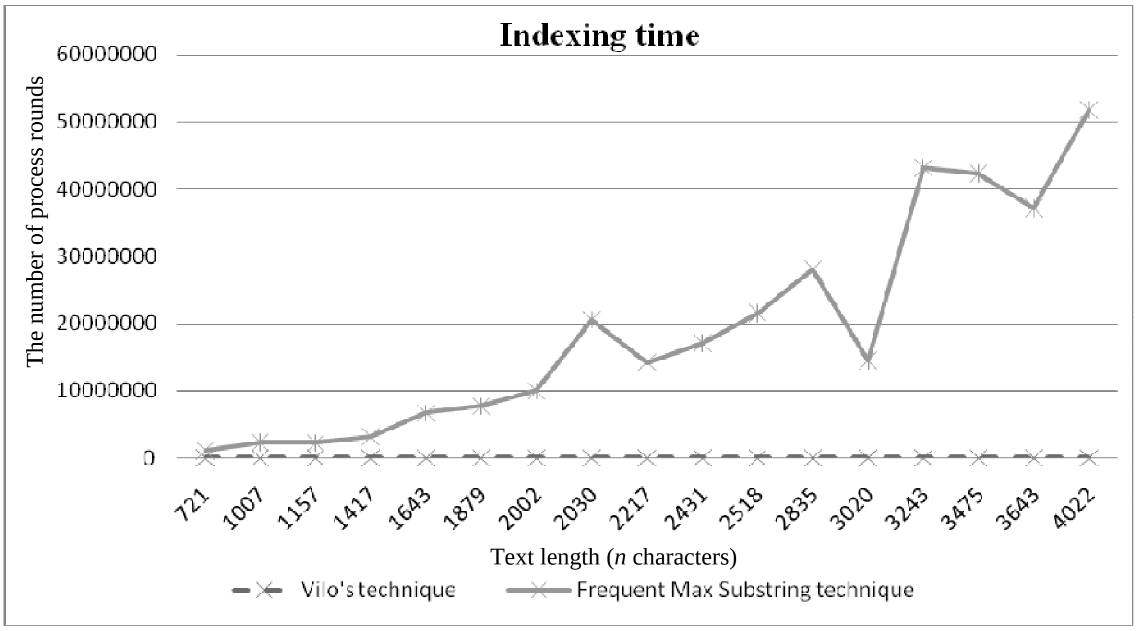


Figure 3.22. Comparison of indexing time of two techniques: Vilo’s technique and proposed frequent max substring technique

It can be observed from Figure 3.19 that the proposed frequent max substring technique uses less indexing time than the word inverted index technique. However, in some cases (at 3,243, 3,475, and 4,022 text lengths), the frequent max technique requires more indexing time than the word inverted index technique. This is because there is a higher number of frequent max substrings being kept in the min-heap structure during the extracting process from these three text documents. When the min-heap structure contains more frequent max substrings, more indexing time is required to check the substring status as described in Section 3.4.2. However, it is less likely that this case will happen in the proposed algorithm during the extraction of indexing terms, as it uses the two reduction rules. The two reduction rules are used to check the substring (indexing term) status in order to reduce the number of indexing terms in the min-heap structure. Meanwhile, Thai text documents need to be parsed and tokenized into individual terms before construction can be performed in the word inverted index. Therefore, this technique requires two processing times: the pre-processing and indexing times. The total time of the word inverted index technique is then $O(n|Dic|) + O(n^2)$. $O(n|Dic|)$ time complexity is required to parse words from a given Thai text document, by using a set of all possible words in the dictionary to match a given Thai text document for the segmenting process. $O(n^2)$ time complexity is required for constructing the inverted index. As a result, the word inverted index method takes $O(n|Dic|) + O(n^2)$ time complexity for indexing Thai text documents, where $|Dic|$ is dictionary size. Meanwhile the proposed frequent max substring technique does not require pre-processing time. However, the one of the drawbacks of the proposed frequent max substring technique is the indexing time required for constructing an index when compared to the other three techniques: suffix array, 3-gram inverted index and Vilo's technique. As can be observed from the results, the proposed frequent max substring technique requires more time to check the substring status for indexing. This is due to the use of the two reduction rules (as described in Section 3.4.2) to reduce the number of indexing terms. However, the indexing time is dependent on the given frequency threshold value and the size of maximum indexing terms. As mentioned, the proposed frequent suffix trie structure presented in Section 3.4.1 is used to perform indexing. The indexing time is therefore not known before the proposed frequent suffix trie structure is built, because it is dependent on the given frequency threshold value and the size of maximum indexing terms. For instance, at every depth of the frequent suffix trie structure, the indexing time is proportional to the total number of nodes and the time required to check the substring status of all nodes at that depth. Therefore, each depth requires $O(n^2)$ in time complexity. As mentioned, the indexing time is also dependent on the given frequency threshold value and the depth of the resultant frequent suffix trie or the size of maximum indexing terms. As a result, this method requires $O(n^2d)$ in time complexity where d is the size of maximum indexing terms.

In the 3-gram inverted index technique, Thai text documents need to be tokenized into 3-grams before construction can be performed. As a result, the 3-gram inverted index also requires two processing times: the pre-processing and indexing times. The 3-gram inverted index approach takes $O(n) + O(n^2)$ time complexity as $O(n)$ time complexity is required to generate all 3-grams from a given Thai text document and $O(n^2)$ time complexity is used for constructing the inverted index. As observed from the Figure 3.20, the 3-gram approach requires less indexing time than the proposed frequent max substring technique, although this technique needs pre-processing.

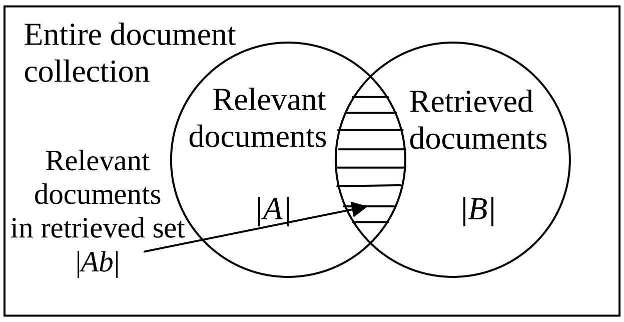
In Figure 3.21, it can be observed that the indexing time of the suffix array method increased slightly depending on text length. This technique first generates all suffixes of the string. Then all suffixes are sorted in alphabetical order to compute the frequency and location of indexing terms. As a result, this method takes $O(n^2)$ time complexity for constructing an index. Furthermore, the indexing time used to index the suffix array can be changed, dependent on sorting algorithms. In this experiment, Quicksort is used in sorting suffixes as it is the simplest efficient approach to building a suffix array.

From Figure 3.22, Vilo's technique takes *O(nd)* time complexity where *d* is the depth of the trie or the size of maximum indexing terms. In this technique, the pattern trie data structure is used to perform indexing. The indexing time is not known before the trie is built because it is dependent on the frequency threshold value and the depth of the trie. For instance, if the defined frequency = 1, the algorithm constructs the full suffix trie where depth *n* takes $O(n^2)$. In contrast, if the defined frequency = 2, the algorithm takes *O(nd)* time complexity where *d* is depth of the trie or the size of maximum indexing terms that occur at least at the given frequency threshold value.

It can be observed that the 3-gram inverted index, suffix array and Vilo's techniques performed well in terms of indexing time as these three techniques are straightforward and simple in constructing the index. The 3-gram inverted index, suffix array and Vilo's techniques take $O(n) + O(n^2)$, $O(n^2)$ and *O(nd)* time complexity respectively. Meanwhile the word inverted index and proposed frequent max substring techniques take $O(n|Dic|) + O(n^2)$ and $O(n^2d)$ time complexity respectively, where $|Dic|$ is dictionary size and *d* is the size of maximum indexing terms. These two techniques required more time than the other three methods to compute complex tasks, such as indexing term segmentation using dictionary matching and checking superstring definitions in the proposed frequent max substring technique.

3.5.3 Evaluation of retrieval performance

Retrieval performance is based on an ability to partition the text document collection into relevant and non-relevant documents, and into retrieved and not retrieved documents, according to what the user wants or queries as shown in Figure 3.23. Two most commonly used evaluation measures used to evaluate retrieval performance are precision and recall [7], [69]. Precision and recall were introduced in the Cranfield studies to summarize and compare search results [96]. In information retrieval, precision is defined as the ability to retrieve top-ranked documents that are mostly relevant. The formal definition of recall is the ability of the search to find all of the relevant documents in the text document collection. The definitions of these two measures assume that, for a given query, there is a set of documents that is retrieved and a set that is not retrieved. This obviously applies to the results of a Boolean search. If relevance is assumed to be binary, then the results of the query can be summarized as shown in Figure 3.23. In this figure, *A* is the relevant set of documents for the query, $\overline{A}$ is the non-relevant set, *B* is the set of retrieved documents, and $\overline{B}$ is the set of documents that are not retrieved. The operation ∩ gives the intersection of two sets. For example, $A \cap B$ is the set of documents that are both relevant and retrieved as shown in Figure 3.23.


Entire document collection
Relevant documents
Retrieved documents
Relevant documents in retrieved set |*Ab*|
|*A*|
|*B*|

	Relevant	Non-relevant
Retrieved	$A \cap B$	$\overline{A} \cap B$
Not retrieved	$A \cap \overline{B}$	$\overline{A} \cap \overline{B}$

Precision = Number of relevant documents retrieved / Total number of documents retrieved, can be denoted by $\frac{|A \cap B|}{|B|}$

Recall = Number of relevant documents retrieved / Total number of relevant documents, can be denoted by $\frac{|A \cap B|}{|A|}$

where |.| gives the size of the set.

Figure 3.23. Precision and recall for given example information request

From Figure 3.23, consider the example query *q* and the set *A* of relevant documents. Let |*A*| be the number of relevant documents in set *A*. Assume that a given retrieval strategy processes the query *q* and generates the retrieved document set *B*. Let |*B*| be the number of retrieved documents in set *B*. Therefore, the relevant documents in the retrieved document set *B* is in the desired document set *Ab* that satisfies the user's query, where |*Ab*| is the number of documents in the intersection of sets *A* and *B*. As a result, precision and recall values can be calculated from the number of documents in different sets. In other words, precision is the proportion of retrieved documents that are relevant. Recall is the proportion of relevant documents that are retrieved. There is an implicit assumption in using these measures that the task involves retrieving as many of the relevant documents as possible and minimizing the number of non-relevant documents retrieved.

In order to evaluate the retrieval performance of the five indexing techniques, the above standard precision and recall measures are used as measures of their performance. The five indexing approaches used in the evaluation are suffix array, word inverted index, n-gram inverted index, Vilo's technique and the proposed frequent max substring technique. In this experiment, eight queries are used as test queries that consist of four phrase queries and four single word queries as shown in Table 3.4. Most queries are noun words and noun phrases that can be used to define document types. All test queries are derived from the most frequent 100 queries that have been used by Thai users when using a search engine (for more detail, see http://business.truehits.net/keyfile/key_5.php). Phrase queries used in this experiment consist of two words, because the survey shows that two-word phrase queries are most often used when searching the web. The statistics were presented by OneStat.com under the title: ‘Most Searchers Have Two Words for Google’ (for more information, see http://searchenginewatch.com/3627479). The survey also shows that two-word phrase queries can normally satisfy users’ needs appropriately.

Table 3.4. All test queries consisting of four phrase queries and four single word queries

Phrase queries	**Single word queries**
การเมืองไทย(Thai politics)	คณะรัฐมนตรี(Cabinet)
เที่ยวไทย(Thai travel)	โรงแรม (Hotel)
ผลการแข่งขัน(Competition result)	นักกีฬา(Athlete)
การเรียนการสอน(Learning and teaching)	เรียน (Learn)

Again, 50 Thai web pages, which consist of the 15 sport, 15 travel, 5 education and 15 political web pages used in the experiment earlier, are used to measure precision and recall. Phrase queries and single word queries are used to test the retrieval performance of the five indexing techniques. In this experiment, the number of documents that is relevant to each query is known. For a given query, and a specific definition of relevance, retrieval performance can be defined as a measure of how well the search results correspond to what the user wants. This retrieval performance can be evaluated by considering precision and recall values. The following presents precision and recall values of the five indexing techniques as shown in Tables 3.5 to 3.9.

Table 3.5. Precision and recall values of word inverted index technique

Phrase queries			Single word queries		
Queries	**Precision**	**Recall**	**Queries**	**Precision**	**Recall**
การเมืองไทย (Thai politics)	0.66	0.66	คณะรัฐมนตรี (Cabinet)	0.33	1
เที่ยวไทย (Thai travel)	0.78	1	โรงแรม (Hotel)	0.54	1
ผลการแข่งขัน (Competition result)	0.5	0.5	นักกีฬา (Athlete)	0.48	1
การเรียนการสอน (Learning and teaching)	1	1	เรียน (Learn)	0.62	1
Average	0.735	0.79		0.4925	1

Table 3.6. Precision and recall values of 3-gram inverted index technique

Phrase queries			Single word queries		
Queries	**Precision**	**Recall**	**Queries**	**Precision**	**Recall**
การเมืองไทย (Thai politics)	0.54	0.66	คณะรัฐมนตรี (Cabinet)	0.28	1
เที่ยวไทย (Thai travel)	0.72	0.72	โรงแรม (Hotel)	0.54	1
ผลการแข่งขัน (Competition result)	0.66	0.5	นักกีฬา (Athlete)	0.42	0.86
การเรียนการสอน (Learning and teaching)	1	1	เรียน (Learn)	0.55	1
Average	0.73	0.72		0.4475	0.965

Table 3.7. Precision and recall values of Vilo's technique

Phrase queries			Single word queries		
Queries	**Precision**	**Recall**	**Queries**	**Precision**	**Recall**
การเมืองไทย (Thai politics)	0.7	0.77	คณะรัฐมนตรี (Cabinet)	0.33	1
เที่ยวไทย (Thai travel)	0.78	1	โรงแรม (Hotel)	0.54	1
ผลการแข่งขัน (Competition result)	0.6	0.75	นักกีฬา (Athlete)	0.48	1
การเรียนการสอน (Learning and teaching)	0.75	1	เรียน (Learn)	0.62	1
Average	0.7075	0.88		0.4925	1

Table 3.8. Precision and recall values of suffix array technique

Phrase queries			Single word queries		
Queries	**Precision**	**Recall**	**Queries**	**Precision**	**Recall**
การเมืองไทย (Thai politics)	0.7	0.77	คณะรัฐมนตรี (Cabinet)	0.33	1
เที่ยวไทย (Thai travel)	0.78	1	โรงแรม (Hotel)	0.54	1
ผลการแข่งขัน (Competition result)	0.6	0.75	นักกีฬา (Athlete)	0.48	1
การเรียนการสอน (Learning and teaching)	0.75	1	เรียน (Learn)	0.62	1
Average	0.7075	0.88		0.4925	1

Table 3.9. Precision and recall values of frequent max substring technique

Phrase queries			Single word queries		
Queries	**Precision**	**Recall**	**Queries**	**Precision**	**Recall**
การเมืองไทย (Thai politics)	0.77	0.77	คณะรัฐมนตรี (Cabinet)	0.33	1
เที่ยวไทย (Thai travel)	0.78	1	โรงแรม (Hotel)	0.54	1
ผลการแข่งขัน (Competition result)	0.75	0.75	นักกีฬา (Athlete)	0.48	1
การเรียนการสอน (Learning and teaching)	1	1	เรียน (Learn)	0.62	1
Average	0.825	0.88		0.4925	1

In order to compare the retrieval performance of different indexing techniques, Table 3.10 shows the average precision and recall values provided by the five indexing techniques.

Table 3.10. Average precision and recall values of five indexing techniques

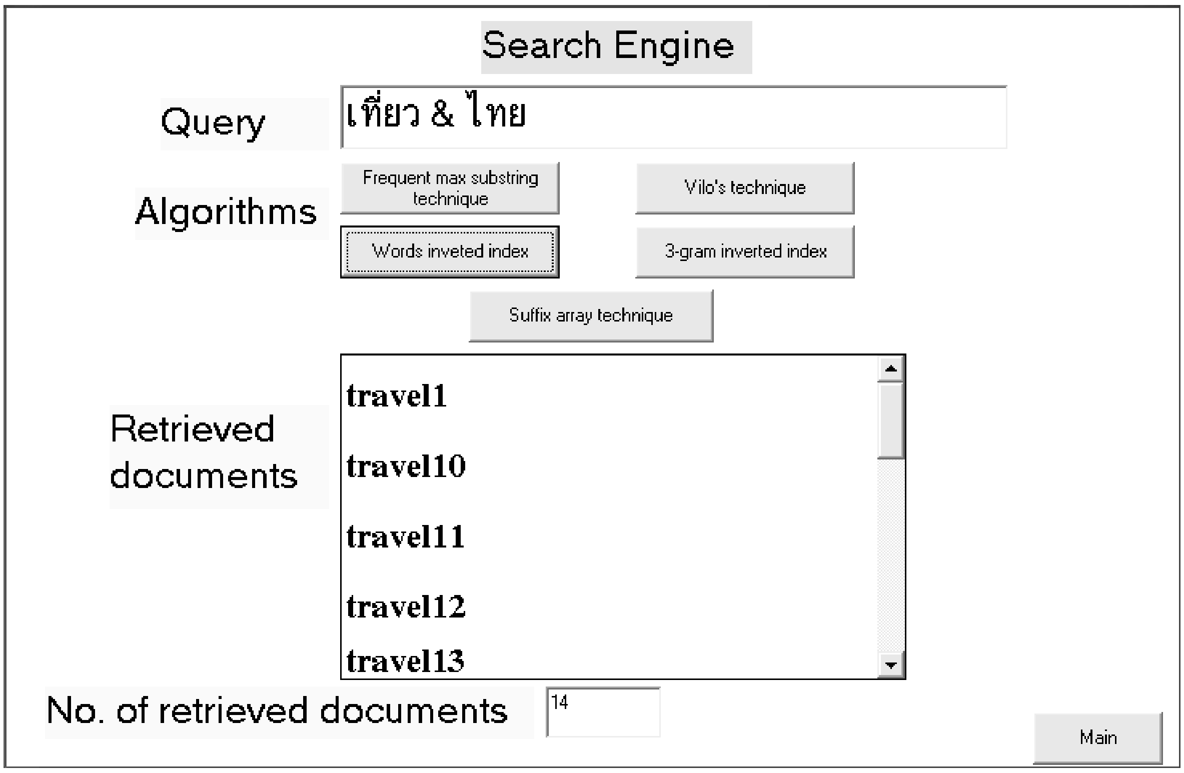
Average precision and recall values from five indexing techniques		
Algorithms	**Avg. Precision**	**Avg. Recall**
Frequent max substring technique	0.6588	0.94
Word inverted index technique	0.6138	0.895
Vilo’s technique	0.6	0.94
Suffix array approach	0.6	0.94
3-gram inverted index technique	0.5888	0.8425

As observed from Table 3.10, the frequent max substring technique provides the best average precision and recall values. This is because the frequent max substring technique is used to extract frequent and long indexing terms, known as frequent max substrings. These indexing terms can be used to define the subject matter of the query more clearly in the relevant documents. As mentioned earlier, the survey given by OneStat.com under the title: ‘Most Searchers Have Two Words for Google’ (http://searchenginewatch.com/article/2067569/Most-Searchers-Have-Two-Words-for-Google) shows that phrase queries usually provide better results than single word queries, in that they provide more relevant results to user’s queries. These phrase queries are normally contained as substrings in frequent max substrings. This is the reason why the frequent max substring technique provides better retrieval performance compared to other techniques. Meanwhile, Vilo’s technique and the suffix array approach provide the same average precision and recall values. This is because these two techniques are used to extract and keep all frequent indexing terms, as mentioned in the previous sections. Additionally, three indexing techniques: the frequent max substring technique, suffix array approach and Vilo’s technique provide the same recall values in this experiment. It can also be observed from Table 3.5 to 3.9 that the recall values of single word queries are higher than that of phrase queries for all techniques. This is because all techniques can basically be used to find relevant documents that contain the single word queries, and the single word queries generally occur more often than the phrase queries in the text collection. Due to this reason, the ability to find desired documents using single word queries is higher than using phrase queries. However, the frequent max substring technique provides better precision values than the suffix array approach and Vilo’s technique, because the number of retrieved documents searched by the frequent max substring technique is less than the number of retrieved documents searched by the other two techniques.

In searching phrase queries with the word inverted index technique, the phrase queries are required to be segmented into individual single words before searching. Following that, all single words are sent to the search process in order to find all documents that contain all segmented single words. Due to this reason, the word inverted index technique provides low precision and recall values, because single words are usually used as general terms rather than specific terms. In addition, in a single word query search, some relevant documents cannot be found with the word inverted index technique if the given single word query cannot match any indexing term in an index, because of the use of different word segmentation techniques.

When comparing different techniques, the 3-gram inverted index technique provides lower precision and recall values than other techniques. When using the 3-gram inverted index technique, the given query needs to be split into 3-gram terms. For example, if the given query is ‘โรงแรม’ or ‘hotel’, this query has to be segmented into ‘โรง’, ‘รงแ’, ‘งแร’ and ’แรม’. One of the drawbacks of splitting the query into 3-gram terms is that all 3-gram terms lose their semantic information and most of them are not meaningful. As a result, the 3-gram inverted index technique provides low precision and recall values. This is mainly because most of the split 3-gram terms cannot define what the user needs and thus cannot find the relevant documents.

Additionally, when queries were used to test retrieval performance, many advantages and disadvantages of each indexing technique were found. One of the main drawbacks of the word inverted index and 3-gram inverted index techniques are that these methods require query processing using segmentation, before searching can be performed. In searching phrase queries, the main problem of the word inverted index technique is that the phrase queries need to be segmented using the same word segmentation algorithm used for extracting indexing terms, so that an obtained set of words from the phrase queries can be used to query the text collection as shown in Figure 3.24.



Search Engine

Query: เที่ยว & ไทย

Retrieved documents:

travel1

travel10

travel11

travel12

travel13

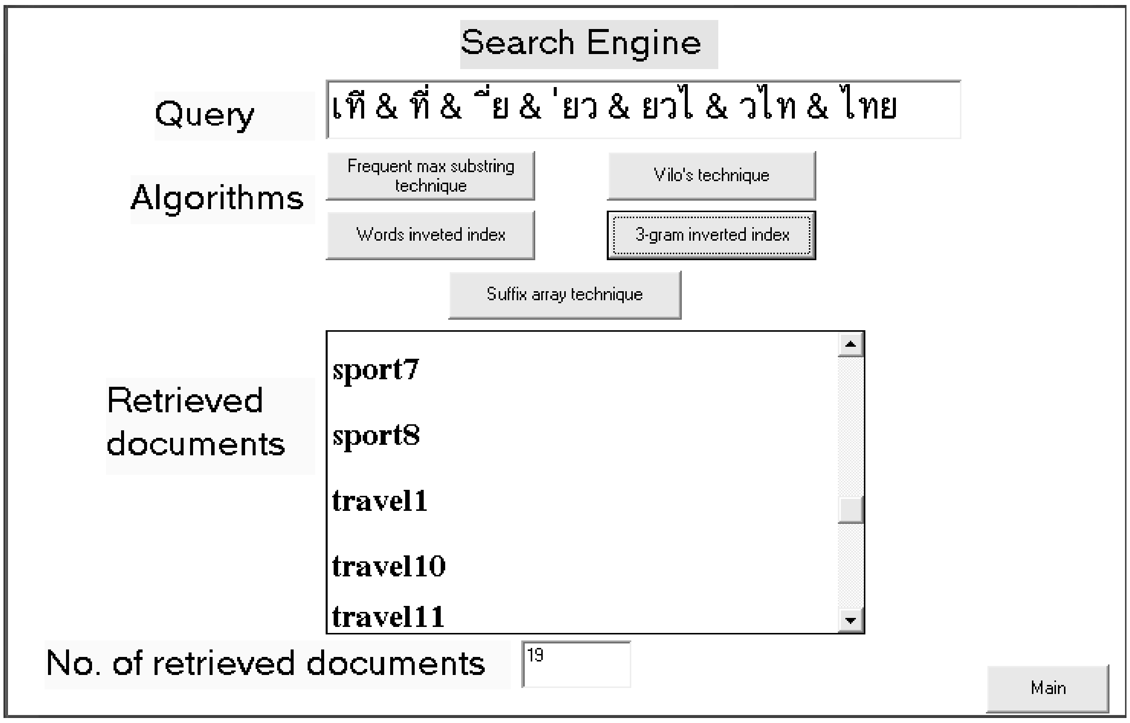
No. of retrieved documents: 14

Figure 3.24. Example of querying text collection by using a phrase query in word inverted index technique

From Figure 3.24, ‘เที่ยวไทย’ which means ‘Thai travel’ in English, was used as the phrase query to search relevant documents. This phrase query has to be segmented into ‘เที่ยว’ (travel) and ‘ไทย’ (Thai) before searching. As a result, ‘เที่ยว’ (travel) and ‘ไทย’ (Thai) terms were used to find all documents that contain these two terms. The Boolean operators such as ‘AND’ are needed for querying the text collection and this makes the query complex.

This problem also happens with the 3-gram inverted indexing technique. In the 3-gram inverted indexing technique, both phrase and single words queries need to be split into

3-gram terms by using the 1-sliding technique as mentioned in Chapter 2. For instance, if the given query is ‘เที่ยวไทย’ (Thai travel), ‘เที่ยวไทย’ (Thai travel) has to be split into seven 3-gram terms before a search can be performed as shown in Figure 3.25.



เที & ที่ & ี่ย & ่ยว & ยวไ & วไท & ไทย

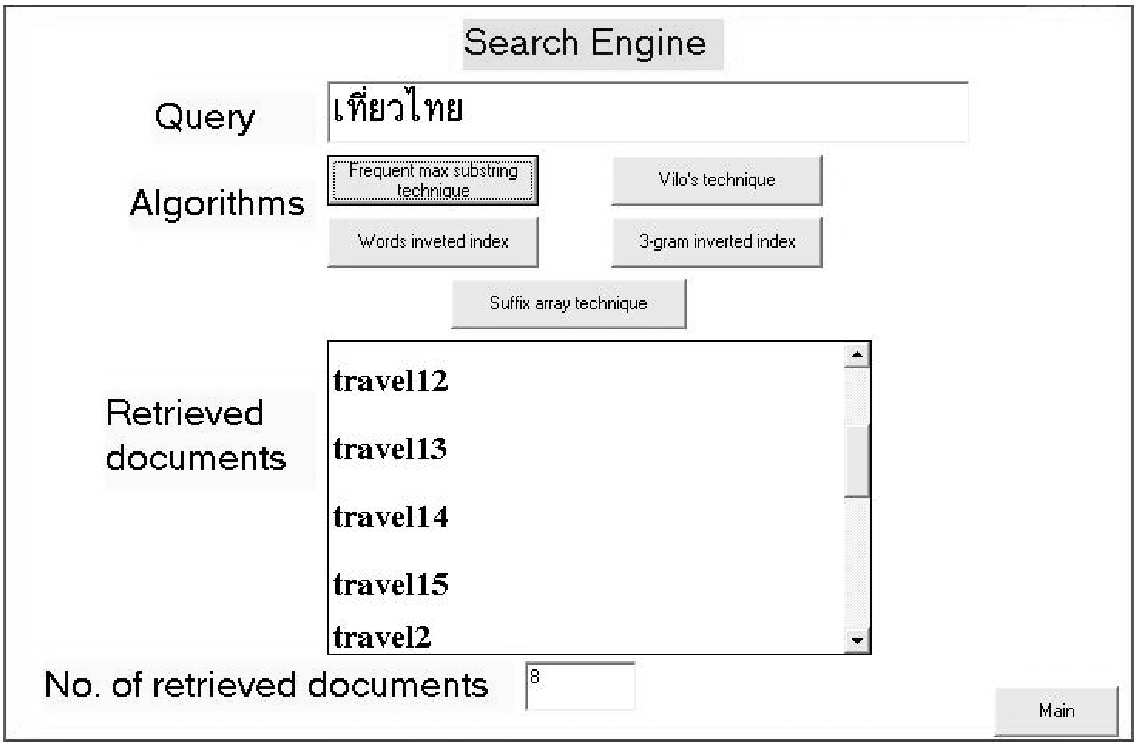
sport7
sport8
travel1
travel10
travel11

19

Figure 3.25. Example of querying text collection in 3-gram inverted index technique

From Figure 3.25, it can be observed that the given query was split into seven 3-gram terms and these terms were used to find documents which contain these split 3-gram terms. This could complicate the query after processing in the 3-gram inverted index technique. Hence, one of the drawbacks of the 3-gram inverted index technique is that this technique requires more resources, such as Boolean operation (AND etcetera), in searching.

In contrast, the advantage of the frequent max substring technique, Vilo's technique and the suffix array approach is that these three indexing techniques do not require pre-processing of the query before the search can be performed. In these three indexing techniques, the queries can directly be used for searching relevant documents that contain the queries as shown in Figure 3.26. This could be used as an option to search exact phrase queries from the documents, as these three indexing techniques can be used to find all frequent substrings from the documents. In addition, although phrase queries were segmented by any word segmentation technique or the 1-sliding technique used in the 3-gram inverted index technique, these three indexing techniques are still able to find the relevant documents as shown in Figure 3.27.

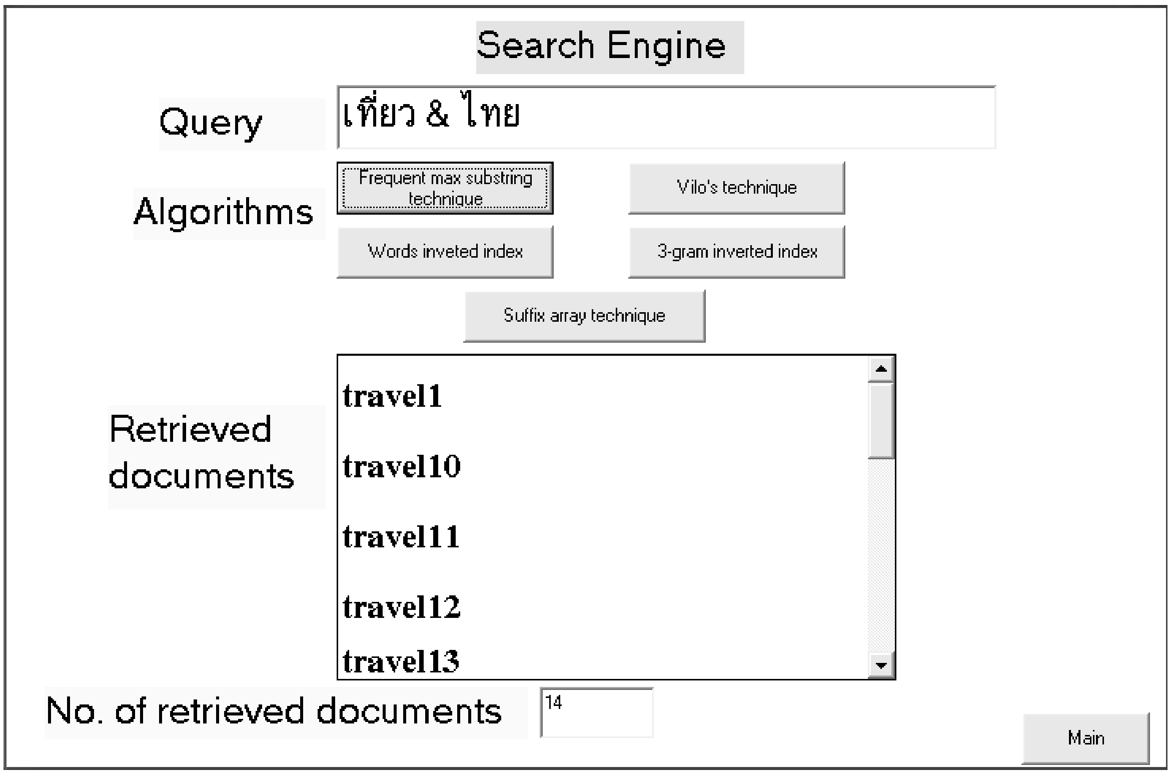


Query: เที่ยวไทย

Retrieved documents:
travel12
travel13
travel14
travel15
travel2

No. of retrieved documents: 8

Figure 3.26. Example of querying text collection using exact phrase query in frequent max substring technique



travel1
travel10
travel11
travel12
travel13

No. of retrieved documents 14

Figure 3.27. Example of querying text collection using a segmented query in frequent max substring technique

In Figure 3.26, ‘เที่ยวไทย’ (Thai travel) was directly used as the phrase query to search relevant documents in the frequent max substring technique and all documents which contain ‘เที่ยวไทย’ (Thai travel) were retrieved. Additionally, the frequent max substring technique also provides good results in terms of searching relevant documents even when ‘เที่ยวไทย’ (Thai travel) was segmented into single words: ‘เที่ยว’ (travel) and ‘ไทย’ (Thai) as depicted in Figure 3.27.

3.6 Conclusion

This chapter presents a proposed technique, called the frequent max substring technique, to extract indexing terms, known as frequent max substrings, for indexing

Thai text documents. The proposed technique is able to construct the index using less storage space to facilitate more efficient Thai text retrieval. The proposed technique used a proposed data structure, called the frequent suffix trie or FST structure, to ensure exhaustive enumeration of substrings to support extraction and storing of frequent max substrings. In practice, the heap data structure is employed to compute the frequent max substrings by using two reduction rules to reduce storage requirement and the time required for extracting frequent max substrings. The experiments were performed on indexing of 50 Thai web pages. The comparison results with different indexing techniques are also presented. Five indexing techniques are compared in terms of indexing efficiency and retrieval performance. From the indexing efficiency evaluation and comparison results, it can be observed that the proposed technique requires less space to store the indexing terms than the suffix array and Vilo's techniques. Meanwhile, the proposed frequent max substring technique provides a similar number of indexing terms when compared to the word inverted index and 3-gram inverted index techniques. In addition, the word inverted index, 3-gram inverted index and frequent max substring techniques yield similar index sizes, which are much smaller when compared to the suffix array and Vilo's approaches. However, one of the drawbacks of the word inverted index and 3-gram inverted index techniques is that these techniques require query processing and pre-processing before searching and indexing can be performed. Due to being language-dependent, the main problem of the word inverted index is that it requires word segmentation (i.e. the need for well-defined linguistic knowledge or the use of a dictionary or a corpus) in order to extract indexing terms, before constructing an index and searching the relevant documents. Like the word inverted index technique, the *n*-gram inverted indexing technique, which is language-independent, also requires the determination of the appropriate *n*-gram before any extraction or searching can be done. Meanwhile the proposed frequent max substring technique does not require any query processing and pre-processing. The proposed frequent max substring technique is also a language-independent technique that could be applied to many applications. In terms of time efficiency, one of the drawbacks of the proposed frequent max substring technique is the indexing time required for constructing an index, when compared to other techniques. This proposed technique requires more time to double check the conditions. However, the indexing time is dependent on the given frequency threshold value and the size of the maximum indexing terms.

Furthermore, the occurrence of the indexing terms in text documents and the given frequency threshold value are also the main factors impacting on the space and time efficiency of the proposed technique. In the retrieval performance evaluation, comparison results show that the proposed frequent max substring technique provides the best precision and recall values, because this proposed technique extracts and keeps frequent and long indexing terms, known as the frequent max substring. These frequent max substrings can normally better describe the content of documents. Meanwhile, Vilo's technique and the suffix array approach provide the same precision and recall values. Additionally, the frequent max substring technique, Vilo's technique and the suffix array approach provide good recall results, as these three techniques can find all documents that contain the query. As a result, these three techniques provided high recall value.

In the word inverted index and 3-gram inverted index techniques, query processing using segmentation is required before searching. The precision and recall values for these two techniques are subject to the quality of the segmentation, i.e. the splitting of general terms. These general terms normally cannot be used to exactly specify what the user needs and they are often not meaningful when used with the 3-gram inverted index technique.

Furthermore, another benefit of the frequent max substring technique, Vilo’s technique and the suffix array approach is that the given query can directly be used to search relevant documents without query processing before searching. In addition, the frequent max substring technique, Vilo’s technique and the suffix array approach are also able to search relevant documents even when the query was segmented into single words. Meanwhile, the word inverted index and 3-gram inverted index techniques require query processing for all cases.

Chapter 4

Hybrid Method: Integration of the Frequent Max Substring Technique and Thai Language-Dependent Technique

4.1 Introduction

The objective of this chapter is to show that the frequent max substring technique presented in Chapter 3 can also be combined with other Thai language-dependent techniques to become a novel language-dependent technique. This hybrid method is based on the integration of the frequent max substring technique and a Thai language-dependent technique for extracting and indexing meaningful indexing terms from Thai text documents. In order to see whether the hybrid method provides significant benefits for indexing Thai text documents, the hybrid method is compared with the word inverted index technique in terms of the number of indexing terms and retrieval performance, as the word inverted index technique is the more widely used language-dependent technique for Thai text indexing [1], [7], [34], [44]. Before presenting the hybrid method, some related works are described in Section 4.2.

4.2 Related works

As outlined in Chapter 2, the word inverted technique is the more widely adopted method used for indexing Thai text documents [1] ,[7], [34], [44]. This technique relies mainly on the outcome of word segmentation techniques [97], [98], [4]. The objective of word segmentation techniques is to extract a set of words contained in the text documents. Keywords are then selected from this set of words in order to create the index for the corresponding text documents as described in Chapter 2. However, due to the limitations of word segmentation techniques and the difficulty in Thai phrase or sentence extraction, only a few significant works on Thai information extraction are available and will be reviewed as follows.

In [99], the study considers the generation of substrings from text corpus of non-segmented languages and focuses on natural language processing (NLP) issues such as morphology and part-of-speech (POS) tagging.

In [98], the alternative method for extracting important keywords from categorized text corpora was proposed. This method was referred to as automatic categorized keyword extraction (ACKE). The algorithm is based on the analysis of frequent substring-sets for mining sequential patterns. The ACKE algorithm is composed of two main steps: (1) a process of generating frequent substrings, which satisfies some constraints, and (2) a process of merging those frequent substrings into keywords. Therefore, applying the ACKE algorithm to a text corpus generates keywords which are highly distinctive in indexing documents.

In sentence segmentation, the tri-gram model was adopted. Current research into sentence boundaries can be found in the paper by Mittrapiyanuruk P. and Sornlertlamvanich in 2000 [100]. Their proposed algorithm was used to extract sentences from a paragraph by detecting the true sentence breaking spaces. The statistical part-of-speech (POS) tagging technique was applied to the space classification problem in this technique. The algorithm considers two consecutive strings with a space in between each time to determine whether the space is a true sentence breaking space. This approach was evaluated by the ORCHID corpus [55], which is a part-of-speech tagged corpus. However, from the evaluation, this approach cannot generate high segmentation accuracy and segmentation of sentences is still a complex task. The segmentation of Thai sentences is difficult mainly due to the reason that the definition of segmenting cannot be fully defined.

In [101], a method for paragraph extraction based on extracting Thai compound nouns was proposed. It is known that compound nouns that occur frequently could carry more semantic information. Therefore, Thai compund nouns play an important role in Thai language processing. For this reason, this approach focused on compound nouns extracted by using part-of-speech (POS) tagging. Furthermore, the term weighting technique was used to calculate term frequency and document frequency, and to score all words by removing stopwords in the paragraph. The similarity between paragraphs was measured according to cosine similarity value. The results were used to select a representative paragraph among similar paragraphs.

4.3 Hybrid method

In this section, the hybrid method that combines the frequent max substring technique and Thai language-dependent technique is described. The hybrid method has similar features as those in [101], as it attempts to generate meaningful indexing terms from Thai text documents using Thai language-dependent techniques (i.e. requiring linguistic knowledge of the Thai language) such as POS tagging and stopword removal. Therefore, the hybrid method consists of two main steps: (1) a process of extracting the frequent max substrings as indexing terms, using the frequent max substring technique, and (2) a process of processing those frequent max substrings into indexing terms which are meaningful, using the Thai language-dependent techniques of stopword removal and POS tagging as shown in Figure 4.1.

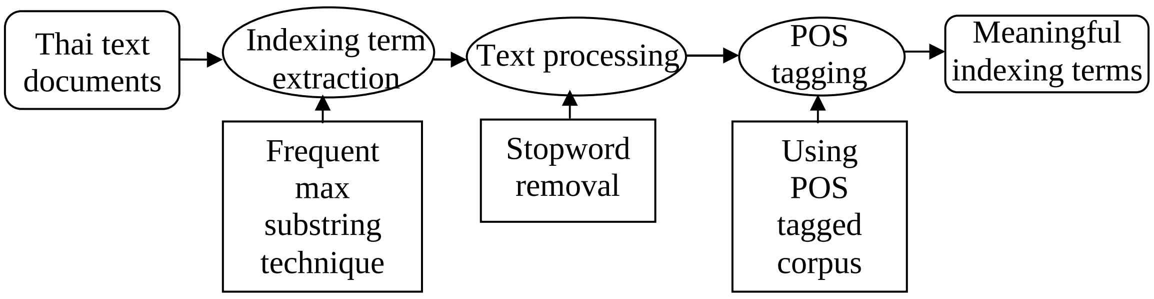


Figure 4.1. A system architecture

From Figure 4.1, Thai text documents are used as an input to extract meaningful indexing terms, and the process of extracting meaningful indexing terms will be described as follows.

Let *D* be a text collection consisting of *n* Thai text documents, d_1 d_2 …. d_n. Firstly, the frequent max substring technique is used to extract frequent max substrings which satisfy the following constraint: the length of the frequent max substrings is more than two at the given frequency threshold value. This process does not require any linguistic knowledge of the Thai language. As mentioned in Chapter 2, most single and bi-gram indexing terms in Thai text documents are insignificant and may not have any useful meaning in the Thai language. These terms are usually used only as parts of words or to form some grammatical constructions in the Thai language.

Assuming the frequent max substring technique produces *m* frequent max substrings, the length of which is more than two, denoted $fms = (fm_1, fm_2, \ldots, fm_m)$, where fm_i is the *i*th frequent max substring extracted from the Thai text documents.

After these frequent max substrings are extracted, the next process is Thai stopword removal. Thai stopwords are frequently occurring insignificant words in the Thai language. These Thai words do not represent the content of the documents as discussed in Section 2.4.1.3 in Chapter 2. Therefore, such words should be removed from the set of frequent max substrings first. In this process, the set of stopwords will be compared to the frequent max substrings extracted from the first step in order to remove these words from the extracted frequent max substrings. The set of Thai stopwords can be denoted by $STWORD = (st_1, st_2, st_j \ldots, st_l)$ where $1 \leq j \leq l$, l is the number of stopwords, and st_j is a member of the Thai stopwords group in the Thai language as shown in Table 2.2 in Chapter 2. Let fm_i be the frequent max substring that is extracted earlier by the frequent max substring technique and this term has a length of more than two. If $fm_i = st_j$, fm_i will be removed from the index. As a result, the index will no longer contain stopwords. Finally, a further process is to perform part-of-speech (POS) tagging, which assigns each frequent max substring the appropriate POS tag in order to extract meaningful indexing terms. The POS tags used in this thesis are derived from the ORCHID project [102]. ORCHID is the name of the collaboration project between Communications Research Laboratory (CRL) of Japan and National Electronics and Computer Technology Center (NECTEC) of Thailand, for building a Thai POS tagged corpus. This is a freely available corpus that contains already segmented Thai texts [55]. This project was started in 1996 with the aim of creating a Thai language corpus for processing the Thai language. The POS tag set consists of 47 tags, where each syntactic category is further divided into subcategories as shown in Table 4.1.

Table 4.1 Thai part-of-speech as tagset for ORCHID[102]

No.	POS	Description	Example
1	NPRP	Proper noun	วินโดวส์ 95, โคโรน่า, โค้ก, พระอาทิตย์
2	NCNM	Cardinal number	หนึ่ง, สอง, สาม, 1, 2, 3
3	NONM	Ordinal number	ที่หนึ่ง, ที่สอง, ที่สาม, ที่1, ที่2, ที่3
4	NLBL	Label noun	1, 2, 3, 4, ก, ข, a, b
5	NCMN	Common noun	หนังสือ, อาหาร, อาคาร, คน
6	NTTL	Title noun	ดร., พลเอก
7	PPRS	Personal pronoun	คุณ, เขา, ฉัน
8	PDMN	Demonstrative pronoun	นี่, นั่น, ที่นั่น, ที่นี่
9	PNTR	Interrogative pronoun	ใคร, อะไร, อย่างไร
10	PREL	Relative pronoun	ที่, ซึ่ง, อัน, ผู้
11	VACT	Active verb	ทำงาน, ร้องเพลง, กิน
12	VSTA	Stative verb	เห็น, รู้, คือ
13	VATT	Attributive verb	อ้วน, ดี, สวย
14	XVBM	Pre-verb auxiliary, before negator ‘ไม่’	เกิด, เกือบ, กำลัง
15	XVAM	Pre-verb auxiliary, after negator ‘ไม่’	ค่อย, น่า, ได้
16	XVMM	Pre-verb, before or after negator ‘ไม่’	ควร, เคย, ต้อง
17	XVBB	Pre-verb auxiliary, in imperative mood	กรุณา, จง, เชิญ, อย่า, ห้าม
18	XVAE	Post-verb auxiliary	ไป, มา, ขึ้น
19	DDAN	Definite determiner, after noun without classifier in between	นี่, นั่น, โน่น, ทั้งหมด
20	DDAC	Definite determiner, allowing classifier in between	นี้, นั้น, โน้น, นู้น
21	DDBQ	Definite determiner, between noun and classifier or preceding quantitative expression	ทั้ง, อีก, เพียง
22	DDAQ	Definite determiner, following quantitative expression	พอดี, ถ้วน
23	DIAC	Indefinite determiner, following noun; allowing classifier in between	ไหน, อื่น, ต่างๆ
24	DIBQ	Indefinite determiner, between noun and classifier or preceding quantitative expression	บาง, ประมาณ, เกือบ
25	DIAQ	Indefinite determiner, following quantitative expression	กว่า, เศษ
26	DCNM	Determiner, cardinal number expression	<u>หนึ่ง</u>คน, เสือ <u>2</u> ตัว
27	DONM	Determiner, ordinal number expression	ที่หนึ่ง, ที่สอง, ที่สุดท้าย

No.	**POS**	**Description**	**Example**
28	ADVN	Adverb with normal form	เก่ง, เร็ว, ช้า, สม่ำเสมอ
29	ADVI	Adverb with iterative form	เร็วๆ, เสมอๆ, ช้าๆ
30	ADVP	Adverb with prefixed form	โดยเร็ว
31	ADVS	Sentential adverb	โดยปกติ, ธรรมดา
32	CNIT	Unit classifier	ตัว, คน, เล่ม
33	CLTV	Collective classifier	คู่, กลุ่ม, ฝูง, เชิง, ทาง, ด้าน, แบบ, รุ่น
34	CMTR	Measurement classifier	กิโลกรัม, แก้ว, ชั่วโมง
35	CFQC	Frequency classifier	ครั้ง, เที่ยว
36	CVBL	Verbal classifier	ม้วน, มัด
37	JCRG	Coordinating conjunction	และ, หรือ, แต่
38	JCMP	Comparative conjunction	กว่า, เหมือนกับ, เท่ากับ
39	JSBR	Subordinating conjunction	เพราะว่า, เนื่องจาก, ที่, แม้ว่า, ถ้า
40	RPRE	Preposition	จาก, ละ, ของ, ใต้, บน
41	INT	Interjection	โอ้ย,โอ้, เออ, เอ๋, อ๋อ
42	FIXN	Nominal prefix	การทำงาน, ความสนุกสนาน
43	FIXV	Adverbial prefix	อย่างเร็ว
44	EAFF	Ending for affirmative sentence	จ๊ะ, จ้ะ, ค่ะ, ครับ, นะ, น่า, เถอะ
45	EITT	Ending for interrogative sentence	หรือ, เหรอ, ไหม, มั้ย
46	NEG	Negator	ไม่, มิได้, ไม่ได้, มิ
47	PUNC	Punctuation	(,), ‘, ,, ;

To tag Thai text documents with appropriate parts-of-speech (POS), the training corpus derived from ORCHID, where texts are manually segmented and tagged with the POS, is used to segment and assign POS to each word. A paragraph in the training corpus is separated into sentences and then into words before assigning POS to each word as shown in Figure 4.2.

```
%TTitle: การประชุมทางวิชาการ ครั้งที่ 1
%ETitle: [1st Annual Conference]
%TAuthor:
%EAuthor:
%TInbook: การประชุมทางวิชาการ ครั้งที่ 1, โครงการวิจัยและพัฒนาอิเล็กทรอนิกส์และคอมพิวเตอร์,
ปีงบประมาณ 2531, เล่ม 1
%EInbook: The 1st Annual Conference, Electronics and Computer Research and Development
Project, Fiscal Year 1988, Book 1
%TPublisher: ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ, กระทรวงวิทยาศาสตร์ เทคโนโลยี
และการพลังงาน
%EPublisher: National Electronics and Computer Technology Center, Ministry of Science,
Technology and Energy
%Page:
%Year: 1989
%File:
#P1
#1
การประชุมทางวิชาการ ครั้งที่ 1//
การ/FIXN
ประชุม/VACT
ทาง/NCMN
วิชาการ/NCMN
<space>/PUNC
ครั้ง/CFQC
ที่ 1/DONM
//
#2
โครงการวิจัยและพัฒนาอิเล็กทรอนิกส์และคอมพิวเตอร์//
โครงการวิจัยและพัฒนา/NCMN
อิเล็กทรอนิกส์/NCMN
และ/JCRG
```

Figure 4.2. Sample of the training corpus which is POS tagged text

In the hybrid method, the training corpus—which is made up of POS tagged texts—is used to segment and assign each frequent max substring with parts-of-speech (POS) by comparing and analyzing the POS of the segmented words in the frequent max substrings. For example, if fm_i consists of seven words, fm_i will then be denoted as fm_i = (w_1, w_2, w_3, w_4, w_5, w_6, w_7). Figure 4.3 shows an example of POS tagging of the frequent max substrings.

Let fm_i = ส่งผลต่อการพัฒนาเศรษฐกิจโดยตรง

English translation It directly impacts on economic system development

$$\downarrow$$

fm_i =	w_1	w_2	w_3	w_4	w_5	w_6	w_7
Segmented words	ส่งผล	ต่อ	การ	พัฒนา	เศรษฐกิจ	โดย	ตรง
POS	VACT	RPRE	FIXN	VACT	NCMN	FIXV	VATT

Figure 4.3. Example of POS tagging of frequent max substrings

To extract meaningful frequent max substrings (i.e. meaningful indexing terms), the POS of segmented words on the frequent max substrings will be considered. Although a frequent max substring may consist of several words and may also be composed of many different POS depending on the words, meaningful frequent max substrings usually contain noun words. This is because keywords are usually carried by nouns that can be used to identify subject matter [101]. The analysis based on the frequency of words occurring in Thai text documents showed that most Thai words are nouns [10]. As a result, frequent max substrings which contain noun words can be considered as meaningful indexing terms rather than frequent max substrings which carry many prepositions or conjunctions etcetera. This shows that by applying the frequent max substring technique to the Thai language-dependent technique can assist Thai text indexing, as this method generates frequent meaningful indexing terms from Thai text documents.

4.4 Experimental studies

In this section, an experiment for extracting meaningful indexing terms based on the hybrid method is presented. Fifty Thai web pages mentioned in Chapter 3 were used in the experiment. In this experiment, the frequent max substrings with a length of more than two are first extracted from the input dataset. The Thai language-dependent technique is then applied to the resultant set of frequent max substrings in order to reduce the number of insignificant indexing terms using stopword removal and to find meaningful indexing terms using POS tagging as described in Section 4.3.

Table 4.2 shows the number of frequent max substrings with a length of more than two that are extracted from the input dataset. The table also shows the number of stopwords removed from the extracted frequent max substrings and the number of meaningful indexing terms extracted from frequent max substrings by using POS tagging.

Table 4.2 Number of insignificant indexing terms and meaningful indexing terms extracted with hybrid method

Text Id	Text name	Text length (*n* characters)	No. of frequent max substrings with a length of more than 2	No. of stopwords	No. of meaningful terms
1	sport2	721	48	6	41
2	sport1	805	68	5	60
3	sport4	874	49	7	38
4	sport3	913	68	14	51
5	travel4	1007	92	16	74
6	sport7	1020	98	7	88
7	sport9	1057	78	14	63
8	travel3	1107	96	16	80
9	sport15	1135	91	12	73

Text Id	**Text name**	**Text length (*n* characters)**	**No. of frequent max substrings with a length of more than 2**	**No. of stopwords**	**No. of meaningful terms**
10	travel11	1157	93	11	79
11	sport6	1417	134	16	112
12	travel13	1444	141	18	117
13	sport5	1468	125	18	102
14	education2	1507	141	22	119
15	political1	1512	153	22	124
16	education4	1544	126	17	103
17	travel2	1561	129	14	114
18	travel1	1563	142	14	123
19	sport8	1643	144	12	129
20	political3	1652	177	26	147
21	travel15	1789	165	15	150
22	political14	1879	193	27	164
23	travel12	1901	159	15	138
24	travel6	2002	200	31	162
25	political13	2030	193	26	165
26	education5	2035	162	14	143
27	political9	2064	218	22	192
28	political15	2107	224	27	191
29	political5	2126	239	36	201
30	education3	2142	223	29	190
31	education1	2217	253	24	224
32	travel14	2290	255	24	230
33	sport13	2339	236	22	208
34	sport10	2367	202	13	185
35	sport11	2368	202	13	185
36	travel10	2431	253	29	221
37	political2	2437	256	37	213
38	political8	2518	249	30	217
39	political4	2735	265	27	233
40	political12	2760	293	28	261
41	travel8	2835	246	24	218
42	travel9	2893	333	23	304
43	political10	2922	369	41	322
44	sport12	3020	331	23	300
45	political11	3243	360	38	319
46	travel7	3475	381	28	347
47	sport14	3580	351	19	329
48	travel5	3643	434	23	401
49	political6	4010	517	49	457
50	political7	4022	470	34	430

It is clear that the number of meaningful indexing terms extracted by the hybrid method as shown in Table 4.2 is less than the number of indexing terms extracted by the frequent max substring technique as shown in Appendix C. This means that the hybrid method reduces the cost of indexing in terms of storage space. Furthermore, most of the meaningful indexing terms extracted by the hybrid method are noun phrases. These indexing terms often contain many keywords that represent the content of Thai text documents. In order to determine whether the hybrid method is appropriate for indexing Thai text documents, a comparison with language-dependent techniques is needed. The word inverted index technique is used in the study to compare the results from the hybrid method in terms of the number of indexing terms and retrieval performance.

Table 4.3 compares the number of indexing terms extracted with the hybrid method with the number of indexing terms generated from the word inverted index technique, with both techniques being used as the language-dependent technique for indexing Thai text documents.

Table 4.3 Comparison of number of indexing terms extracted from hybrid method and word inverted index technique

Text Id	Text name	Text length (*n* characters)	Number of indexing terms	
			Hybrid method	Word inverted index technique
1	sport2	721	41	29
2	sport1	805	60	36
3	sport4	874	38	27
4	sport3	913	51	42
5	travel4	1007	74	30
6	sport7	1020	88	32
7	sport9	1057	63	36
8	travel3	1107	80	43
9	sport15	1135	73	57
10	travel11	1157	79	33
11	sport6	1417	112	67
12	travel13	1444	117	47

Text Id	Text name	Text length (*n* characters)	Number of indexing terms	
			Hybrid method	**Word inverted index technique**
13	sport5	1468	102	77
14	education2	1507	119	48
15	political1	1512	124	54
16	education4	1544	103	56
17	travel2	1561	114	53
18	travel1	1563	123	53
19	sport8	1643	129	73
20	political3	1652	147	67
21	travel15	1789	150	71
22	political14	1879	164	73
23	travel12	1901	138	78
24	travel6	2002	162	104
25	political13	2030	165	80
26	education5	2035	143	74
27	political9	2064	192	74
28	political15	2107	191	68
29	political5	2126	201	79
30	education3	2142	190	74
31	education1	2217	224	84
32	travel14	2290	230	80
33	sport13	2339	208	96
34	sport10	2367	185	86
35	sport11	2368	185	86
36	travel10	2431	221	89
37	political2	2437	213	93
38	political8	2518	217	84
39	political4	2735	233	98
40	political12	2760	261	105
41	travel8	2835	218	88
42	travel9	2893	304	97
43	political10	2922	322	120
44	sport12	3020	300	144
45	political11	3243	319	127
46	travel7	3475	347	129
47	sport14	3580	329	109
48	travel5	3643	401	131
49	political6	4010	457	132
50	political7	4022	430	144

To illustrate the number of indexing terms, Figure 4.4 shows the number of indexing terms extracted by the two language-dependent techniques: the hybrid method and the word inverted index technique.

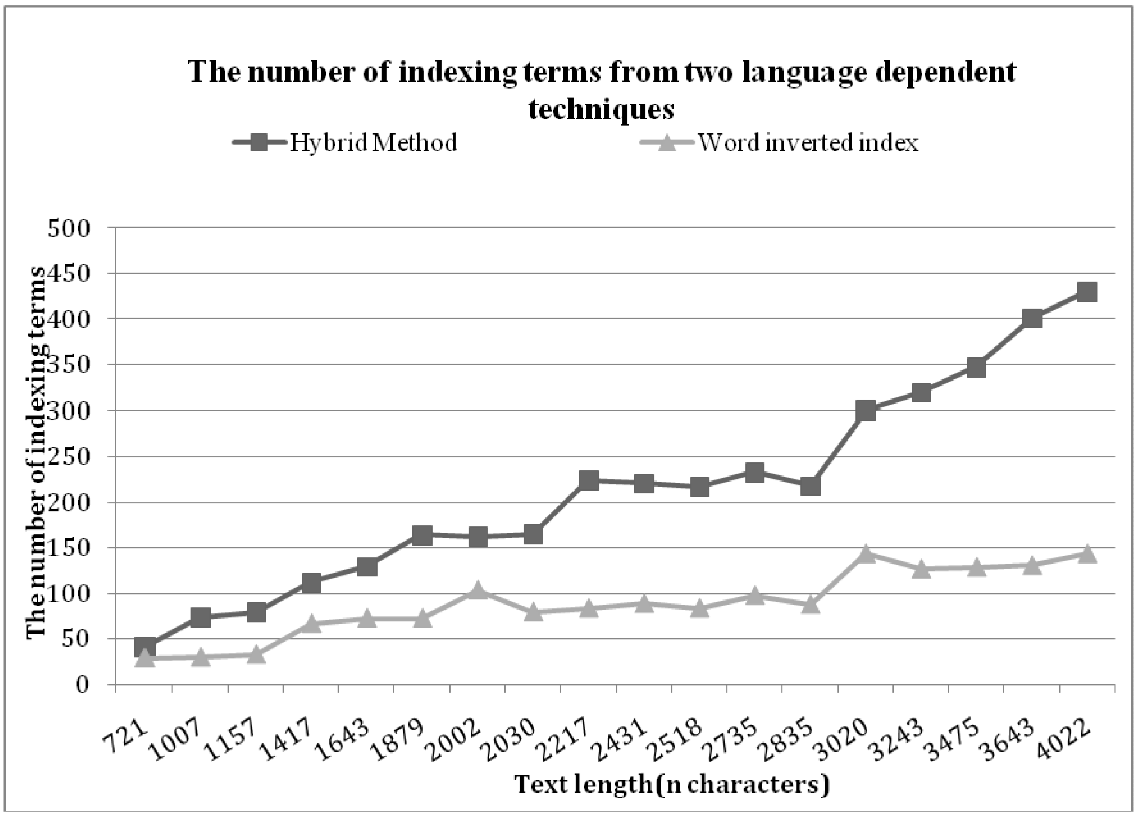


Figure 4.4. Number of indexing terms extracted from two language-dependent techniques

From Figure 4.4, it can be observed that the hybrid method extracted more indexing terms than the word inverted index. When the text length of the document is short, the difference between the hybrid method and the word inverted method is not very big. However, as the text length grows, the number of indexing terms extracted by the hybrid method grows as well. This is because longer text documents usually have a higher possibility of containing frequent max substrings than shorter texts documents do. As has been discussed in Section 4.3, shorter terms are normally insignificant terms in most Thai text documents. Thus the hybrid method works on extracting more meaningful and representative longer terms, and this can be archived in longer text length documents.

Although the hybrid method still extracts more indexing terms than the word inverted index technique does, the hybrid method provides better results in terms of retrieval performance. This will be explained in the following section.

In comparing retrieval performance between two language-dependent techniques, eight test queries, four phrase queries and four single word queries as shown in Table 3.2 in Chapter 3, are used to test the retrieval ability of the hybrid method and the word inverted index technique. Two indices are created from 50 Thai web pages, one using the hybrid method and the other using the word inverted index method. The retrieval performance is measured by precision and recall of the two methods as shown in Table 4.4 and 4.5 respectively. Table 4.6 also shows average precision and recall values provided by the two indexing techniques. Note that one of the drawbacks of the word inverted index is that the phrase queries need to be segmented into words before searching (i.e. word segmentation). Meanwhile, the hybrid method does not require query processing before searching. The hybrid method can directly use phrase queries for searching.

Table 4.4. Precision and recall values of word inverted index technique

Phrase queries			Single word queries		
Queries	**Precision**	**Recall**	**Queries**	**Precision**	**Recall**
การเมืองไทย (Thai politics)	0.66	0.66	คณะรัฐมนตรี (Cabinet)	0.33	1
เที่ยวไทย (Thai travel)	0.78	1	โรงแรม (Hotel)	0.54	1
ผลการแข่งขัน (Competition result)	0.5	0.5	นักกีฬา (Athlete)	0.48	1
การเรียนการสอน (Learning and teaching)	1	1	เรียน (Learn)	0.62	1
Average	0.735	0.79		0.4925	1

Table 4.5. Precision and recall values of the hybrid method

Phrase queries			Single word queries		
Queries	**Precision**	**Recall**	**Queries**	**Precision**	**Recall**
การเมืองไทย (Thai politics)	0.77	0.77	คณะรัฐมนตรี (Cabinet)	0.33	1
เที่ยวไทย (Thai travel)	0.78	1	โรงแรม (Hotel)	0.54	1
ผลการแข่งขัน (Competition result)	0.75	0.75	นักกีฬา (Athlete)	0.48	1
การเรียนการสอน (Learning and teaching)	1	1	เรียน (Learn)	0.62	1
Average	0.825	0.88		0.4925	1

Table 4.6. Average precision and recall values of two indexing techniques

Language-dependent methods	Avg. Precision	Avg. Recall
Word inverted index technique	0.6138	0.895
Hybrid method	0.6588	0.94

From Table 4.5 and 4.6, it can be observed that the hybrid method provides the same precision and recall values as the frequent max substring technique described in Section 3.5.3. The results show that the hybrid method performs better than the word inverted index technique in terms of retrieval performance, even many small and insignificant indexing terms were not retained in the hybrid method. In this experiment, it can be observed that small and insignificant indexing terms may not impact on retrieval performance as these indexing terms are usually not used as a query in searching relevant documents. These terms are also not used as keys to specify subject matter in Thai text documents. However, the retrieval performance of the word inverted index technique and the hybrid method could possibly fall if the given query is made up of stopwords or the given query consists of three characters. Despite this, these cases usually do not happen in practicality, because noun phrases or noun words are usually used as queries in searching relevant documents in most cases.

4.5 Conclusion

This chapter proposes the hybrid method that integrates the frequent max substring technique and Thai language-dependent technique to extract meaningful indexing terms from Thai text documents. In the hybrid method, frequent max substrings with a length of more than two are first extracted. The Thai language-dependent technique is then applied to remove insignificant indexing terms from the set of frequent max substrings, using stopword removal. The resultant frequent max substrings are finally considered using their part-of-speech with the POS tagged corpus in order to find meaningful indexing terms. The experimental comparison results on extracting indexing terms from 50 Thai text documents were presented in this chapter. The hybrid method was compared with the word inverted index technique in terms of the number of indexing terms and retrieval performance. These two techniques are used as language-dependent techniques for indexing Thai text documents. It can be observed that the hybrid method results in a higher number of indexing terms than the word inverted index technique does. However, the hybrid method provides better retrieval performance than the word inverted index technique. One reason for this is that with the hybrid method, the meaningful indexing terms extracted are mostly noun phrases and these indexing terms better represent the content of the documents. Consequently, the hybrid method could be used as an option for indexing Thai text documents. This experiment also shows that the frequent max substring technique can be used with other language-dependent techniques when necessary, to become an effective hybrid language-dependent technique.

Chapter 5

Non-Segmented Document Clustering Using Self-Organizing Map and the Frequent Max Substring Technique

5.1 Introduction

This chapter proposes a non-segmented document clustering method using the self-organizing map (SOM) and the frequent max substring technique to improve the efficiency of information retrieval. The objective of this chapter is to demonstrate that the frequent max substring technique presented in Chapter 3 can be used with other techniques to enhance the clustering of non-segmented documents. SOM is selected for the illustration mainly because it has been widely used for document clustering and is successful in text indexing [103], [104]. However, when applying it to a non-segmented document, the challenge is to identify any interesting patterns efficiently. There are two main phases in the proposed method: the pre-processing phase and clustering phase. In the preprocessing phase, the frequent max substring technique is first applied to discover frequent max substrings from non-segmented text documents. These frequent max substrings are then used as indexing terms, together with their number of occurrences, to form a document vector. In the clustering phase, SOM is used to generate the document cluster map by using the feature vector of the frequent max substrings. To demonstrate the proposed technique, experimental studies and comparison results on clustering Thai text documents, which consist of non-segmented texts, are presented in this chapter. The results show that the proposed technique can be used for Thai texts. The document cluster map generated from the proposed method can be used to find the relevant documents more efficiently.

5.2 Document clustering

Document clustering is an important area [45] in today's world due to the rapid increase in the number of electronic documents. Document clustering, which can sometimes be generalized as text clustering, indentifies the similarity of documents and summarizes a large number of documents using key attributes of the clusters. Document clustering uses analysis of the distance between documents in order to find the similarities of documents and may assist fast information retrieval or filtering [105]. This is because the clustering technique categorizes documents into groups based on their similarities in terms of their member occurrences. Thus clustering can be used to categorize document databases and digital libraries, as well as providing useful summary information of the categories for browsing purposes. In information retrieval, a typical search on a document database or the world wide web can return several thousands of documents in response to users' queries. It is often very difficult for users to identify their documents of interest from such a huge number of documents. Clustering documents enables the user to have a clear and easy grasp of the relevant documents from the collection of documents that are similar to each other and could be relevant to the user's queries. Hence clustering normally enables the user to locate the right document with increased efficiency.

For text clustering in information retrieval, a document is normally considered as a bag of words, even though a document actually consists of a sequence of sentences and each sentence is composed of a sequence of words. Very often the positions of words are ignored when performing document clustering. Words, also known as indexing terms, and their weights in documents are usually used as important parameters to compute the similarity of documents [106]. Those documents that contain similar indexing terms and frequencies are grouped under the same cluster. This process is straightforward for European languages where words are clearly defined by word delimiters such as space or other special symbols. European texts are explicitly segmented into word tokens that are used as indexing terms. Many algorithms have been developed to calculate the similarity of documents and to build clusters for fast information retrieval. In contrast, document clustering can be a challenging task for many Asian languages such as Chinese, Japanese, Korean and Thai, because these languages are non-segmented languages, (i.e. a sentence is written continuously as a sequence of characters without explicit word boundary delimiters). Due to this characteristic, texts in a non-segmented document cannot be directly used to calculate the similarity. Normally, pre-processing could be necessary to discover keywords (i.e. indexing terms) for Asian documents before clustering [107], [108]. As a result, most approaches for clustering non-segmented documents consist of two phases: a keyword extraction process to extract the keywords, and a document clustering process to compute the similarity between the input documents.

5.3 Keyword extraction

Keywords are usually regarded as an important key to identifying the main content of documents [98], [109], [108]. Most of the semantics are usually carried by nouns, although a sentence in a natural language text is composed of nouns, pronouns, articles, verbs, adjectives, adverbs, and connectives. Keyword extraction is one of the main processes in text mining. Most keyword extraction methods proposed in literature were accomplished by constructing a set of words from given texts. Keywords are then selected from the set of words during the pre-processing step. Many approaches have been proposed to extract keywords from non-segmented documents such as Chinese [63], Japanese [110] or Thai documents [111]. Most techniques are based on word segmentation, which is one of the most widely used information extraction techniques in natural language processing (NLP). However, most word segmentation approaches involve complex language analysis and require long computational time as described in Chapter 2. After keyword extraction is performed, keywords are then transformed into a feature vector of the words that appear in documents. The term-weights (usually term-frequencies) of the words are also contained in each feature vector. The vector space model (VSM) has been a standard model of representing documents by containing the set of words with their frequencies [45]. In the VSM, each document is replaced by the vector of the words. The vector size is dependent on the number of keywords that appear in the documents. Let w_{ik} be the weight of keyword k that appears in the document i, and $D_i = (w_{i1}, w_{i2},\dots, w_{it})$ is the feature vector for document i, where t is the number of unique words of all documents. Therefore, the size of the feature vector is equal to t dimension. In Figure 5.1, an example of the document vectors where t is equal to 3 is depicted.

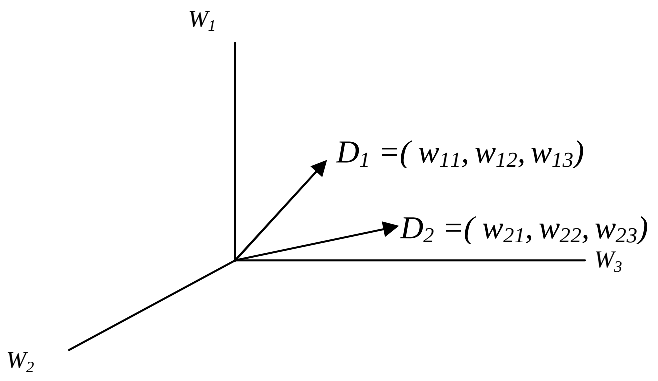

W_1
$D_1 =(w_{11}, w_{12}, w_{13})$
$D_2 =(w_{21}, w_{22}, w_{23})$
W_3
W_2

Figure 5.1. Example of document vectors in 3-dimension

From Figure 5.1, the similarity between two documents can be computed with one of several similarity measures based on two corresponding feature vectors, for example cosine measure, Jaccard measure, and Euclidean distance measure [112].

5.4 Document clustering algorithms and related works

In document clustering, there are two main approaches: the hierarchical and partitional approaches [113], [114], [115]. The hierarchical approach produces document clusters by using a nested sequence of partitions that can be represented in the form of a tree structure called a dendrogram. The root of the tree contains one cluster covering all data points, and singleton clusters of individual data points are shown on the leaves of the tree. There are two basic approaches when performing hierarchical clustering: agglomerative (bottom up) and divisive (top down) clustering [115]. The advantage of the hierarchical approach is that it can take any form of similarity function, and also the hierarchy of clusters allows users to discover clusters at any level of detail. However, this technique may suffer from the chain effect, and its space requirement is at least quadratic or $O(n^2)$ compared to the *k*-means algorithm that provides $O(Iknm)$ where I is the number of necessary iterations, k is the number of clusters, n is the number of documents and m is the dimensionality of the vectors. The partitional approach [116], on the other hand, can be divided into several techniques—for example, *k*-means [117], Fuzzy c-means [118], and QT (quality threshold) [119] algorithms. The *k*-means algorithm is more widely used among all clustering algorithms because of its efficiency and simplicity. The basic idea of the *k*-means algorithm is that it separates given data into *k* clusters where each cluster has the center point, also called centroid, which can be used to represent the cluster. *K*-data points are randomly selected as the centroids by the algorithm. All data points are then assigned to the closest centroid by computing the distance between every data point and each centroid. Therefore, each centroid and its members can form a cluster. The algorithm also re-computes the centroid of each cluster using the data in the current cluster, and this step is repeated until the centroids stabilize. The main advantages of the *k*-means algorithm are its efficiency and simplicity. Its weaknesses are that it is only applicable to data sets where the notion of the mean is defined, the number of clusters can be identified by users, and it is sensitive to data points that are very far away from other points called outliers [45].

Furthermore, the self-organizing map (SOM) [120], [121] can be used as one of the clustering algorithms, as this technique has been widely used and is successful for document clustering due to its popularity and performance in unsupervised density mapping of an input distribution [103], [122]. The SOM is an artificial neural network architecture that uses unsupervised learning. SOM is capable of ordering high dimensional data into a two-dimensional map by grouping similar (or closely related) inputs together. To use SOM in document clustering, text documents are described by features with high dimensionality, and SOM based techniques have been successfully applied to document clustering.

Many clustering techniques have been developed and can be applied to cluster segmented languages. Most of these traditional approaches use documents as the basis for clustering [123], [124]. The vector space document (VSD) model is a widely used data representation model for document clustering [125]. This data model starts with a representation of any document as a feature vector of the words that appear in documents. The term-weights of the words are also contained in each feature vector. The similarity measures are used to compute the similarity of two document vectors. An alternative approach of document clustering is phrase-based document clustering. Etzioni [126] introduced the notion of phrase-based document clustering. They proposed to use a generalized suffix-tree to obtain information about the phrases between two documents and use common phrases to cluster the documents. According to [104], Bakus, Hussin, and Kamel used a hierarchical phrase grammar extraction procedure to identify phrases from documents and used these phrases as features for document clustering. The self-organizing map (SOM) method was used as the clustering algorithm. An improvement in the clustering performance was demonstrated in their paper when using phrases rather than single words as features [23].

Mladenic and Grobelnik used a Naive Bayesian method to classify documents based on word sequences of different length [127]. Experimental results show that using the word sequences with a length of no more than three words can improve the performance of a text classification system. But when the average length of used word sequences is longer than three words, there is no difference between using word sequences or single words.

However, there is not much research on phrase-based document clustering for Asian languages, primarily due to the fact that most Asian language texts are non-segmented and it is difficult to separate words and phrases from the non-segmented texts. Most document clustering approaches require a pre-processing stage where word segmentation, stopword removal or semantic analysis is performed. Natural language processing techniques provide good support for this step. Word segmentation is an important step involved in most natural language processing tasks. A text is separated into a sequence of tokens by using word segmentation techniques.

In [107], a Chinese document clustering method using a data mining technique and neural network model was proposed. This technique was divided into two main parts:

first, the pre-processing part, which provides Chinese sentence segmentation, and the second part, which is clustering, the dynamical SOM model was adopted for dynamically clustering segmented text documents. In addition, this method uses the term vectors clustering process instead of the document vectors clustering process.

In the Thai language, Kruengkrai and Jaruskulchai propose a model for document clustering in Thai text documents [128]. They investigated complete links for hierarchical clustering and single passes for non-hierarchical clustering techniques. These two algorithms are applied to cluster Thai news archives. The research also employs a parallel algorithm in calculating the similarity between two documents. They implemented an algorithm on the PIRUN Linux cluster, which is a parallel computer using cluster computing technology. Experimental results show that there is no difference in the clustering results given by the two algorithms.

5.5 SOM based clustering using the frequent max substring technique for non-segmented texts

In this section, a new method that combines Kohonen's SOM and the frequent max substring technique, to process the non-segmented text documents into clusters is described. SOM is one of the main unsupervised learning methods in the family of artificial neural networks (ANN) that was first developed by Teuvo Kohonen in 1984 [129]. The SOM can be visualized as a regular two-dimensional array of cells or nodes (neurons). The SOM algorithm defines a mapping from the input vector onto a two-dimensional array of nodes. When the input vector $x(t) \in R^n$ is given, it is connected to all neurons in the SOM array denoted as vector $m_i(t) \in R^n$, which are associated by each neuron and is gradually modified in the learning process. The input vector $x(t) \in R^n$ is the input data set where t is the indexing term of the input documents. This input data set has to be mapped with all neurons in the map denoted as a two-dimensional network of cells or the model vector $m_i(t) \in R^n$.

In mapping, the node where vector m_i is the most similar to the input vector x will be activated. This node is often called a best-matching node or a winner. The winner and a number of its neighboring nodes in the SOM array are then turned towards the input vector x according to the learning principle.

In this chapter, a set of non-segmented documents (Thai documents) is used as an input to train a map using SOM. Those Thai documents used in Chapter 3 will be converted to a text collection to be used in this chapter. The process of clustering will be described as follows.

Let D be a document collection consisting of n documents, d_1, d_2, ..., d_n. Firstly, the frequent max substring technique is used to generate a set of frequent max substrings, *FMAX*, at the given frequency threshold value θ from the document collection. These extracted frequent max substrings will be used as the set of indexing terms for the document collection.

Assuming the above process produces m frequent max substrings from the document collection, denoted as $FMAX = (fm_1, fm_2, ..., fm_m)$, where fm_i is the ith frequent max substring generated from the document collection. These m substrings are used as the indexing terms.

The weight w_{ij} represents the frequency of indexing term fm_i occurring in document d_j for each indexing term and each document. An $m*n$ matrix of such weights is then calculated. In this matrix, row i represents the frequencies of occurrence of the ith indexing term fm_i in the n documents, while jth column represents the document vector for document j. An example of the matrix is depicted in Figure 5.2.

$$V = \begin{array}{cccccccccl}
 & & & & & & & & & fm_i \\
4 & 3 & 7 & 0 & 0 & 0 & \ldots & 0 & 0 & \text{ผลการแข่งขัน (Competition result)} \\
3 & 3 & 0 & 0 & 2 & 0 & & 2 & 0 & \text{เป็นอันดับที่ (Competition rank)} \\
6 & 4 & 5 & 0 & 0 & 0 & & 0 & 0 & \text{ตารางการแข่งขัน (Competition time table)} \\
2 & 2 & 2 & 0 & 0 & 0 & & 0 & 0 & \text{กรรมการตัดสิน (Umpire)} \\
3 & 3 & 4 & 0 & 0 & 0 & & 0 & 0 & \text{รอบรอบชนะเลิศ (Semi final round)} \\
0 & 0 & 0 & 8 & 4 & 7 & & 4 & 9 & \text{อัตราการแลกเปลี่ยนเงินตรา (Currency exchange rate)} \\
: & & & & & & & & & \quad : \\
0 & 0 & 0 & 3 & 0 & 4 & \ldots & 0 & 0 & \text{ธุรกิจการลงทุน (Investment business)} \\
0 & 0 & 0 & 2 & 0 & 2 & & 0 & 0 & \text{อสังหาริมทรัพย์ (Real estate)} \\
0 & 0 & 0 & 4 & 3 & 4 & & 3 & 3 & \text{สถานะทางการเงิน (Financial status)} \\
0 & 0 & 2 & 3 & 4 & 3 & & 4 & 0 & \text{งบประมาณประจำปี (Yearly budget)} \\
d_1 & d_2 & d_3 & d_4 & d_5 & d_i & \ldots & d_{n-1} & d_n &
\end{array}$$

Figure 5.2. Example of document matrix at given frequency threshold value θ is equal to 2

Figure 5.2 shows an example of document matrix. In a document matrix, each element w_{ij} is at least at θ if fm_i occurs in the document d_j or 0 if fm_i does not appear in the document d_j, i.e.

$$w_{ij} = \begin{cases} \geq \theta & \text{if } fm_i \text{ occurs in } d_j \\ 0 & \text{otherwise.} \end{cases}$$

After the document matrix is obtained, the document vectors are presented to SOM for clustering. These documents can be clustered according to the similarity of their document vectors. Two documents containing the same or similar document vectors will map to the same neuron. In contrast, the two documents may map to two distant neurons if they contain different or non-overlapping frequent max substrings. Furthermore, the documents with similar frequent max substrings may map to neighbouring neurons. This means that neurons can form document clusters by examining mapped neurons in the document cluster map. In Figure 5.3, the organization of the document map, which clusters similar documents into the same neuron, is depicted as shown in the boxes. Frequent max substrings in the boxes represent the content of documents in the collection.

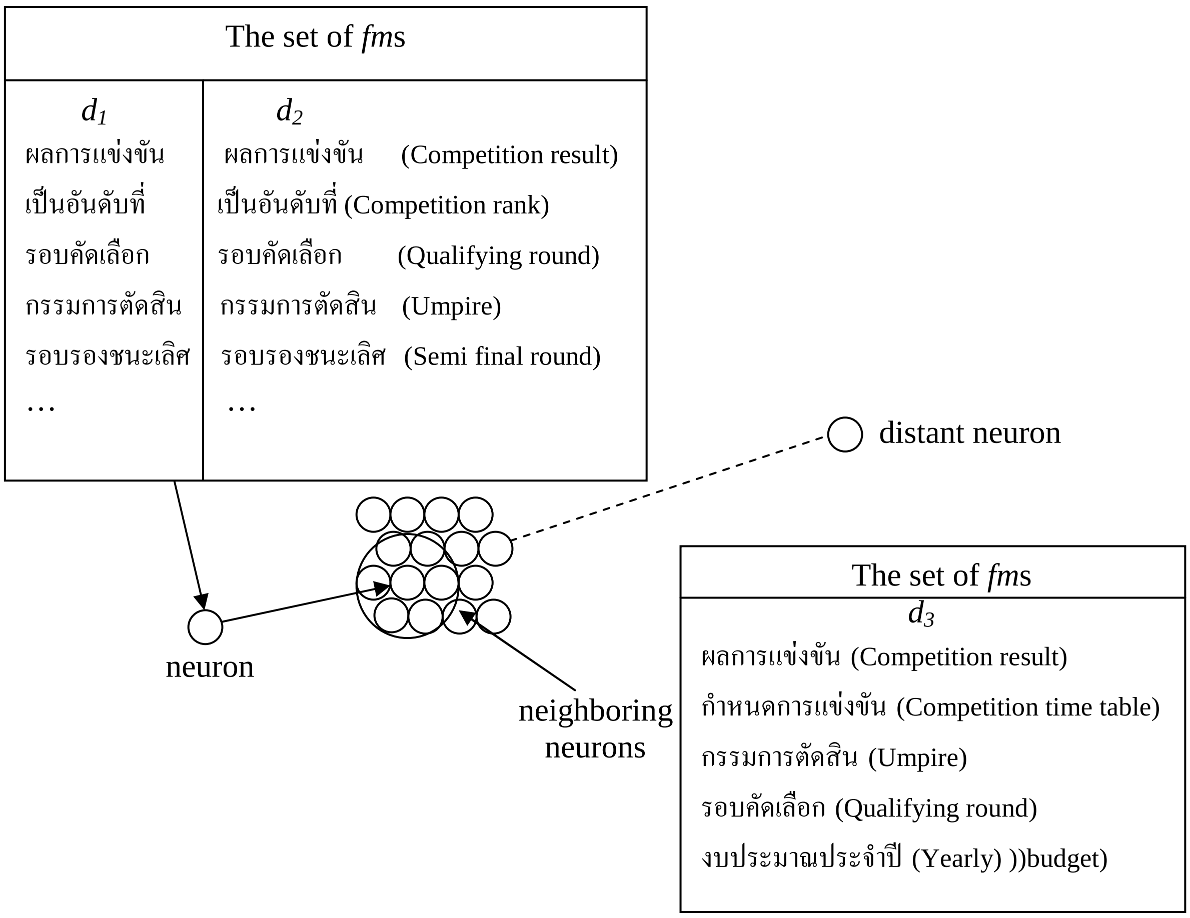

The set of *fm*s
d_1
ผลการแข่งขัน
เป็นอันดับที่
รอบคัดเลือก
กรรมการตัดสิน
รอบรองชนะเลิศ
…
d_2
ผลการแข่งขัน (Competition result)
เป็นอันดับที่ (Competition rank)
รอบคัดเลือก (Qualifying round)
กรรมการตัดสิน (Umpire)
รอบรองชนะเลิศ (Semi final round)
…
distant neuron
neuron
neighboring neurons
The set of *fm*s
d_3
ผลการแข่งขัน (Competition result)
กำหนดการแข่งขัน (Competition time table)
กรรมการตัดสิน (Umpire)
รอบคัดเลือก (Qualifying round)
งบประมาณประจำปี (Yearly)))budget)

Figure 5.3. Document cluster map

After the SOM has been trained, the document clusters are formed by labeling each neuron that contains certain documents of similar type. The documents in the same neuron may not contain exactly the same set of frequent max substrings, but they usually contain mostly overlapping frequent max substrings. As a result, the document cluster map can be used as a prediction model to generate the different groups of similar documents, and each group will then be used to specify the document type by comparing the frequent max substrings of each group with the keywords of each area. In Figure 5.4, clustering the documents into different groups, by mapping input data with neurons in the document cluster map to find document groups of several types is depicted.

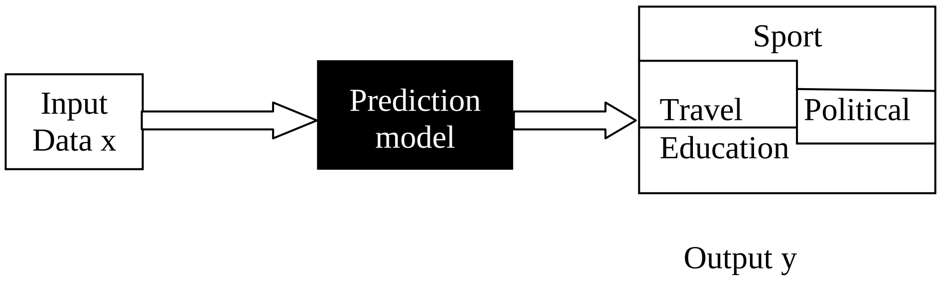


Figure 5.4. Neuron network architecture

From Figure 5.4, the following will describe the process of matching input data *x* with neurons in the document cluster map by using SOM.

Let's consider the input vector $x = [x_1, x_2, \ldots, x_n]^t \in R^n$ as the input data set where *t* is the *FMAX* of the input documents. This input data set has to be matched with all neurons in the map, which is denoted as a two-dimensional network of cells or the model vector $m_i = [m_{i1}, m_{i2}, \ldots, m_{in}]^t \in R^n$ depicted in Figure 5.5. Each neuron *i* in the network contains the model vector m_i, which has the same number of indexing terms as the input vector x.

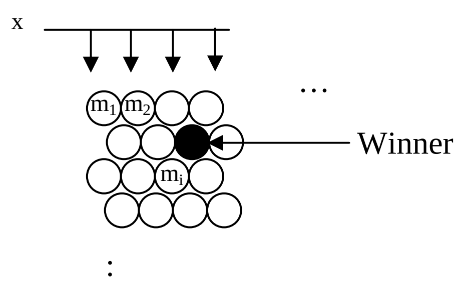


Figure 5.5. Self-organizing map

From Figure 5.5, the input vector x is compared with all neurons in the map or the model vector m_i to find the best matching node, called the winner. The winner unit is the neuron on the map where the set of frequent max substrings of the input vector x is the same or similar to the set of frequent max substrings of the model vector m_i, by using some matching criterion, for example the Euclidean distances between x and m_i. As a result, this method can be used to cluster documents into different groups, and it is also suggested that this can be used to reduce the search time for relevant documents.

5.6 Experimental studies and comparison results

In this section, the experiment for clustering non-segmented documents (Thai documents) based on the proposed SOM and frequent max substring technique is presented. The proposed technique is also compared with the hierarchical clustering technique using single words in a group of documents [130], [128], [131], [132]. Fifty Thai documents used as the text collection in Chapter 3 were used as an input dataset to train a map. All documents were found on Thai news websites, and consist of 15 sport, 15 travel, 15 political, and 5 education documents as shown in Appendix A. In the proposed technique, the set of frequent max substrings was first generated by the frequent max substring technique from the document dataset, at the given frequency threshold value θ. Thirty-five frequent max substrings were extracted, consisting of long and frequently occurring terms in sport, travel, political and educational documents. These were used as the set of indexing terms for these 50 documents. The 50 input documents were then transformed to a document matrix of weighted frequent max substring occurrences. Hence, these 35 indexing terms and 50 input documents form a 35 * 50 matrix, where each document vector was represented by a column of the matrix and the rows of the matrix correspond to the indexing terms. This 35 * 50 matrix is used to train a map using SOM, and the number of neurons selected in this experiment is nine as shown in Figure 5.6. In this experimental study, nine neurons were selected after trials with different number of neurons, as it provided the best result.

Figure 5.6 shows the map containing nine neurons and 50 Thai documents. Each neuron contains a group of similar documents.

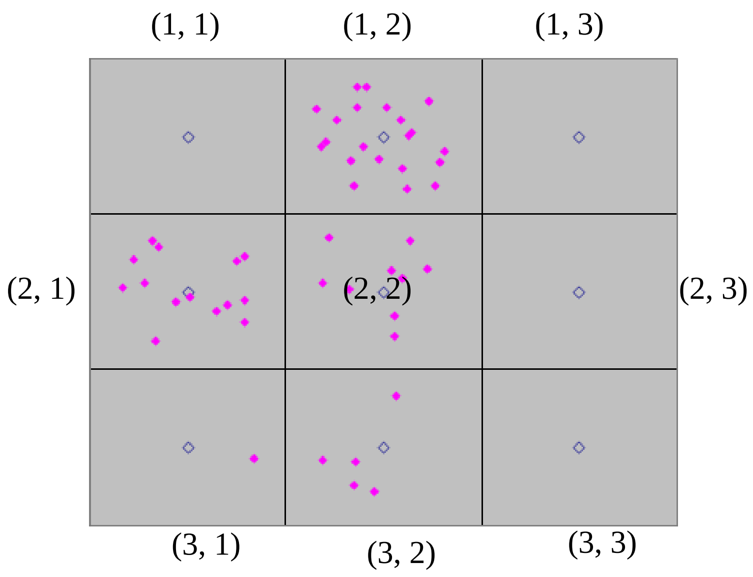


Figure 5.6. SOM contains nine neurons and a group of similar documents from collection of 50 Thai documents

After a 35 * 50 matrix is used to train a map using SOM, the SOM generates a two-dimensional document cluster map as shown in Figure 5.6. This map contains nine neurons (with each neuron corresponding to each cluster for this case), but only five neurons containing groups of similar documents. From Figure 5.6, the experimental result showed that SOM can cluster 50 documents into five neurons on the map, and similar documents were grouped into the same neuron as shown in Table 5.1.

Table 5.1. Clustering results of using SOM and frequent max substring technique

Neuron ID	Row	Column	Document ID
Neuron 5	1	2	Political1, Education1, Education2, Education3, Education4, Sport1, Sport2, Sport3, Sport4, Sport5, Sport6, Sport7, Sport8, Sport9, Sport10, Sport11, Sport12, Sport13, Sport14, Sport15, Travel10
Neuron 2	2	1	Political2, Political3, Political4, Political5, Political6, Political7, Political8, Political9, Political10, Political11, Political12, Political13, Political14, Political15,
Neuron 4	2	2	Travel2, Travel4, Travel5, Travel6, Travel7, Travel8, Travel9, Travel13, Travel15,
Neuron 1	3	1	Travel12
Neuron 3	3	2	Education5, Travel1, Travel3, Travel11, Travel14

One of the basic requirements to measure the performance of the proposed clustering approach is to compare the results with different clustering approaches in terms of occurrences of the group of documents. In this chapter, the proposed technique is compared with hierarchical based document clustering using single words. This method is chosen because it has been widely used and has been applied successfully in many applications in the area of document clustering [132], [133], [131]. This method has also been used to perform Thai document clustering [130], [128], [134].

To use this technique with the Thai language, single words are first extracted by using Thai word segmentation techniques from the same document dataset used in the experiment discussed earlier. After word segmentation is performed, single words are then transformed into feature vectors of the words that appear in the documents. The term-frequencies of the words are also contained in each feature vector. The feature vectors of the words are then used to compute the similarity of the documents by using the hierarchical clustering approach. In hierarchical clustering, the feature vectors of the words with their frequencies were used as input data, and the number of clusters was set to nine, as nine clusters provided the best result as well. The experimental result showed that the hierarchical clustering program can cluster 50 documents into nine clusters as shown in Table 5.2.

Table 5.2. Clustering results of using hierarchical clustering approach

Cluster ID	Document ID
Cluster 1	Education1, Education2, Education3, Education4, Sport9, Sport13, Travel1, Travel2, Travel3, Travel4, Travel5, Travel7, Travel8, Travel9, Travel10, Travel11, Travel12, Travel13, Travel14, Travel15, Political15
Cluster 2	Education5
Cluster 3	Sport1
Cluster 4	Sport2, Sport3, Sport4, Sport5, Sport6, Sport7, Sport8, Sport15
Cluster 5	Sport10, Sport11
Cluster 6	Sport12
Cluster 7	Sport14
Cluster 8	Travel6
Cluster 9	Political1, Political2, Political3, Political4, Political5, Political6, Political7¸ Political8, Political9, Political10, Political11, Political12, Political13, Political14

From the above experimental studies, both techniques give good results for groups of political documents. The political documents were grouped into neuron 2 in the proposed technique and cluster 9 in the hierarchical clustering technique. In addition, the group of education documents are also fairly well clustered by both techniques as the same education documents, Education1, Education2, Education3 and Education4, were grouped into neuron 5 in the proposed technique and cluster 1 in the hierarchical clustering technique. However, it can be seen from this experiment that the groups of education and sport documents are mapped onto the same neuron (Neuron5) in the proposed technique, because they both contain mostly overlapping frequent max substrings such as ผลการแข่งขัน (competition result), การจัดอันดับ (position ranking), ได้รับรางวัล (getting award), etcetera. Meanwhile, travel and education documents were grouped into the same cluster (cluster 1) with the hierarchical clustering technique, because the travel and education documents share many words. Furthermore, some of the education, sport and travel documents are distributed across several small clusters as shown in Table 5.2. In the proposed technique, some errors occurred within the group of travel documents as shown in Table 5.1. The travel documents were mapped onto several neurons due to overlapping terms that appeared across different type of documents.

As observed from the results, the proposed technique can be used to cluster non-segmented documents into several groups according to their similarity. The accuracy of this technique is 83.25 percent, while the accuracy of the hierarchical clustering approach is 79.75 percent. The accuracy in this case is calculated by the number of correctly clustered documents, based on the benchmark results generated from a Thai expert. In the proposed technique, it can also be observed that the documents are clustered into five neurons only, even though this proposed technique is started with nine neurons. In contrast, the hierarchical clustering approach created many small clusters that contain only a few documents and the nine clusters are all used to contain documents. This shows that the difference of clustering result between the proposed technique and the hierarchical clustering approach does not come only from the difference between the frequent max substrings and single words which are used as the indexing terms, but also from the difference between the performance of the proposed technique and the hierarchical clustering approach. The proposed technique can generate empty clusters (neurons), whereas the hierarchical clustering approach cannot do so. As a result, the proposed method may be used to cluster non-segmented documents more efficiently when compared to the hierarchical clustering approach.

Furthermore, the accuracy of the clustering depends very much on the content of documents, and the indexing terms generated. The content of one document may have overlapping frequent terms from two different types of documents. For instance, in the proposed technique, Education5, Travel1, Travel3, Travel11 and Travel14 documents are mapped onto neuron 3 in Table 5.1, because they present information on ecotourism, containing overlapping terms from education and travel documents. Meanwhile, the Education5 document was separated from the groups of education and travel documents in the hierarchical clustering approach because Education5 contains different content, ecotourism, from the content of most education and travel documents as shown in Table 5.2.

An additional advantage of the proposed technique is that it is more computationally efficient than the hierarchical clustering approach. This is because it is a language-independent technique. This means it does not rely on any knowledge of language and does not require a pre-processing step to perform segmentation as describe in Chapter 3. In contrast, the hierarchical clustering approach is language-dependent, and therefore requires the word segmentation technique in order to extract single words from text documents before clustering can be performed.

5.7 Conclusion

This chapter describes a non-segmented document clustering method using the self-organizing map (SOM) and the frequent max substring technique. The frequent max substring technique is first used to discover patterns of interest, called frequent max substrings, rather than individual words from Thai text documents, and these frequent max substrings are then used as indexing terms with their number of occurrences to form a document vector. SOM is then applied to generate a document cluster map by using a document vector. The experimental studies and comparison results on clustering 50 Thai text documents is presented in this chapter. The proposed technique was compared to the hierarchical based document clustering technique, with the use of single words for grouping document occurrences. From the experimental results, the proposed technique can be used to cluster 50 Thai documents into different clusters with an accuracy of 83.25 percent, while the hierarchical clustering approach provides an accuracy of 79.75 percent. The hierarchical clustering approach also created many small clusters that contained only a few documents. As a result, the generated document cluster map from the proposed technique may be used to find documents relevant to a user’s query more efficiently.

Chapter 6

Non-Segmented Text Problems

6.1 Introduction

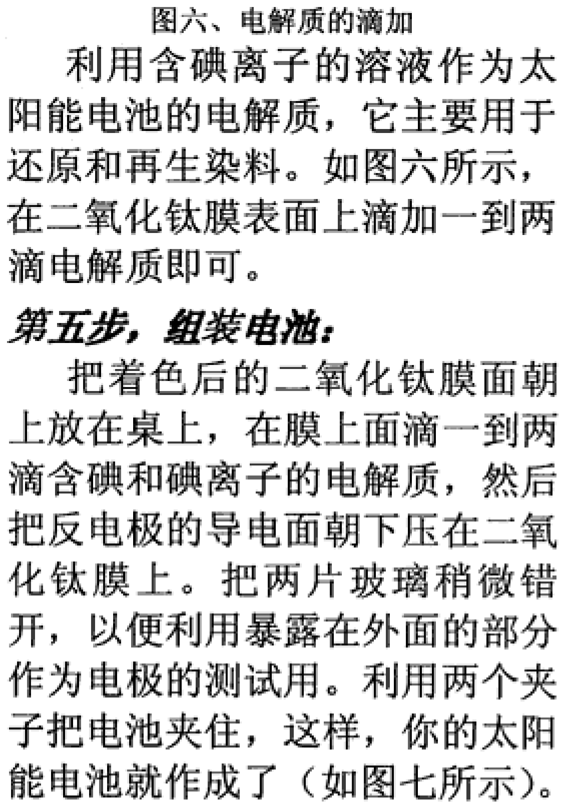
The objective of this chapter is to demonstrate the applicability of the frequent max substring technique presented in Chapter 3 to other non-segmented text problems. Its primary purpose is to show that the frequent max substring technique is not only applicable to Thai text indexing and non-segmented document clustering, but it is also applicable for indexing other non-segmented texts like the Chinese language and genome sequences in bioinformatics. In this chapter, applying the frequent max substring technique to the Chinese language and genome sequencing is described in Sections 6.2.3 and 6.3.3 respectively. The main objective is to show that the frequent max substring technique is a versatile text indexing technique. This chapter is divided into two main sections: non-segmented language problems (Section 6.2) and genome sequencing problems (Section 6.3). Section 6.2 is subdivided into: characteristics of the Chinese language, related works, application of the frequent max substring technique to the Chinese language, experimental studies and comparison results. Meanwhile, Section 6.3 includes: characteristics of the genome sequence, related works, applying the frequent max substring technique to genome sequencing, experimental studies and comparison results.

6.2 Non-segmented language problems

Many natural languages are non-segmented languages. These languages share similar characteristics as the Thai language in terms of the structure of writing. They are written in a string of symbols without explicit word boundary delimiters. This suggests that the frequent max substring technique can be applied for other non-segmented languages. In this chapter, the Chinese language is selected for the illustration because it is one of the non-segmented languages that has been widely used in the world. The Chinese language is similar to the Thai language in many ways, because they are both non-segmented languages. In the Chinese language, each character has its own meaning, so it can be regarded as a word. On the other hand, several Chinese characters can be linked together to make a phrase. A phrase may consist of two, three or more characters, but there are no spaces between Chinese characters except punctuation marks such as ‘,’ or ‘。’ (full stop) [36]. In addition to Chinese, many other Asian languages such as Japanese and Korean are also considered to be non-segmented languages. Words in these languages, which are similar to Thai and Chinese, are not naturally separated by any word delimiting symbols such as white spaces, and in some cases semicolons and commas. Therefore, there are many challenges for indexing non-segmented languages, as outlined in Chapter 2. Many efforts have been devoted to researching and developing indexing techniques for such problems. To demonstrate the use of the frequent max substring technique for the Chinese language, the experimental and comparison results on Chinese text documents are presented in Section 6.2.4. Before the experiment is presented, the characteristics of the Chinese language are first described in the next section, followed by an overview of the related works.

6.2.1 Characteristics of the Chinese language

Among many non-segmented languages, the Chinese language shares the same problem with the Thai language. The Chinese language is considered as un-delimited text, where the structure of writing is a string of symbols without explicit word boundary delimiters [36], [35], [37], [38], [5]. Chinese writings consist of mainly Han characters (hanzi), which are also used in the Japanese language (known as kanji) and in Korean (known as hanja). In modern Chinese, the pictograph words have been simplified by using characters made up of seven strokes (horizontal and vertical strokes, left-falling and right-falling strokes, a point stroke, and a hook stroke) as shown in Figure 6.1 (see http://www.solideas.com/solrcell/chinese.html).


图六、电解质的滴加

利用含碘离子的溶液作为太
阳能电池的电解质，它主要用于
还原和再生染料。如图六所示，
在二氧化钛膜表面上滴加一到两
滴电解质即可。

第五步，组装电池：

把着色后的二氧化钛膜面朝
上放在桌上，在膜上面滴一到两
滴含碘和碘离子的电解质，然后
把反电极的导电面朝下压在二氧
化钛膜上。把两片玻璃稍微错
开，以便利用暴露在外面的部分
作为电极的测试用。利用两个夹
子把电池夹住，这样，你的太阳
能电池就作成了（如图七所示）。

Figure 6.1. Example of Chinese texts

In Chinese texts, words can be composed of one or more characters and a word boundary is not necessarily used between two characters. This means Chinese sentences are continuous strings of characters without white spaces or punctuation marks. Like the Thai language, Chinese does not have variations of words: no changes of tenses, gender and no plural forms. The number of commonly used Chinese characters is around 8,000 to 13,000 characters [135].

In the higher levels, the Chinese language can be classified as a non-segmented language. Due to the reason that it does not have word delimiters, readers have to use their own knowledge to analyze context. The segmentation and indexing techniques used for the Chinese language consist of both the language-dependent and language-independent techniques. In the language-dependent technique, word based and rule based indexing techniques are used for indexing the Chinese language. The language-independent techniques include the character based and *n*-gram based indexing techniques. These techniques are similar to those used in the Thai language as described in Chapter 2. However, well-defined linguistic knowledge for each particular language is still required to perform segmentation and indexing in the language-dependent technique. Meanwhile, character based or *n*-gram based indexing techniques, language-independent techniques, can be directly applied to the Chinese language without the well-defined linguistic knowledge requirement. Furthermore, the frequent max substring technique can also be applied to the Chinese language, as this technique does not require linguistic knowledge of the language. The following section describes some techniques used in the area of text indexing for the Chinese language.

6.2.2 Related works

In order to improve information retrieval systems, many indexing techniques have been proposed for the Chinese language. One such technique is the inverted index technique. The inverted index technique can be regarded either as a language-dependent or language-independent technique, depending on the segmentation method used. To index the Chinese language with the inverted index technique, a segmentation algorithm is often an essential part of the indexing technique. A segmentation algorithm is used to segment text documents into indexing terms before the inverted index can be constructed, as described in Chapter 2. Although indexing terms can be manually identified by human experts for Chinese texts, the process is time consuming and labour intensive in most circumstances. Therefore, segmentation algorithms are used to automatically extract indexing terms from Chinese texts [40]. In Chinese text indexing, several researchers have attempted to develop more efficient techniques of text segmentation to divide text documents into words or terms [40]. At present, the majority of the methods proposed for extracting indexing terms in the Chinese language fall into one of two main categories: character based (CB) and word based (WB), which are both used in the Chinese language [36].

In character based methods, single-gram, bi-gram or tri-gram [60], [61] are used as the indexing terms. However, bi-gram indexing is the most popular technique that is widely used to segment Chinese text documents, since 80 per cent of modern Chinese words are bi-syllabic [62]. The representation of bi-gram is to use all contiguous overlapping 2-character pairs as indexing terms by using the 1-sliding technique for extraction.

In word based (WB) indexing approaches, Foo & Li conducted experiments to study the impact of Chinese word segmentation and its effect on IR. Four automatic character based segmentation approaches and a manual one were used to index and evaluate the accuracy of these approaches. The experiments revealed that the segmentation approach had an effect on IR effectiveness. Better IR results could be achieved by using the same method for query and document processing, which increased the probability of matching queries to documents.

6.2.3 Applying the frequent max substring technique to the Chinese language

In this section, the process of extracting frequent max substrings as indexing terms by using the frequent max substring technique is presented. The same processes as described in Chapter 3 are used to extract indexing terms from Chinese text documents. The given frequency threshold value is set to 2 so that one occurrence of indexing terms can be ignored from the texts, as indexing terms occurring less frequently in text documents could usually be assumed to be insignificant in defining subject matter [77], [69]. The following describes briefly the steps of extracting frequent max substrings as indexing terms from Chinese text documents (For more detail, see Chapter 3)

Let string *s* = '假的作真的时真的亦为假的'

and the given frequency threshold value of $\theta = 2$

1. Extract the frequent substring set, *FSS(s*, θ), with their frequencies and positions. These indexing terms are kept, and are sorted in order of occurrence in the text documents on the index data structure for further processes.

2. Then, the frequent max substring set, *FMAX(s*, θ), is extracted by selecting indexing terms having no superstring from the frequent substring set, *FSS(s*, θ), in order to reduce the number of the indexing terms.

From the above steps, the example of the FST structure that was constructed from Chinese text documents can be shown in Figure 6.2.

Let string *s* = '假的作真的时真的亦为假的'

and the given frequency threshold value or θ = 2

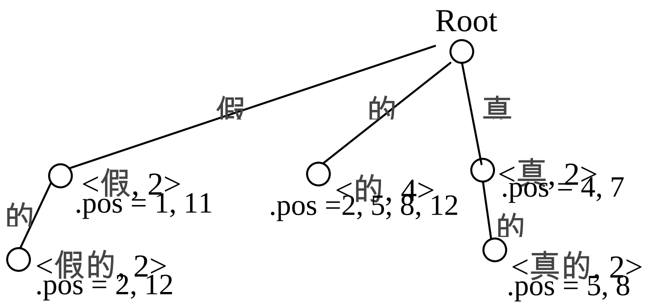


Figure 6.2. FST structure using frequent max substring technique on Chinese text documents

Figure 6.2 shows the FST structure. The result is *FMAX(s*, θ) = {<的, 4>, <假的, 2>, <真的, 2> }

6.2.4 Experimental results

In this section, the experiment of indexing Chinese text documents using the frequent max substring technique is presented. In this experiment, the given frequency threshold value is set to 2, therefore only indexing terms that occur at least two times are of interest. In order to measure indexing efficiency, the frequent max substring technique is compared with bi-gram based indexing in terms of the number of indexing terms and retrieval time. Bi-gram based indexing is chosen because it is one of the most widely used techniques for indexing Chinese text documents [60], [36]. One of the advantages of bi-gram based indexing and the frequent max substring technique is language-independence [111], [136], [7], [26], [65]. As a result, these methods are used for indexing Asian languages [22], [21], [111].

In this experiment, the text collection used for evaluation is a set of Chinese text documents obtained from the website: http://www-personal.umich.edu/~dporter/sampler/sampler.html. The documents have varying lengths. The set of documents consists of 20 text documents and contains 28,522 characters. The document lengths range from 564 to 2,187 characters. In the frequent max substring technique, the set of frequent max substrings is extracted as indexing terms from the set of Chinese text documents, at the given frequency threshold value. As presented in Chapter 3, the proposed frequent suffix trie structure was employed to extract the indexing terms and construct the index in this proposed technique. Table 6.1 shows the number of indexing terms extracted from the frequent max substring technique.

Table 6.1. Number of indexing terms extracted from frequent max substring technique

The frequent max substring technique			
Id	Text name	Text size (*n* characters)	The number of indexing terms
1	Chinese1	564	87
2	Chinese2	612	93
3	Chinese3	732	101
4	Chinese4	856	96
5	Chinese5	915	113
6	Chinese6	1,003	135
7	Chinese7	1,051	129
8	Chinese8	1,108	153
9	Chinese9	1,274	142
10	Chinese10	1,409	136
11	Chinese11	1,500	172
12	Chinese12	1,623	169
13	Chinese13	1,792	199
14	Chinese14	1,874	178
15	Chinese15	1,885	188
16	Chinese16	1,961	203
17	Chinese17	2,013	224
18	Chinese18	2,048	231
19	Chinese19	2,115	269
20	Chinese20	2,187	253

In this experiment, the frequent max substring technique is compared with the bi-gram based indexing technique in order to evaluate indexing efficiency. In bi-gram based indexing, pre-processing is required to extract bi-gram terms before indexing can be performed. The bi-gram term is a substring of *2* overlap or non-overlap successive characters extracted from texts. Extracting a set of bi-gram terms from text documents can be done by using the 1-sliding technique as described in Chapter 2. For instance, let a simple Chinese text document *d* containing the string '假的作真的时真的亦为假' and this document *d* is a string of characters *s1, s2, ..., sN*. Therefore, the *i*th bi-gram term extracted from the document *d* is the substring *si*, *si*+1, ..., *si*+*n*. Figure 6.3 shows the bi-gram terms overlap sequence of the document *d* containing the string *s* '假的作真的时真的亦为假的' .

Let *d*:	假的作真的时真的亦为假的
Bi-gram terms:	假的, 的作, 作真, 真的, 的时, 时真, 真的, 的亦, 亦为, 为假, 假的

Figure 6.3. Example of bi-gram terms from document *d*

For bi-gram based indexing, after bi-gram terms are extracted from the text documents, all tokenized bi-gram terms are then stored in the inverted index for retrieval. The same technique described in Chapter 2 is employed to construct the bi-gram inverted index. The number of indexing terms extracted from the bi-gram based indexing is shown in Table 6.2.

Table 6.2. Number of indexing terms extracted from bi-gram based indexing

The bi-gram based indexing technique			
Id	Text name	Text size (*n* characters)	The number of indexing terms
1	Chinese1	564	62
2	Chinese2	612	68
3	Chinese3	732	87
4	Chinese4	856	94
5	Chinese5	915	102
6	Chinese6	1,003	112
7	Chinese7	1,051	113
8	Chinese8	1,108	131
9	Chinese9	1,274	127
10	Chinese10	1,409	134
11	Chinese11	1,500	148
12	Chinese12	1,623	151
13	Chinese13	1,792	172
14	Chinese14	1,874	162
15	Chinese15	1,885	169
16	Chinese16	1,961	181
17	Chinese17	2,013	208
18	Chinese18	2,048	223
19	Chinese19	2,115	246
20	Chinese20	2,187	237

In order to compare the two indexing techniques: the bi-gram based indexing technique and the frequent max substring technique for Chinese text documents, the number of indexing terms extracted from both techniques is compared. In Figure 6.4, a comparison of bi-gram based indexing and the frequent max substring technique is presented. The vertical axis represents the number of indexing terms and the horizontal axis represents the text document size (*n* characters).

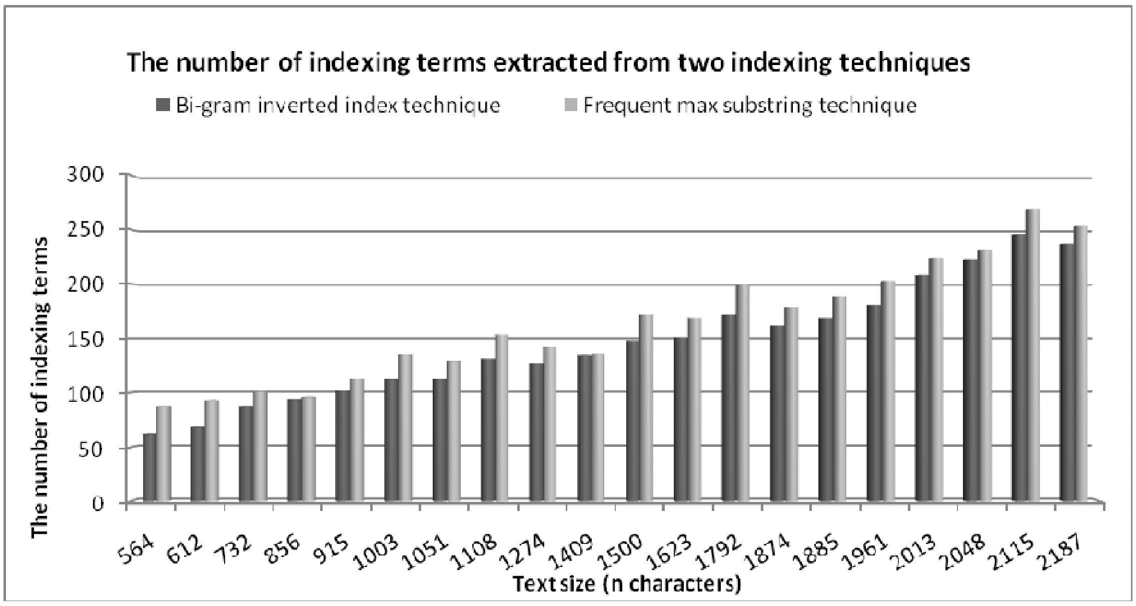


Figure 6.4. Comparison of number of indexing terms extracted from bi-gram based indexing and frequent max substring technique

From the comparison results, it can be observed that the bi-gram based indexing technique extracted a lesser number of indexing terms than the frequent max substring technique. The indexing terms extracted from the bi-gram based indexing technique are small and each term consists of two characters. Most indexing terms are meaningful indicating that bi-syllable is a good assumption for these Chinese text documents. Meanwhile, the frequent max substring technique generates slightly more indexing terms compared to the bi-gram based indexing technique. The indexing terms extracted from the frequent max substring technique have varying lengths. This means more space is needed to store indexing terms compared to the bi-gram based indexing technique. However, the advantage of the frequent max substring technique is that it does not require text pre-processing and query processing before indexing and retrieval can be performed, as mentioned in Chapter 3. One of the drawbacks of the bi-gram based indexing technique is that it requires text pre-processing in order to extract the bi-gram terms by using the 1-sliding technique before indexing can be performed as explained in Chapter 2. Furthermore, query processing is also required in terms of segmentation before retrieval can be performed as described in Chapters 2 and 3. In the bi-gram based indexing technique, the query has to be segmented into bi-gram terms, so that these terms can be used to match the extracted bi-gram terms kept in the index as described in Chapter 3. For instance, if the given query is '假的作真的', it has to be segmented into four bi-gram terms: 假的, 的作, 作真 and 真的 before retrieval. However, query processing is the main factor in making the query complex and requires more retrieval time, as all segmented bi-gram terms are used for looking up the relevant documents at the same time, using Boolean 'AND' as described in Chapter 3. Therefore, the bi-gram based indexing technique usually requires more retrieval time than the word based, rule based and the frequent max substring techniques [26], [10], [39], [10].

In this experiment, the retrieval time of two indexing techniques is also compared. Five queries are used as test queries in order to evaluate retrieval time. All queries can directly be used to look up relevant documents in the frequent max substring technique. Meanwhile, all queries have to be segmented into bi-gram terms before looking up relevant documents in the bi-gram based indexing technique. Table 6.3 shows the average time used by two indexing approaches for retrieving the text collection. Overall, the frequent max substring technique performed better than the bi-gram based indexing in terms of retrieval time.

Table 6.3. Comparison of retrieval time used by bi-gram based indexing and frequent max substring technique

	The frequent max substring technique	**The bi-gram based indexing technique**
Avg. retrieval time	0.9 sec.	2.6 sec.

Consequently, it can be observed that the frequent max substring technique requires more space to store and extract the indexing terms, compared to the bi-gram based indexing technique. However, the bi-gram based indexing technique suffers from poorer retrieval time in looking up relevant documents when compared to the frequent max substring technique.

6.3 Genome sequencing

Over the last decade, genome sequence databases have grown rapidly and have been widely used by molecular biologists for homology searching. The survey shows that the GenBank contains over 77 Gbp (giga, i.e. 10^9, base-pairs) from over 73 million sequence entries [137]. Due to the large amount of data available, the task of providing efficient indexing has become important. It has become critical to develop scalable data management techniques for sequence storage and retrieval. In searching such databases, efficient indexing techniques are essential for indexing a massive amount of sequence data for retrieval. In fact, various algorithms and data structures on strings can be applied to genome sequences because they can be regarded as a sequence of string [138], [139]. The most widely used data structures are suffix trees and suffix arrays as described in Chapter 3. However, it is sometimes difficult to use the conventional suffix based methods for genome sequence databases because of the drawbacks of index sizes.

In this chapter, the frequent max substring technique is also applied to genome sequencing problems as the characteristic of the genome sequence is similar to Thai texts and genome sequencing is not based on human languages. To demonstrate that the frequent max substring technique can be applied to genome sequencing, the experimental and comparison results are presented in Section 6.3.4. Before the illustration is presented, the characteristics of the genome sequence are first described in the next section, followed by some related works.

6.3.1 Characteristics of the genome sequence

In the modern era of molecular biology, the genome sequence can be refer to all of a living thing's hereditary information [140]. This hereditary information is encoded in DNA or RNA, which are used for maintaining, building and running an organism, and passing life on to the next generation. In most organisms, the genome includes genes that are packaged in chromosomes, and the non-coding sequences of the DNA that affects specific characteristics of living things. The genome term was introduced by Hans Winkler, Professor of Botany at the University of Hamburg, Germany, in 1920. This genetic material or DNA can be represented as long texts with a specific alphabet, known as the nucleotide bases, for example, {A, C, G, T} in the genome. Most patterns usually occur frequently in the texts because there is only a four-character alphabet to represent genome sequences. A typical example of the genome sequence is shown in Figure 6.5.

Human PIPSL	TCACCTCTAGTTGAAGAGACTTTGCAAATGCTAACTACAAGT
Chimp PIPSL	TCACCTGTAGTTGGAGAGACTTTGCAAATGCTAACTACAAGT
Human	TCACCTCTAGTTGGAGAGACTTTGCAAATGCTAACTACAAGT
Chimpanzee	TCACCTCTAGTTGGAGAGACTTTGCAAATGCTAACTACAAGT
Cow	TCACCTGTAGTTGGAGAGACTTTGCAAACGCTAACTACAAGT
Mouse	TCACCTGTAGTTGGACAACCTTTGCAAATACTAAATTTGAGT
Dog	TCACCTGTAGTTGGAGAGACTTTGCCAATGCAAACTACAAGC

Figure 6.5. Example of nucleotide structure of some species' genes

In fact, the genome contains many relationships. For instance, the genome is the largest part that can be divided into chromosomes, chromosomes are the smaller parts that contain genes, and inside the genes represents the DNA, which is the smallest part. These relationships can be depicted as shown in Figure 6.6.

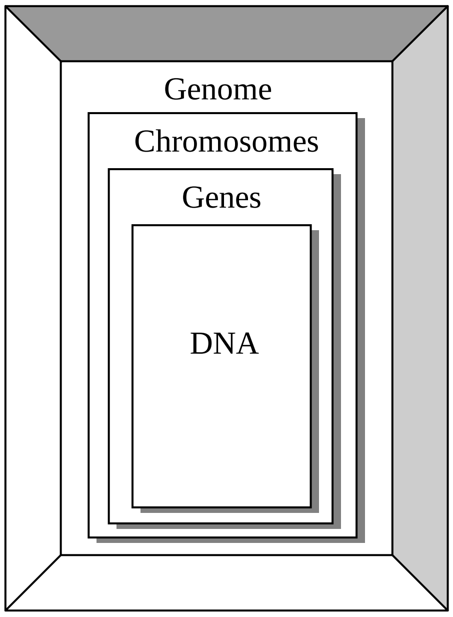


Figure 6.6. Relationships of genome

There are many types of living things in the world that can be divided into many species such as cows, dogs, mice, chimpanzees, humans and so on. These species have their own distinctive genome: the cow genome, the dog genome, the mouse genome, the chimpanzee genome, the human genome and so on. Therefore, genomes can be classified according to species, and can also be used to identify individuals. For example, the genome of people in this world can be classified as the human genome, and each person also has a unique genome and characteristics that can be used to identify individuals. However, two persons may have the same genome if they are identical twins. This significantly shows that the genomes between two persons can be more similar than the genomes between people and other species.

6.3.2 Related works

As mentioned in the previous section, genome sequence databases are increasing in size exponentially. Due to this challenge of the ever increasing data available, many approaches have been proposed for indexing and searching from genomic databases. The basic methods proposed earlier perform a full text search without using indices as described in Chapter 3 [70]. However, one of the drawbacks of this technique is its poor searching ability. As a result, the suffix tree, suffix trie and suffix array data structures have been widely used in biological sequence analysis, because these structures are fundamental data structures for string matching [141], [139], [137]. Unfortunately, the existing basic algorithms for constructing these data structures do not support large inputs when they are used in real-life applications, thus requiring that the input is small enough to be kept in main memory. Therefore, it is difficult to use them for genome-scale databases, because of their massive amount of index sizes. In order to address this particular drawback, many researchers have improved several algorithms based on these data structures in order to handle huge amounts of genome sequence data.

Vilo introduced an algorithm for discovering frequent substrings from biosequences in 1998 [93], [76]. This algorithm systematically generates a pattern trie while maintaining information about the occurrences of each substring. It is basically a generalization of the *wotd* (write-only top-down) suffix trie construction algorithm [79], [91] to find frequent substrings of a string. This technique is interested in substrings that occur at least at the frequency threshold value in the string, by constructing only the subtrees of the suffix trie that correspond to the frequent substrings. This algorithm has been successfully used for analyzing the full genome of yeast and for predicting certain regulatory elements [76].

According to [137], [142], Phoophakdee and Zaki proposed an approach for indexing genome-scale sequences using suffix trees, called TRELLIS+, which effectively scales a large amount of genome sequence data using only a limited amount of main-memory, based on a string buffering strategy. Their works focus on a disk-based suffix tree to develop scalable data management techniques for retrieval, analysis and storage of complete and partial genomes. In this algorithm, the index size is not increased when the input sequence is very large. The experimental results showed that TRELLIS+ outperforms existing suffix tree approaches. Their technique was able to index genome-scale sequences and also allowed rapid searching over the disk-based index.

In 2001, Kunihiko Sadakane and Tetsuo Shibuya presented storing indices in memory in a compressed form [139]. The compressed suffix array was used as a data structure in this technique. The suffix array can be compressed in the same scale of the string in this technique. It stores the suffix array at the cost of a small increase in access time. The experimental results showed that the overhead of using the compressed suffix array is reasonable in practice. By using this technique, the compressed suffix array of a string requires the same index size as the string itself. They also proposed an approximate string matching algorithm in order to support the compressed suffix array.

Hugh E. Williams and Justin Zobel proposed a technique for searching genome sequence databases in 2002, known as the index-based approach for both selecting sequences that display broad similarity to a query and for fast local alignment [143]. Several criteria were applied to satisfy the use of this technique. These indexing and retrieval techniques are embodied in a full-scale prototype retrieval system, CAFÉ, that is based on techniques used in text retrieval and in approximate string matching for databases [144]. The principal features of CAFÉ are the incorporation of data structures for query resolution and the indexing technique used. The experimental studies show that this index-based searching technique provides good results with low computational requirements for local alignments. The index-based searching technique produces results which are comparable with existing exhaustive search schemes.

In 2009, Marina Barsky, Ulrike Stege, Alex Thomo and Chris Upton proposed the external-memory suffix tree construction algorithm for very large inputs [141], known as B^2ST. This algorithm is able to construct suffix trees for input sequences significantly larger than the size of the available main memory [145]. B^2ST minimizes random access to the input string and accesses the disk-based data structures sequentially. It is able to build a disk-based suffix tree for virtually unlimited sizes of input strings, thus filling the ever growing gap between the increase of main memory in modern computers and the much faster increase in the size of genomic databases.

6.3.3 Applying the frequent max substring technique to genome sequencing

In this section, the frequent max substring technique proposed in this thesis is employed to classify frequent max substrings from genome sequencing where the structure is written as a sequence of the specific alphabet in the genome without explicit delimiters. The main objective of doing this is to show that the frequent max substring technique can be applied to indexing genome sequencing. It also allows the construction of the index using the proposed frequent suffix trie data structure as described in Chapter 3.

To illustrate the applicability of the frequent max substring technique for genome sequencing, the following example shows how the proposed frequent suffix trie data structure is used to represent all substrings with their frequency and list of positions of a genome sequence.

Let string *s* = ‘ATGATGT’ as the genome sequence.

And the given frequency threshold value or $\theta = 2$ as most substrings in biosequences usually occur frequently, because there are only four possible characters in the alphabet (A, C, G, T) to represent all genome sequences.

Step 1: Append ‘*$*’ to the string and define the position of each character in the sequence.

String *s* :	A	T	G	A	T	G	T	$
Positions :	1	2	3	4	5	6	7	8

Step 2: Enumerate all suffixes of the string.

1:	ATGATGT$
2:	TGATGT$
3:	GATGT$
4:	ATGT$
5:	TGT$
6:	GT$
7:	T$
8:	$

Step 3: All suffixes are used to create the frequent suffix trie structure, as shown in Figure 6.7.

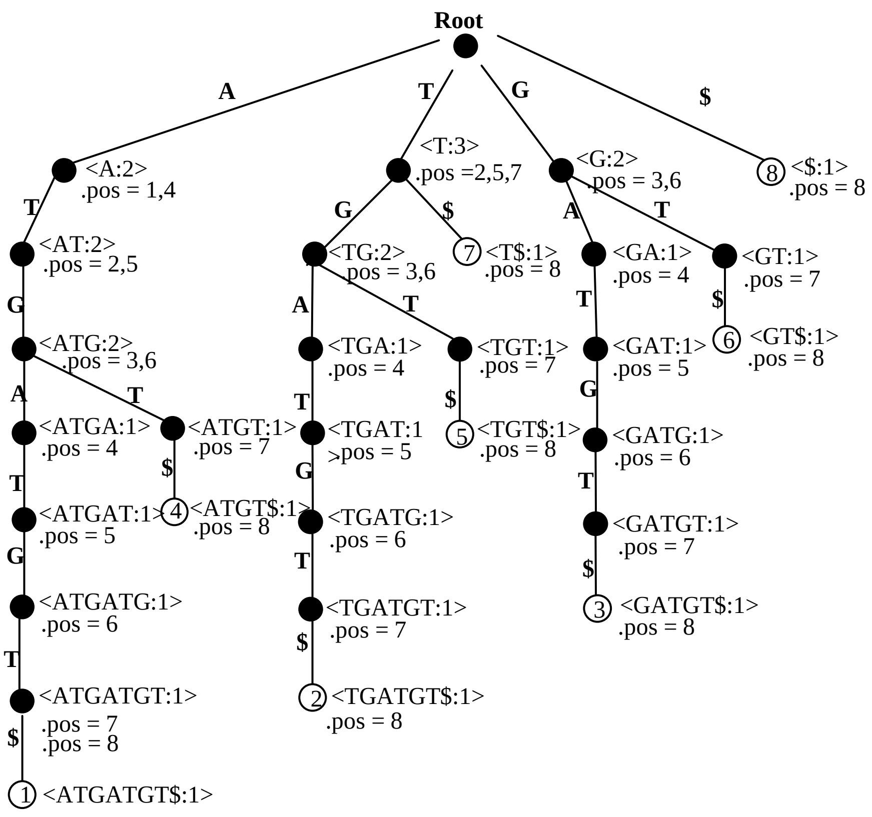


Figure 6.7. Frequent suffix trie structure of string *s* = 'ATGATGT'

The frequent max substring technique that was described in Chapter 3 is used to extract frequent max substrings from the genome sequence using the min-heap structure as shown in the following steps.

Min-heap structure

Firstly, all substrings with a length of 1 are extracted, together with their frequencies and list of positions. The frequencies of these substrings are then checked in order to select only the frequent substrings with a length of 1. These frequent substrings are finally kept in the min-heap structure for further processes.

A, 2 .pos=1, 4	T, 3 .pos=2, 5, 7	G, 2 .pos=3, 6

Next, <A, 2> is removed from min-heap in order to indicate that <A, 2> is detected and extracts its child substrings for the next process. After <A, 2> is removed from min-heap, the algorithm extracts child substrings of <A, 2> using list of positions or pointers of <A, 2> to reduce time complexity. Child substrings consist of <AT, 2>. <AT, 2> is kept in min-heap using the insertion rule, because <AT, 2> is the substring that occurs in two different positions in string *s*.

T, 3 .pos=2, 5, 7	AT, 2 .pos=2, 5	G, 2 .pos =3, 6

<T, 3> is removed from min-heap, after which child substrings of <T, 3> are extracted using the list of positions or pointers of <T, 3>. Child substrings consisting of <TG, 2> and <T$, 1>. <G, 2> are deleted from min-heap because <TG, 2> is a proper superstring of <G, 2> at the same frequency, and <TG, 2> is kept in min-heap instead, using the insertion rule, because its frequency is equal to the given frequency threshold value.

AT, 2 .pos=2, 5	TG, 2 .pos =3, 6

<AT, 2> is removed from min-heap and then its child substrings are extracted using its list of positions (pointers). They consist of <ATG, 2>. <TG, 2> is deleted from min-heap because <TG, 2> is a substring of <ATG, 2> with the same frequency. After that, <ATG, 2> is kept in min-heap using the insertion rule because <ATG, 2> is the substring that occurs in two different locations in string *s*.

ATG:2 .pos =3, 6	

<ATG, 2> is removed from min-heap and then its child substrings are extracted using its list of positions. They consist of <ATGA, 1> and <ATGT, 1>. They are not kept in min-heap because their frequencies are less than *θ*.

The algorithm will stop when min-heap is empty. This means all substrings in min-heap were detected and processed completely.

From the above algorithm, the resulting frequent suffix trie structure can be depicted in Figure 6.8.

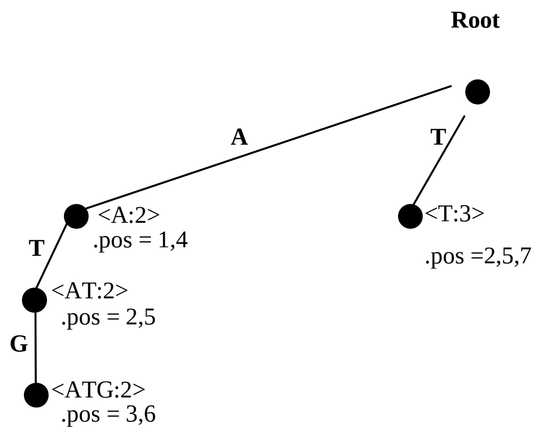


Figure 6.8. Frequent suffix trie structure using proposed frequent max substring technique

Figure 6.8 shows the frequent suffix trie structure. The result is *FMAX(s, θ)* = {<T, 3>, <ATG, 2>}

From observation, the frequent max substring set, *FMAX,* is able to contain all frequent substrings as shown in Table 6.4.

Table 6.4. Number of frequent max substrings and frequent substrings

Frequent max substrings	Frequent substrings
<T, 3>	<T, 3>
<ATG, 2>	<A, 2>
	<G, 2>
	<AT, 2>
	<TG, 2>
	<ATG, 2>

It can be observed from Table 6.4 that <T, 3>, <A, 2>, <G, 2>, <AT, 2>, <TG, 2> and <ATG, 2> are substrings of <T, 3> and <ATG, 2>, which are extracted from the frequent max substring technique. The indexing terms <A, 2>, <G, 2>, <AT, 2>, <TG, 2> and <ATG, 2> can be enumerated from indexing term <ATG, 2>, while <T, 3> can be enumerated from <T, 3>. It is considered that <T, 3> and <ATG, 2> can be kept, instead of keeping <A, 2>, <G, 2>, <AT, 2> and <TG, 2> in order to reduce index size and the number of indexing terms.

6.3.4 Experimental studies and comparison results

To illustrate the feasibility of using the frequent max substring technique for genome sequencing, a comparison experiment was carried out to compare the performance from Vilo's method. Vilo's technique is selected here because this technique aims to find frequent substrings in order to reduce index size and the number of indexing terms. Meanwhile, other techniques focus more on compression and merging the data structure or memory in order to support the growth of genome sequence databases. In order to compare the two different algorithms for genome sequencing, the number of indexing terms that are enumerated by using Vilo's algorithm and the number of indexing terms that are enumerated by using the frequent max substring technique are compared. The dataset used for this evaluation is the genome sequence found on the website: http://www.broadinstitute.org/cgi-bin/annotation/methanosarcina/download-sequence.cgi. Genome sequences have various lengths. The set of genome sequences consists of 20 sequences and contains 52,500 characters. The sequence lengths start from 250 to 5,000 characters.

From Figure 6.9 to Figure 6.17, the results of the comparison of the two indexing methods used in this experiment are presented. The given frequency threshold values were set between two and ten, so that the difference number of the indexing terms can be examined when the given frequency threshold value varied. (For more information on the dataset for different thresholds refer to Appendix F). Note that two is the common frequency threshold value used to find frequent substrings [76]. In this experiment, the given frequency threshold values stop at ten because the difference between the number of indexing terms in the two techniques starts to be very small at ten. In Figure 6.9 to Figure 6.17, the vertical axis represents the number of indexing terms extracted by the two different approaches: Vilo’s algorithm and the frequent max substring technique. The horizontal axis represents the sequence lengths.

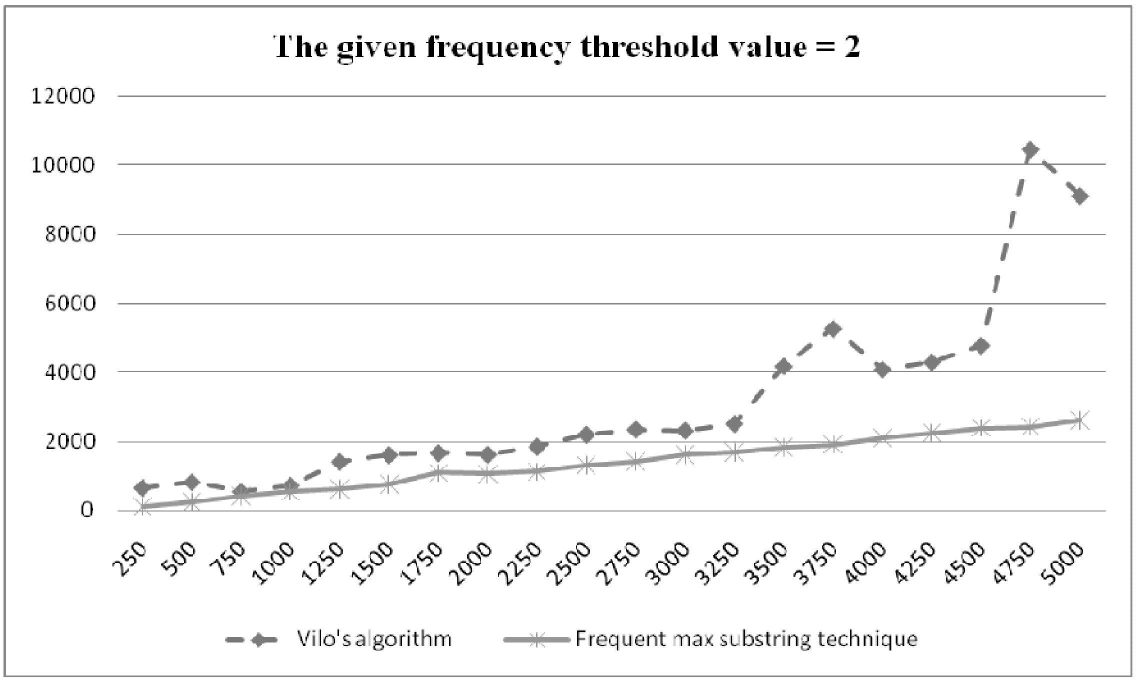


Figure 6.9. Comparison of number of indexing terms extracted from two approaches at given frequency threshold value = 2

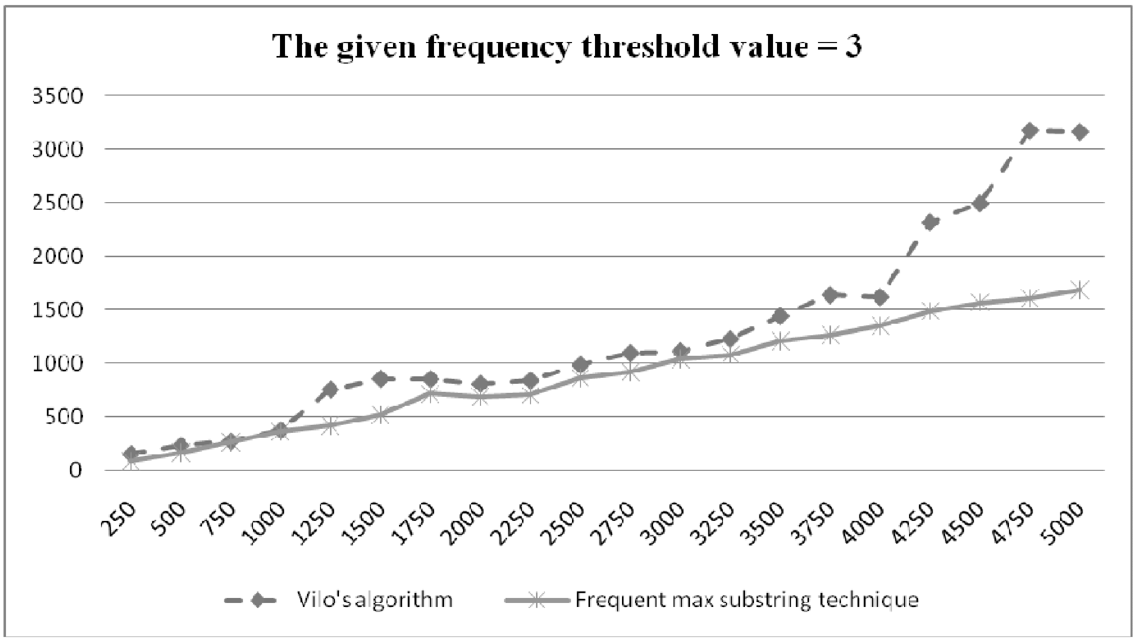


Figure 6.10. Comparison of number of indexing terms extracted from two approaches at given frequency threshold value = 3

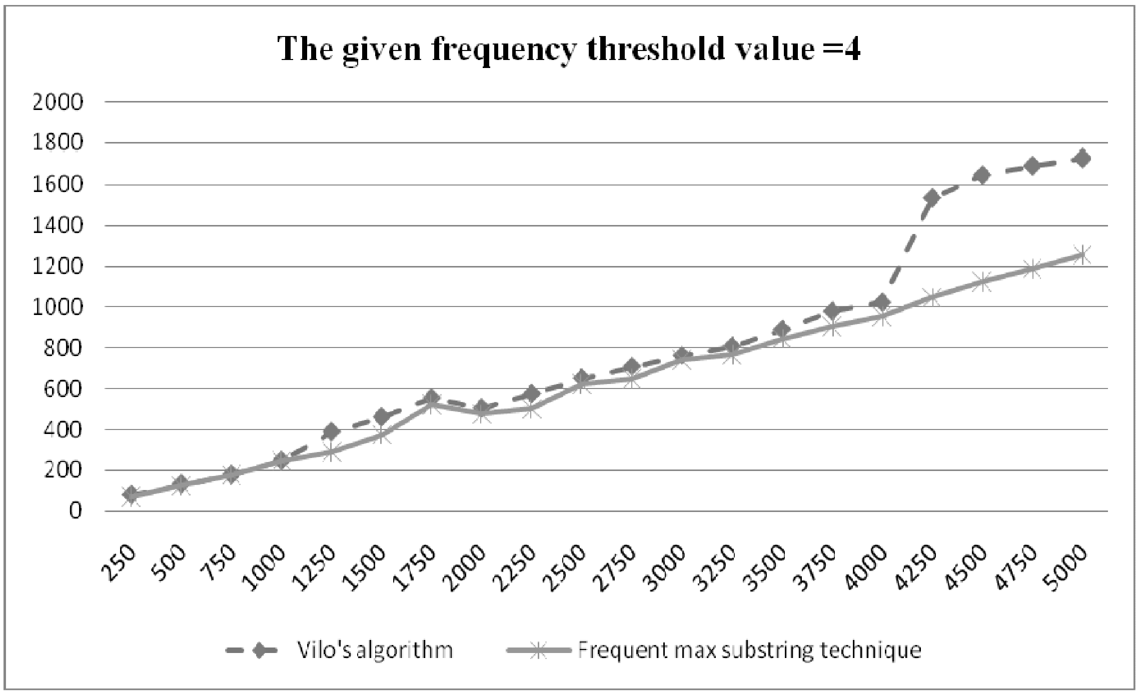


Figure 6.11. Comparison of number of indexing terms extracted from two approaches at given frequency threshold value = 4

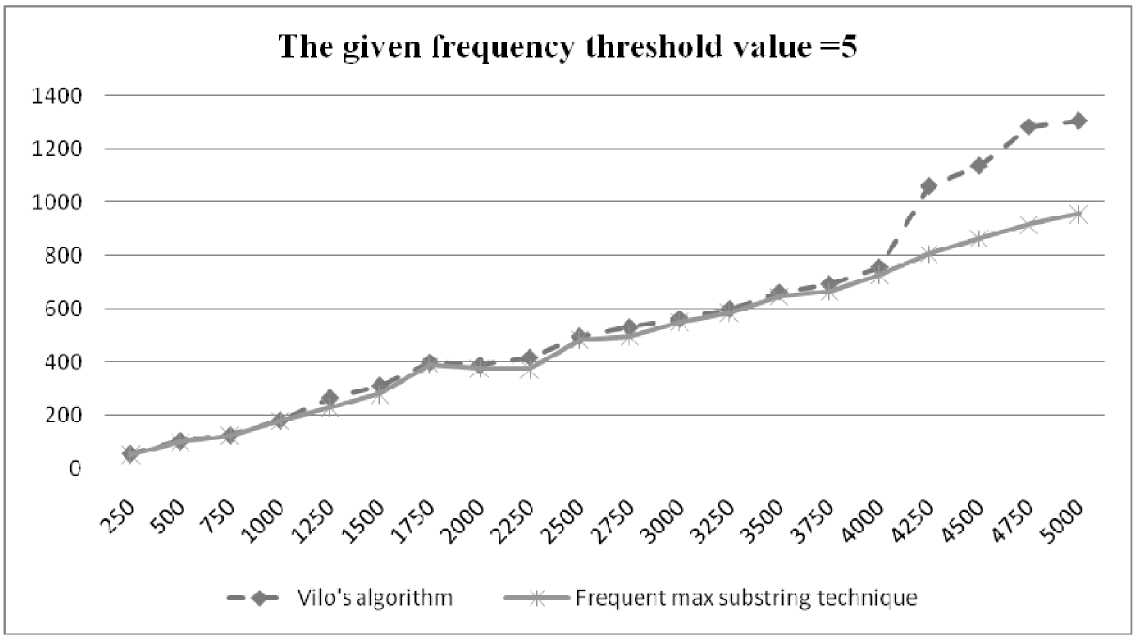


Figure 6.12. Comparison of number of indexing terms extracted from two approaches at given frequency threshold value = 5

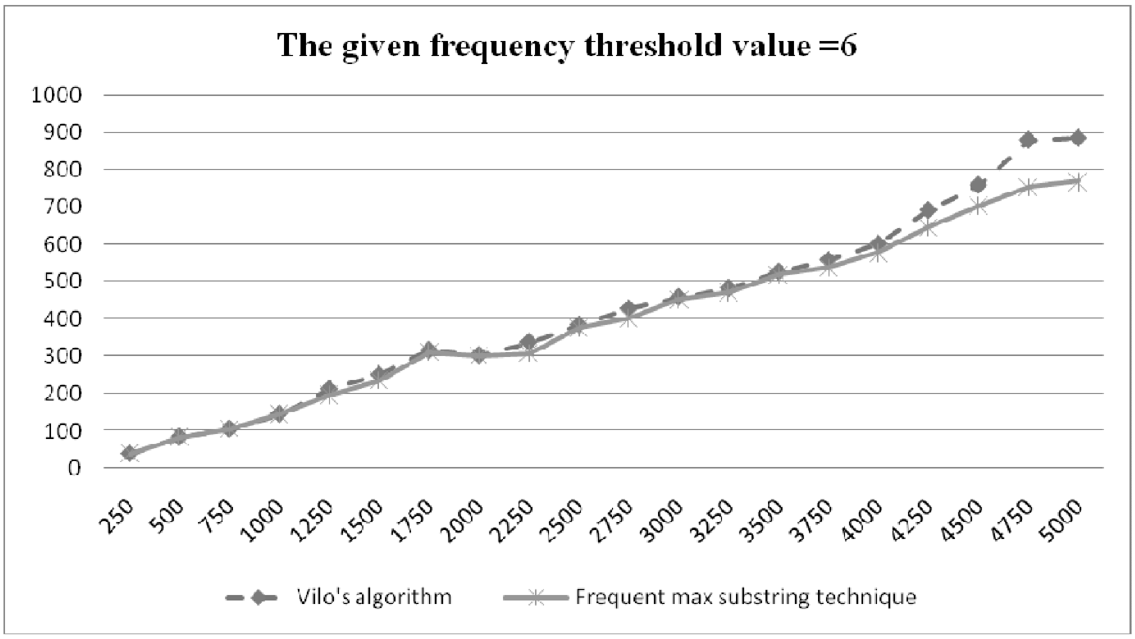


Figure 6.13. Comparison of number of indexing terms extracted from two approaches at given frequency threshold value = 6

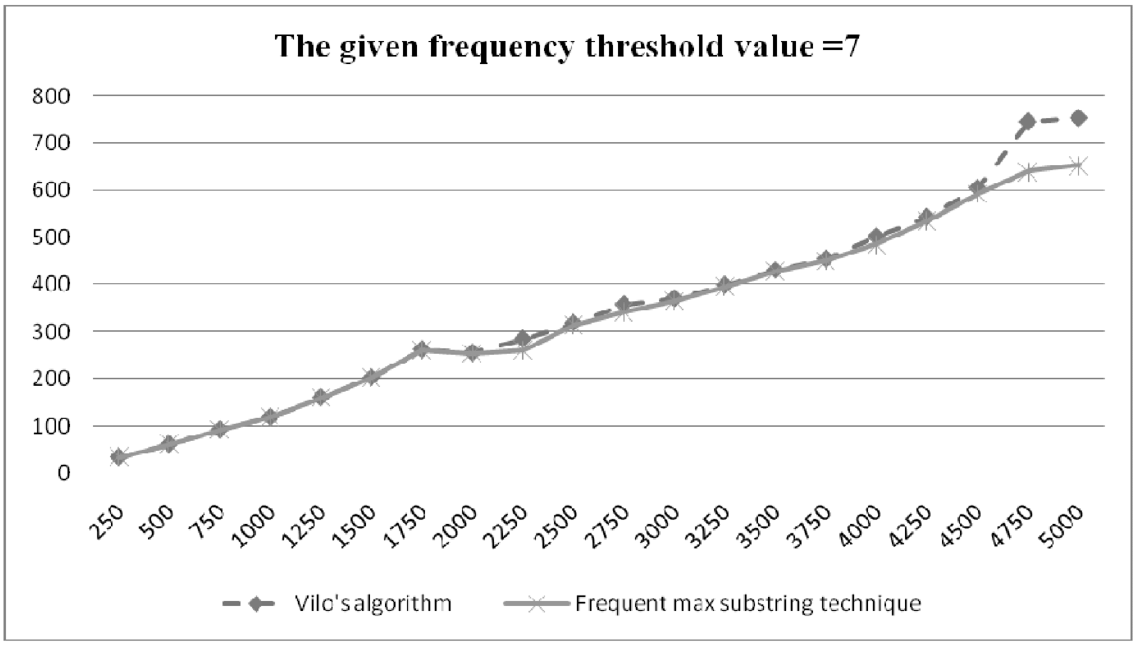


Figure 6.14. Comparison of number of indexing terms extracted from two approaches at given frequency threshold value = 7

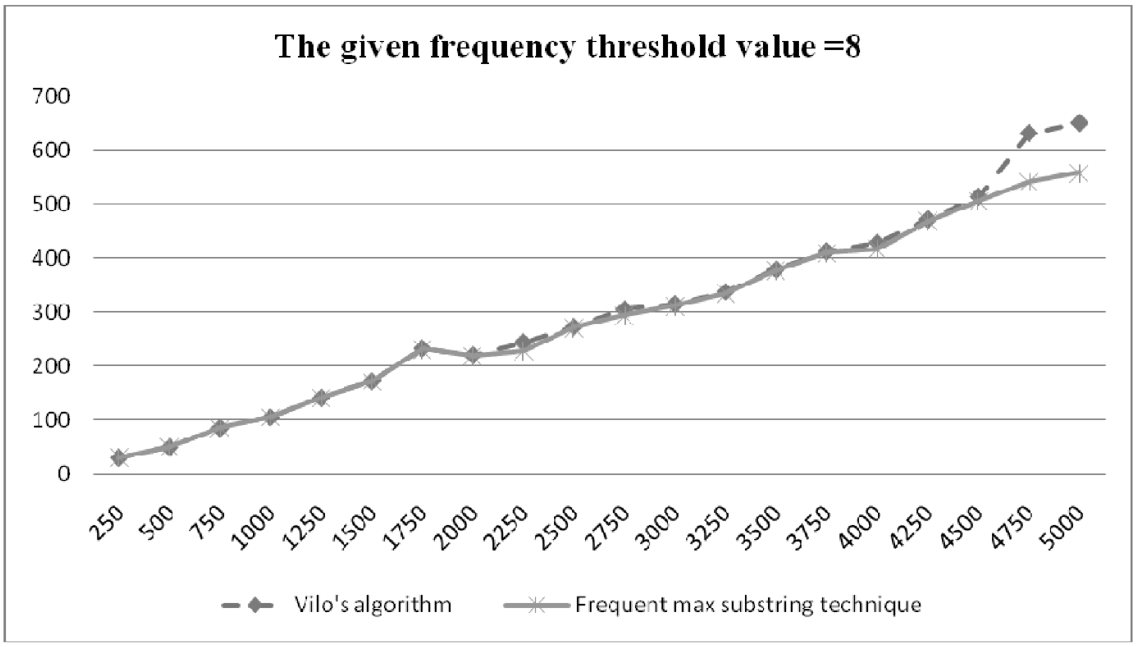


Figure 6.15. Comparison of number of indexing terms extracted from two approaches at given frequency threshold value = 8

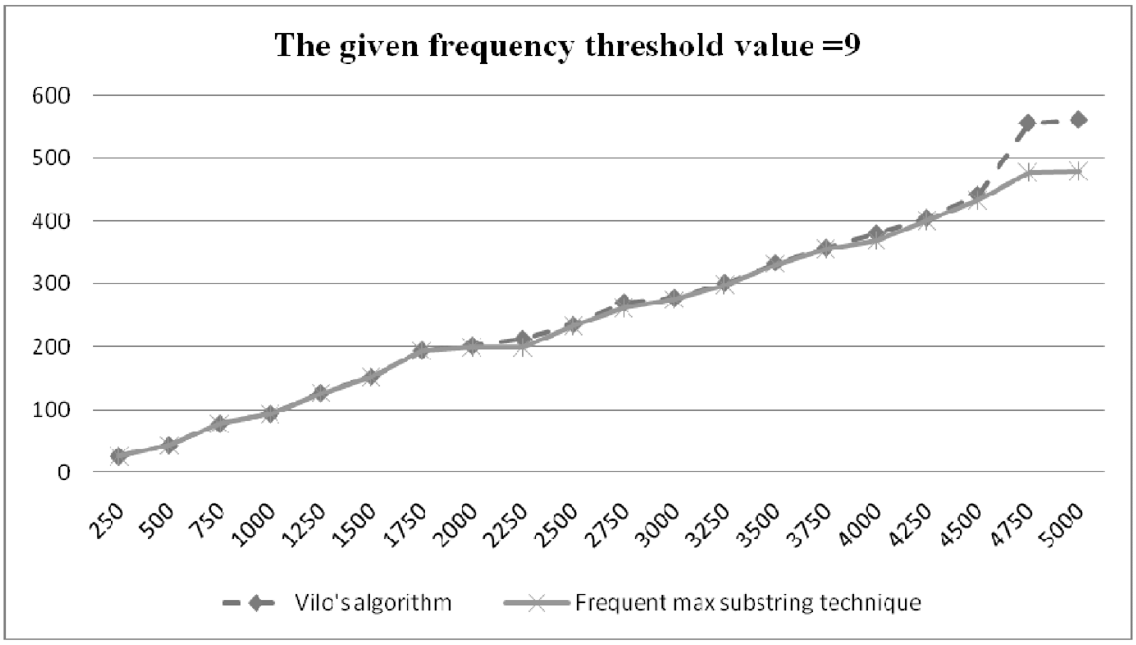


Figure 6.16. Comparison of number of indexing terms extracted from two approaches at given frequency threshold value = 9

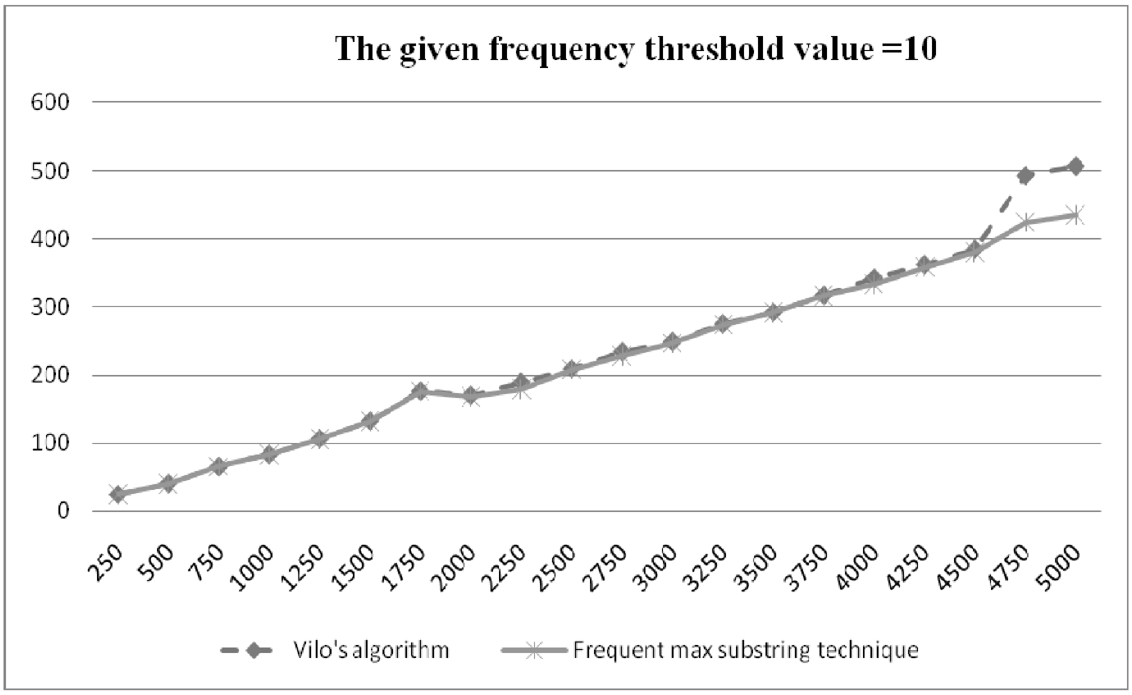


Figure 6.17. Comparison of number of indexing terms extracted from two approaches at given frequency threshold value = 10

The results showed that Vilo's algorithm extracted a greater number of indexing terms than the frequent max substring technique at the low given frequency threshold value. All indexing terms extracted from Vilo's algorithm were contained by all indexing terms extracted from the frequent max substring technique as described in Chapter 3. However, the number of indexing terms is likely to be closer to each other when the given frequency threshold value increases. In order to support the experimental studies, the reduction rate of both algorithms was also compared. The conventional suffix trie algorithm that was described in Chapter 3 was used as the Naïve method and compared with the two approaches. The reduction rate of the number of indexing terms using the conventional suffix trie algorithm is equal to 0 per cent as this technique extracted the complete set of indexing terms (substrings) from strings. The reduction rate can be evaluated using the measurement in the proportion of the number of indexing terms extracted from the conventional suffix trie and the number of indexing terms extracted from other compared techniques. Therefore, the proportion of the reduction rate using two techniques can be shown in Figure 6.18, when compared to the conventional suffix trie algorithm.

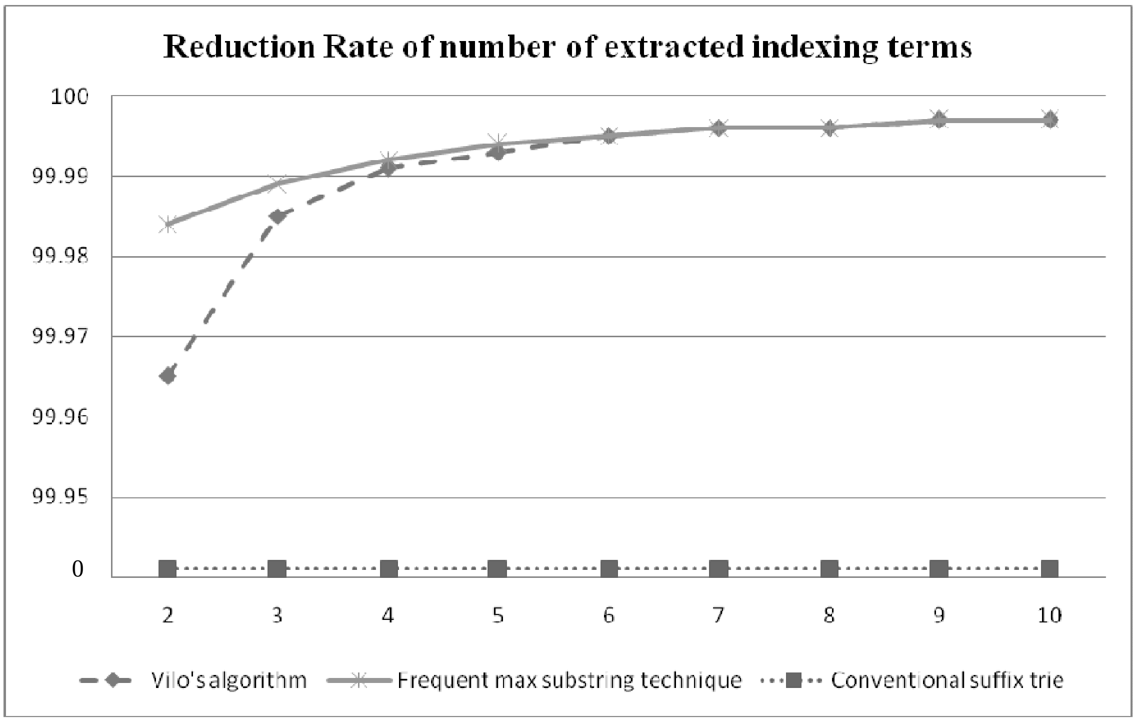


Figure. 6.18. Reduction rate of number of indexing term enumerations using frequent max substring technique and Vilo's technique when compared with conventional suffix trie algorithm

From Figure 6.18, the experimental results showed that the frequent max substring technique can increase the reduction rate of the number of indexing terms by up to 99.98 percent at low frequency threshold value, and the reduction rate has increased slightly at the higher frequency threshold value. Meanwhile, the reduction rate of Vilo's technique is lower than the reduction rate of the frequent max substring technique, although Vilo's algorithm also provided a high reduction rate of up to 99.96 percent at low frequency threshold value. However, it can be observed that the reduction rate of both algorithms is likely to be closer when the frequency threshold value is higher. In addition, the reduction rate of the number of indexing terms for both algorithms increases according to the given frequency threshold values and the maximum size of indexing terms.

6.4 Conclusion

The purpose of this chapter is to demonstrate the applicability of the frequent max substring technique. The experiment results show that the frequent max substring technique can be applied not just to Thai text indexing or non-segmented document clustering but it can also be applied for indexing other non-segmented texts like the Chinese language and the genome sequence in bioinformatics. These non-segmented texts share the same features with the Thai language in terms of structure. The frequent max substring technique can be applied to other non-segmented texts because this proposed technique is language-independent and can be applied to any pattern or structure as described in Chapter 3. In this chapter, applying the frequent max substring technique to the Chinese language and genome sequencing was demonstrated. From the experiment and comparison results, it can be observed that the frequent max substring technique is a versatile technique that can also be applied to other non-segmented text problems.

Chapter 7

Conclusions

7.1 Contribution and outcomes

This study has focused on a problem concerning the development of Thai text indexing techniques. The research in this study has provided a better understanding of indexing Thai text documents based on language-dependent and language-independent techniques. To conclude this research, it can be summarized by the following:

➢ **Successfully developed an efficient Thai text indexing technique:**

A new text indexing technique, called the frequent max substring technique, is proposed for indexing Thai text documents. This technique is used to extract indexing terms and construct an index for Thai text documents. The proposed technique addresses several challenges that arise from indexing Thai text documents via language-dependent and language-independent techniques.

It is demonstrated that the use of the proposed technique can improve performance in terms of construction time, as this proposed technique is language-independent (i.e. does not require the use of a dictionary or corpus or grammatical knowledge of language). The proposed frequent max substring technique also does not require any text pre-processing in extracting indexing terms before indexing can be performed. This is why the proposed technique is fast and simple when compared to language-dependent techniques. When using a language-dependent technique like the word inverted index technique, one of the main drawbacks is that it requires well-defined linguistic knowledge or the use of a dictionary or a corpus in order to extract indexing terms before constructing an index.

Among the language-independent techniques, the proposed technique also improves space efficiency in order to address one of the drawbacks of some language-independent techniques. The proposed technique extracts only strings that are both long and frequently occurring, known as frequent max substrings, from Thai text documents as indexing terms, in order to reduce the number of insignificant indexing terms from an index. Due to this reason, the proposed technique is able to construct the index using less storage space to facilitate more efficient Thai text retrieval.

In the evaluation of the frequent max substring technique, it is compared to four other indexing techniques: the word inverted index, the n-gram inverted index, the Vilo technique and the suffix array. The experiments were performed by indexing 50 Thai web pages. Five indexing techniques were compared in terms of indexing efficiency and retrieval performance.

From the indexing efficiency evaluation, it can be concluded that the frequent max substring technique requires less space to store indexing terms than the suffix array and Vilo's techniques. The number of indexing terms extracted from the frequent max substring technique is around 21.50 percent of the number of indexing terms extracted from the suffix array technique, and is around 13.63 percent of the number of indexing terms extracted from Vilo's technique. Meanwhile, the frequent max substring technique provides a similar number of indexing terms and index size when compared to the word inverted index and 3-gram inverted index techniques. The number of indexing terms and index size generated by the inverted index, 3-gram inverted index and frequent max substring techniques are much smaller, around 89 percent, when compared to the suffix array and Vilo’s approaches. The frequent max substring technique is also more efficient in computation when compared to the word inverted index, as the frequent max substring technique does not require text pre-processing in segmenting text documents like the word inverted index technique does. The results of the experiment in Chapter 3 have shown that the frequent max substring technique uses less indexing time than the word inverted index technique. The indexing time used by the frequent max substring technique is around 70.47 percent of the indexing time used by the word inverted index technique. In the word inverted index, text documents need to be parsed and tokenized into individual linguistic units using word segmentation approaches, which are very time consuming depending on the segmentation algorithm.

In retrieval performance evaluation, results have shown that the frequent max substring technique provides the best precision and recall values over other existing techniques used in the comparison study. This is because the proposed technique extracts frequent max substrings, which can normally describe the content of documents better. Additionally, the frequent max substring technique, Vilo’s technique and suffix array approach provide similar recall results and these three techniques can find all documents that contain the query. Meanwhile, the precision and recall values for the word inverted index and 3-gram inverted index techniques are subject to the quality of segmentation, (i.e. the splitting of general terms). This is because the word inverted index and 3-gram inverted index techniques require query processing using segmentation before searching. However, lower precision and recall values from the word inverted index and 3-gram inverted index techniques are usually caused by query processing, because the set of split terms normally cannot be used to exactly specify what the user needs. Furthermore, another advantage of the frequent max substring technique, Vilo’s technique and suffix array approach over the word inverted index and 3-gram inverted index techniques is that the given query can directly be used to search relevant documents without query pre-processing. In addition, the frequent max substring technique, Vilo’s technique and suffix array approach are also able to search relevant documents even when the query was segmented into single words using the query processing technique. Meanwhile, the word inverted index and 3-gram inverted index techniques require query processing for all cases.

- **Proposed data structure to facilitate fast and accurate enumeration of substrings to support the frequent max substring technique:**

In order to efficiently extract indexing terms and construct the index, the frequent max substring technique used a proposed data structure, called the frequent suffix trie or FST structure, to ensure exhaustive enumeration of substrings to support the extraction of frequent max substrings as indexing terms. The proposed data structure facilitates the extraction of substrings, together with their occurrence information and frequency, while the conventional suffix trie extracts only substrings without any information [47]. In practice, the heap data structure is employed to compute the frequent max substrings by using the two reduction rules to reduce storage requirement and the time required for extracting the frequent max substrings.

- **Successfully developed the frequent max substring technique to perform Thai text indexing for language-dependent and language-independent techniques:**

This thesis also shows that the frequent max substring technique can also be combined with other Thai language-dependent techniques to become a hybrid language-dependent technique. The hybrid method is used for extracting and indexing meaningful indexing terms from Thai text documents. From the experimental results, the hybrid method can be used to reduce the number of indexing terms and index sizes when compared to the frequent max substring technique, and it provides better retrieval performance than the word inverted index technique. Consequently, the hybrid method could be used as an alternative language-dependent technique for indexing Thai text documents.

- **Successfully developed an integrated method using the proposed frequent max substring technique with self-organising map (SOM) to enhance non-segmented document clustering:**

This thesis also proposes a non-segmented document clustering method using self-organizing map (SOM) and the frequent max substring technique to improve the efficiency of information retrieval. This demonstrates that the frequent max substring technique can be used with other techniques to enhance clustering of non-segmented documents. In the evaluation of the proposed document clustering technique, experimental studies and comparison results on clustering of Thai text documents which consist of non-segmented texts are presented. The proposed technique is compared to the hierarchical clustering approach, which is the more widely used document clustering technique in the Thai language. The results have shown that the proposed technique can be used for clustering Thai text documents. The generated document cluster map from the proposed technique can be used to find the relevant documents according to a user's query more efficiently, when compared to the hierarchical clustering approach.

- **Successfully applied the proposed technique with some other non-segmented texts like Chinese and genome sequences:**

Finally, the applicability of the frequent max substring technique to other non-segmented text problems has been demonstrated in this thesis. Due to being a language-independent technique, the frequent max substring technique can be very versatile. It is not only applicable for Thai text indexing and non-segmented document clustering, but it is also applicable for indexing other non-segmented texts like the Chinese language and genome sequences in bioinformatics. To demonstrate the use of the frequent max substring technique for these non-segmented texts (the Chinese language and genome sequences), the experimental and comparison results are presented. The results have shown that the frequent max substring technique requires less retrieval time than the bi-gram based indexing technique in looking up relevant documents in the Chinese language. The retrieval time used by the frequent max substring technique is around 33 percent of the retrieval time used by the bi-gram based indexing technique. This is because the bi-gram based indexing technique needs query processing. Meanwhile, the frequent max substring technique also requires less space for storing indexing terms, and constructing the index than the Vilo method, in indexing genome sequences. The space for storing indexing terms used by the frequent max substring technique is around 84.63 percent of the space for storing indexing terms used by the Vilo's technique in indexing genome sequences.

However, this research also revealed some underlying limitations of the proposed technique. One of the drawbacks of the proposed frequent max substring technique is the indexing time required for constructing an index when compared to other language-independent techniques. This limitation is caused by the use of two reduction rules as described in Section 3.4.2 in Chapter 3, in order to reduce the number of indexing terms during the process. However, the indexing time is dependent on the given frequency threshold value and the size of the maximum indexing terms.

7.2 Future work and directions

This thesis is mainly designed to develop an efficient Thai text indexing technique in order to enhance the performance of Thai text indexing. From the outcomes of this research, there are some possible directions in which this research could be continued in the future. The following are discussions and recommendations of some possible future work to expand the knowledge in this area.

1. One of the disadvantages of the frequent max substring technique is the indexing time required for constructing the index. More detailed analysis and evaluation are necessary to formulate alternatives in the extraction algorithm, in order to reduce the indexing time used by the frequent max substring technique. However, a balance needs to be taken into account to find the optimum point between better performance and better indexing time.

2. The continual development of the frequent max substring technique could be expanded to include the search algorithm. A new search algorithm could be developed in the future in order to find an appropriate searching technique for the frequent max substring technique. This would lead to the improvement of the frequent max substring technique in terms of search ability.

This study has proposed the use of the frequent max substring technique to address the indexing problem for the Thai language and to enhance the performance of Thai text indexing. It is believed that the research and experimental studies from this thesis have contributed to the improvement of the performance in several practical applications. It is also hoped that further work will continue in order to improve the efficiency of Thai text indexing.

APPENDIXES

APPENDIX A: Addresses of Thai text collection

Id	Name	Address of data set
1	sport2	http://www.astv-tv.com/news1/viewdata.php?data_id=1001731
2	sport1	http://www.phuketcity.go.th/pkm/index.php?option=com_content &task=view&id=1620&Itemid=97&lang=th_TH
3	sport4	http://news.mjob.in.th/sport/cat10/news19541/
4	sport3	http://radiothailand.prd.go.th/chonBuri/040newsboard/aspboard_ Question.asp?GID=356
5	travel4	http://news.sanook.com/scoop/scoop_361899.php
6	sport7	http://www.seagames2007.th/th/index.php?option=com_content &task=view&id=1149&Itemid=107
7	sport9	http://www.seagames2007.th/th/index.php?option=com_content &task=view&id=1154&Itemid=107
8	travel3	http://www.ryt9.com/s/prg/258847/
9	sport15	http://cfd-press.blogspot.com/2009_06_01_archive.html
10	travel11	http://www.siamfreestyle.com/forum/index.php?showtopic=1918
11	sport6	http://www.seagames2007.th/th/index.php?option=com_content &task=view&id=1118&Itemid=107
12	travel13	http://www.scholiday.co.th/scwb/viewtopic.php?f=4&t=90
13	sport5	http://news.mumuu.com/sport/page2/
14	education2	http://www.intel.com/cd/corporate/education/apac/tha/news/3195 61.htm#Link4
15	political1	http://www.spokesman.go.th/tape/410721t.txt
16	education4	http://www.intel.com/cd/corporate/education/apac/tha/news/3195 61.htm#Link2
17	travel2	http://travel.sanook.com/news/news_07858.php
18	travel1	http://www.welcomethai.com/hotelreservation/news.asp?id=11
19	sport8	http://www.seagames2007.th/th/index.php?option=com_content &task=view&id=1147&Itemid=107
20	political3	http://www.matichon.co.th/matichon/view_news.php?newsid=01 col01210652&sectionid=0116&day=2009-06-21
21	travel15	http://www.phuketall.com/news/news.php?id=2785
22	political14	http://www.serichon.com/board/index.php?topic=26662.0
23	travel12	http://thai.tourismthailand.org/festival-event/content-6781.html
24	travel6	http://thai.tourismthailand.org/news/content-1491.html
25	political13	http://www.atnnonline.com/atnnonline/index.php/component/cont ent/article/41-special-report/167-2009-05-28-08-00-13.html
26	education5	http://www.intel.com/cd/corporate/education/apac/tha/news/3195

		61.htm#Link1
27	political9	http://vivaldi.cpe.ku.ac.th/~note/event/?id=397567
28	political15	http://th.wikipedia.org/wiki/%E0%B8%81%E0%B8%B2%E0%B8%A3%E0%B8%82%E0%B8%B1%E0%B8%9A%E0%B9%84%E0%B8%A5%E0%B9%88%E0%B8%97%E0%B8%B1%E0%B8%81%E0%B8%A9%E0%B8%B4%E0%B8%93_%E0%B8%8A%E0%B8%B4%E0%B8%99%E0%B8%A7%E0%B8%B1%E0%B8%95%E0%B8%A3_%E0%B8%88%E0%B8%B2%E0%B8%81%E0%B8%95%E0%B8%B3%E0%B9%81%E0%B8%AB%E0%B8%99%E0%B9%88%E0%B8%87%E0%B8%99%E0%B8%B2%E0%B8%A2%E0%B8%81%E0%B8%A3%E0%B8%B1%E0%B8%90%E0%B8%A1%E0%B8%99%E0%B8%95%E0%B8%A3%E0%B8%B5
29	political5	http://203.151.20.17/news_detail.php?newsid=1218170712
30	education3	http://www.intel.com/cd/corporate/education/apac/tha/news/319561.htm#Link16
31	education1	http://www.intel.com/cd/corporate/education/apac/tha/news/319561.htm#Link23
32	travel14	http://www.scholiday.co.th/scwb/viewtopic.php?f=4&t=90
33	sport13	http://www.marketeer.co.th/inside_detail.php?inside_id=6833
34	sport10	http://www.ryt9.com/s/prg/285415/
35	sport11	http://www.thaipr.net/nc/readnews.aspx?newsid=48B88F177C5D488E79B59DD02C018C2F
36	travel10	http://www.ryt9.com/s/prg/130748/
37	political2	http://www.freemarketthai.com/tag/%E0%B8%84%E0%B8%A7%E0%B8%B2%E0%B8%A1%E0%B8%82%E0%B8%B1%E0%B8%94%E0%B9%81%E0%B8%A2%E0%B9%89%E0%B8%87%E0%B8%97%E0%B8%B2%E0%B8%87%E0%B8%81%E0%B8%B2%E0%B8%A3%E0%B9%80%E0%B8%A1%E0%B8%B7%E0%B8%AD%E0%B8%87/
38	political8	http://www.suthichaiyoon.com/WS01_A001_news.php?newsid=7656
39	political4	http://www.naewna.com/news.asp?ID=142987
40	political12	http://politicalbase.in.th/index.php/%E0%B8%8A%E0%B8%A7%E0%B8%99_%E0%B8%AB%E0%B8%A5%E0%B8%B5%E0%B8%81%E0%B8%A0%E0%B8%B1%E0%B8%A2
41	travel8	http://www.rd1677.com/branch.php?id=48957
42	travel9	http://www.thai-tour.com/wb/view_topic.php?id_topic=1553
43	political10	http://www.thaiedresearch.org/thaied_news/index1.php?id=12382
44	sport12	http://www.komchadluek.net/detail/20090316/5600/%E2%80%9C%E0%B8%8A%E0%B8%A5%E0%B8%9A%E0%B8%B8%E0%B8%A3%E0%B8%B5%E2%80%9D%E0%B8%99%E0%B8%B3%E0%B8%97%E0%B8%AD%E0%B8%87%E2%80%9C%E0%B8%81%E0%B8%B2%E0%B8%8D%E0%B8%88%E0%B8%99%E0%B8%9A%E0%B8%B8%E0%B8%A3%E0%B8%B5%E0%B9%80%E0%B8%81%E0%B8%A1%E0%B8%AA%E0%B8%AA%E0%B8%AA%E0%B8%AA

		%B9%8C.html
45	political11	http://www.ryt9.com/s/govh/430506/
46	travel7	http://www.ryt9.com/s/prg/144633/
47	sport14	http://www.geocities.com/patcharee_toy/toynews3.htm
48	travel5	http://thai.tourismthailand.org/news/release-content-2059.html
49	political6	http://www.siamturakij.com/home/news/print_news.php?news_id=413328535
50	political7	http://news.mcot.net/politic/inside.php?value=bmlkPTg1NTg4Jm50eXBlPXRleHQ=

APPENDIX B: Details of Thai text collection

Text id	Text name	The number of characters (*n* characters)	Text size (bytes)
1	sport2	721	721
2	sport1	805	805
3	sport4	874	874
4	sport3	913	913
5	travel4	1007	1007
6	sport7	1020	1020
7	sport9	1057	1057
8	travel3	1107	1107
9	sport15	1135	1135
10	travel11	1157	1157
11	sport6	1417	1417
12	travel13	1444	1444
13	sport5	1468	1468
14	education2	1507	1507
15	political1	1512	1512
16	education4	1544	1544
17	travel2	1561	1561
18	travel1	1563	1563
19	sport8	1643	1643
20	political3	1652	1652
21	travel15	1789	1789
22	political14	1879	1879
23	travel12	1901	1901
24	travel6	2002	2002
25	political13	2030	2030
26	education5	2035	2035
27	political9	2064	2064
28	political15	2107	2107
29	political5	2126	2126
30	education3	2142	2142
31	education1	2217	2217
32	travel14	2290	2290
33	sport13	2339	2339
34	sport10	2367	2367
35	sport11	2368	2368
36	travel10	2431	2431
37	political2	2437	2437
38	political8	2518	2518
39	political4	2735	2735
40	political12	2760	2760
41	travel8	2835	2835
42	travel9	2893	2893
43	political10	2922	2922

Text id	**Text name**	**The number of characters (*n* characters)**	**Text size (bytes)**
44	sport12	3020	3020
45	political11	3243	3243
46	travel7	3475	3475
47	sport14	3580	3580
48	travel5	3643	3643
49	political6	4010	4010
50	political7	4022	4022

APPENDIX C: Comparison of number of indexing terms extracted from five indexing techniques

Text id	Text name	The number of indexing terms				
		Suffix array	Word inverted index	3-gram inverted index	Vilo's technique	Frequent max substring technique
1	sport2	276864	29	132	1139	145
2	sport1	342930	36	146	1363	156
3	sport4	402477	27	124	998	194
4	sport3	438240	42	158	1022	204
5	travel4	530689	30	142	939	244
6	sport7	544170	32	161	3850	247
7	sport9	583464	36	202	3088	209
8	travel3	638739	43	167	1148	256
9	sport15	670785	57	250	2262	226
10	travel11	696514	33	172	1255	270
11	sport6	1037244	67	281	1838	329
12	travel13	1076502	47	224	1605	354
13	sport5	1112010	77	324	3047	312
14	education2	1170939	48	236	2229	353
15	political1	1178604	54	254	1997	335
16	education4	1228252	56	267	4457	319
17	travel2	1255044	53	242	1543	398
18	travel1	1258215	53	244	1548	409
19	sport8	1388335	73	302	2014	360
20	political3	1403374	67	325	3818	396
21	travel15	1642302	71	311	2628	395
22	political14	1809477	73	340	2332	430
23	travel12	1851574	78	332	2608	379
24	travel6	2051049	104	354	2545	465
25	political13	2108155	80	392	4475	384
26	education5	2118435	74	352	6013	409
27	political9	2178552	74	395	4456	383
28	political15	2269239	68	353	2073	505
29	political5	2309899	79	365	2543	475
30	education3	2344419	74	371	3674	481
31	education1	2509644	84	409	4582	507
32	travel14	2675865	80	344	2442	519
33	sport13	2790427	96	421	2687	537
34	sport10	2856969	86	393	4314	446
35	sport11	2859360	86	393	4314	446
36	travel10	3012009	89	435	3821	530
37	political2	3026754	93	434	3400	511
38	political8	3229335	84	409	4458	502
39	political4	3804385	98	466	4897	525
40	political12	3873660	105	500	4592	591
41	travel8	4085235	88	421	3783	473

Text id	**Text name**	**The number of indexing terms**				
		Suffix array	**Word inverted index**	**3-gram inverted index**	**Vilo's technique**	**Frequent max substring technique**
42	travel9	4252710	97	458	3668	654
43	political10	4337709	120	551	3562	695
44	sport12	4631170	144	582	3232	695
45	political11	5334735	127	613	5210	670
46	travel7	6119475	129	615	5071	703
47	sport14	6492330	109	156	8351	696
48	travel5	6721335	131	655	5122	775
49	political6	8134285	132	720	5177	894
50	political7	8182759	144	700	5711	818

APPENDIX D: Comparison of index sizes used by five indexing techniques

Text id	Text name	Index size (bytes)				
		Suffix array	Word inverted index	3-gram inverted index	Vilo's technique	Frequent max substring technique
1	sport2	261723	188	678	10705	783
2	sport1	326025	236	749	14098	961
3	sport4	384123	184	638	8305	999
4	sport3	419067	273	808	7727	1052
5	travel4	509542	219	735	7534	1344
6	sport7	522750	215	826	90172	1371
7	sport9	561267	281	1030	48990	1181
8	travel3	615492	272	864	9776	1411
9	sport15	646950	364	1272	22660	1336
10	travel11	672217	243	880	9937	1405
11	sport6	1007487	437	1429	14178	1825
12	travel13	1046178	337	1145	14629	1901
13	sport5	1081182	509	1643	36353	1755
14	education2	1139292	326	1207	29244	1829
15	political1	1146852	381	1292	20313	1897
16	education4	1195828	389	1357	102130	1716
17	travel2	1222263	352	1232	12523	1990
18	travel1	1225392	352	1242	12545	1993
19	sport8	1353832	452	1534	16394	1969
20	political3	1368682	471	1647	65888	2989
21	travel15	1604733	489	1585	27917	2222
22	political14	1770018	536	1724	19042	2484
23	travel12	1811653	527	1683	26557	2197
24	travel6	2009007	677	1803	22346	2630
25	political13	2065525	570	1986	61628	2529
26	education5	2075700	542	1790	140910	2575
27	political9	2135208	523	2020	51654	2707
28	political15	2224992	475	1788	16868	2697
29	political5	2265253	537	1854	24561	2691
30	education3	2299437	516	1884	51449	2834
31	education1	2463087	639	2072	58914	4381
32	travel14	2627775	521	1748	21429	3053
33	sport13	2741308	611	2134	22485	2837
34	sport10	2807262	558	2005	51060	2797
35	sport11	2809632	558	2005	51060	2797
36	travel10	2960958	601	2203	41547	3096
37	political2	2975577	658	2209	34813	3049
38	political8	3176457	593	2090	55627	3139
39	political4	3746950	657	2373	61902	3287
40	political12	3815700	738	2531	54046	3495
41	travel8	4025700	619	2140	39845	3073

Text id	Text name	Index size (bytes)				
		Suffix array	Word inverted index	3-gram inverted index	Vilo’s technique	Frequent max substring technique
42	travel9	4191957	653	2323	36276	4017
43	political10	4276347	812	2792	34906	3936
44	sport12	4567750	864	2936	24516	3703
45	political11	5266632	912	3105	57387	4107
46	travel7	6046500	892	3123	53360	4445
47	sport14	6417150	702	816	142595	4933
48	travel5	6644832	920	3318	51313	4721
49	political6	8050075	930	3651	48897	5458
50	political7	8098297	995	3572	58823	5105

APPENDIX E: Comparison of indexing times used by five indexing techniques

Text id	Text name	Indexing time (processing round)				
		Suffix array	Word inverted index	3-gram inverted index	Vilo's technique	Frequent max substring technique
1	sport2	519841	75168576	520562	15862	11436502
2	sport1	648025	83993700	648830	26565	21384825
3	sport4	763876	91253466	764750	15732	13749768
4	sport3	833569	95361024	834482	20086	18338518
5	travel4	1014049	105273794	1015056	23161	23323127
6	sport7	1040400	106646100	1041420	52020	53060400
7	sport9	1117249	110553744	1118306	31710	33517470
8	travel3	1225449	115838694	1226556	28782	31861674
9	sport15	1288225	118800450	1289360	31780	36070300
10	travel11	1338649	121128644	1339806	19669	22757033
11	sport6	2007889	148716984	2009306	22672	32126224
12	travel13	2085136	151589676	2086580	34656	50043264
13	sport5	2155024	154144404	2156492	51380	75425840
14	education2	2271049	158298294	2272556	22605	34065735
15	political1	2286144	158831064	2287656	49896	75442752
16	education4	2383936	162241976	2385480	44776	69134144
17	travel2	2436721	164054856	2438282	24976	38987536
18	travel1	2442969	164268174	2444532	25008	39087504
19	sport8	2699449	172807454	2701092	41075	67486225
20	political3	2729104	173768924	2730756	82600	136455200
21	travel15	3200521	188424636	3202310	64404	115218756
22	political14	3530641	198072906	3532520	41338	77674102
23	travel12	3613801	200433836	3615702	68436	130096836
24	travel6	4008004	211285074	4010006	50050	100200100
25	political13	4120900	214296950	4122930	101500	206045000
26	education5	4141225	214834950	4143260	168905	343721675
27	political9	4260096	217956336	4262160	74304	153363456
28	political15	4439449	222587694	4441556	35819	75470633
29	political5	4519876	224635286	4522002	55276	117516776
30	education3	4588164	226360134	4590306	115668	247760856
31	education1	4915089	234452184	4917306	64293	142537581
32	travel14	5244100	242339250	5246390	45800	104882000
33	sport13	5470921	247639286	5473260	35085	82063815
34	sport10	5602689	250670034	5605056	56808	134464536
35	sport11	5607424	250778304	5609792	56832	134578176
36	travel10	5909761	257603346	5912192	70499	171383069
37	political2	5938969	258253764	5941406	90169	219741853
38	political8	6340324	267041454	6342842	85612	215571016
39	political4	7480225	290648450	7482960	65640	179525400
40	political12	7617600	293374200	7620360	71760	198057600
41	travel8	8037225	301558950	8040060	99225	281302875

Text id	Text name	Indexing time (processing round)				
		Suffix array	Word inverted index	3-gram inverted index	Vilo's technique	Frequent max substring technique
42	travel9	8369449	307896204	8372342	69432	200866776
43	political10	8538084	311067354	8541006	99348	290294856
44	sport12	9120400	321796100	9123420	48320	145926400
45	political11	10517049	346281054	10520292	132963	431199009
46	travel7	12075625	371859750	12079100	121625	422646875
47	sport14	12816400	383471700	12819980	171840	615187200
48	travel5	13271449	390449454	13275092	102004	371600572
49	political6	16080100	431255450	16084110	100250	402002500
50	political7	16176484	432594254	16180506	128704	517647488

APPENDIX F: Comparison of number of indexing terms extracted from Vilo's method (contracted form 'Vilo') and the frequent max substring technique (contracted form 'FM') at given frequency threshold values between 2 and 10

Text id	Text name	Text Length (*n* characters)	The number of indexing terms					
			Threshold = 2		Threshold = 3		Threshold = 4	
			Vilo	FM	Vilo	FM	Vilo	FM
1	g_250	250	652	117	153	85	79	68
2	g_500	500	818	252	233	165	132	122
3	g_750	750	547	420	273	266	177	176
4	g_1000	1000	720	551	377	360	246	246
5	g_1250	1250	1405	626	757	418	388	289
6	g_1500	1500	1606	778	857	517	460	372
7	g_1750	1750	1663	1096	855	716	552	519
8	g_2000	2000	1620	1057	809	685	503	474
9	g_2250	2250	1843	1127	841	704	574	498
10	g_2500	2500	2195	1306	985	866	650	619
11	g_2750	2750	2342	1406	1095	917	708	644
12	g_3000	3000	2309	1631	1112	1043	761	741
13	g_3250	3250	2510	1695	1227	1079	806	766
14	g_3500	3500	4176	1833	1443	1207	888	842
15	g_3750	3750	5268	1908	1639	1261	978	904
16	g_4000	4000	4078	2097	1620	1353	1022	951
17	g_4250	4250	4293	2247	2317	1491	1534	1044
18	g_4500	4500	4777	2387	2491	1569	1645	1121
19	g_4750	4750	10461	2421	3173	1609	1690	1183
20	g_5000	5000	9116	2611	3161	1687	1726	1254

Text id	**Text name**	**Text Length (*n* characters)**	**The number of indexing terms**					
			Threshold = 5		**Threshold = 6**		**Threshold = 7**	
			Vilo	**FM**	**Vilo**	**FM**	**Vilo**	**FM**
1	g_250	250	56	54	39	39	34	34
2	g_500	500	105	102	84	84	62	62
3	g_750	750	125	125	105	105	92	92
4	g_1000	1000	181	181	145	145	119	119
5	g_1250	1250	266	233	212	196	161	160
6	g_1500	1500	311	282	250	236	203	203
7	g_1750	1750	399	390	316	312	263	261
8	g_2000	2000	388	377	303	300	256	254
9	g_2250	2250	417	375	338	308	285	261
10	g_2500	2500	496	485	384	376	319	314
11	g_2750	2750	531	498	428	402	358	342
12	g_3000	3000	562	551	458	452	371	365
13	g_3250	3250	599	586	482	472	399	395
14	g_3500	3500	663	646	526	518	431	429
15	g_3750	3750	693	664	558	538	455	450
16	g_4000	4000	756	728	601	578	502	485
17	g_4250	4250	1061	806	691	646	544	535
18	g_4500	4500	1137	864	758	704	606	593
19	g_4750	4750	1284	914	879	754	746	640
20	g_5000	5000	1306	956	884	769	754	653

Text id	Text name	Text Length (*n* characters)	The number of indexing terms					
			Threshold = 8		Threshold = 9		Threshold = 10	
			Vilo	FM	Vilo	FM	Vilo	FM
1	g_250	250	30	30	26	26	24	24
2	g_500	500	49	49	43	43	40	40
3	g_750	750	85	85	77	77	65	65
4	g_1000	1000	105	105	93	93	83	83
5	g_1250	1250	141	141	126	126	106	106
6	g_1500	1500	172	172	152	152	132	132
7	g_1750	1750	233	231	194	193	176	176
8	g_2000	2000	220	218	202	200	170	168
9	g_2250	2250	244	228	212	200	189	179
10	g_2500	2500	273	271	235	234	209	208
11	g_2750	2750	305	293	270	262	234	228
12	g_3000	3000	315	311	277	275	249	247
13	g_3250	3250	338	335	301	299	275	274
14	g_3500	3500	379	377	333	332	292	292
15	g_3750	3750	413	410	357	356	317	317
16	g_4000	4000	429	417	380	369	342	334
17	g_4250	4250	472	468	405	401	362	359
18	g_4500	4500	514	505	442	434	385	381
19	g_4750	4750	632	541	556	478	493	425
20	g_5000	5000	651	558	561	480	507	436

List of References

[1] C. Haruechaiyasak, C. Damrongrat, C. Sangkeettrakarn, S. Kongyoung, and N. Angkawattanawit, "Sansarn Look!: A Platform for Developing Thai-Language Information Retrieval Systems," in *21st International Technical Conference on Circuits/Systems, Computers and Communications*, Chiang Mai, Thailand, 2006.

[2] S. Haruechaiyasak, S.Kongyoung, and C. Damrongrat, "LearnLexTo: A Machine-Learning Based Word Segmentation for Indexing Thai Texts," in *ACM 17th Conference on Information and Knowledge Management,* 2008.

[3] W. Kanlayanawat and S. Prasitjutrakul, "Automatic Indexing for Thai Text with Unknown Words Using Trie Structure," in *Proceeding of NLP Pacific Rim Symposium*, 1997, vol. 14, no. 2, pp. 153-172, 2001.

[4] R. Sukhahuta and D. Smith, "Information Extraction Strategies for Thai Documents," *International Journal of Computer Processing of Oriental Languages (IJCPOL),* pp. 14(2):153-172, 2001.

[5] K. Khankasikam and N. Muansuwan, "Thai Word Segmentation a Lexical Semantic Approach," in the *10th Machine Translation Summit (MT Summit),* Phuket, Thailand, 2005, p. 331.

[6] P. Sojka and D. Antos, "Context Sensitive Pattern Based Segmentation: A Thai Challenge," in *Proceedings of EACL 2003 Workshop Computational Linguistics for South Asian Languages—Expanding Synergies with Europe*, Budapest, 2003, pp. 65-72.

[7] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval.* New York: ACM Press, 1999.

[8] C. Jaruskulchai, "Thai Text Segmentation: Problems and Potential Solutions," in the *Sixth Annual Workshop on Science and Technology Exchange between Thai Professionals in North America and Thailand,* Edmonton, Alberta, Canada, 1996.

[9] A. Kawtrakul, C. Thumkanon, Y. Poovorawan, P. Varasrai, and M. Suktarachan, "Automatic Thai Unknown Word Recognition," in *Proceeding of NLPRS'97*, 1997, pp. 341-348.

[10] J. Chuleerat, *"An Automatic Indexing for Thai Text Retrieval,"* PhD thesis, George Washington University, USA, 1998.

[11] T. Theeramunkong, V. Sornlertlamvanich, T. Tanhermhong, and W. Chinnan, "Character-Cluster Based Thai Information Retrieval," in *Proceedings of the Fifth International Workshop on Information Retrieval with Asian Languages* Hong Kong, 2000, pp. 75-80.

[12] C. Haruechaiyasak, S. Kongyoung, and M. N. Dailey, "A Comparative Study on Thai Word Segmentation Approaches," in *Proceedings of Electrical Engineering/Electronics, Computer, Telecommunications, and Information Technology*, 2008.

[13] Y. Poovorawan, "Dictionary-based Thai Syllable Segmentation (in Thai)," in *9th Electrical Engineering Conference,* Bangkok, 1986.

[14] V. Sornlertlamvanich, "Word Segmentation for Thai in Machine Translation System," Bangkok.

[15] Y. Thairatananond, "Towards the Design of a Thai Text Syllable Analyzer," in *Asian Institute of Technology*, 1981.

[16] S. Charnyapornpong, "A Thai Syllable Separation Algorithm," in *Asian Institute of Technology*, 1983.

[17] T. Theeramunkong and S. Usanavasin, "Non-Dictionary-Based Thai Word Segmentation using Decision Trees," in *Proceedings of the First International Conference on Human Language Technology Research*, 2001, pp. 1-5.

[18] S. Meknavin, P. Charoenpornsawat, and B. Kijsirikul, "Feature-based Thai Word Segmentation," in *Proceedings of the Natural Language Processing Pacific Rim Symposium (NLPRS'97)*, Phuket, Thailand 1997.

[19] C. Kruengkrai and H. Isahara, "A Conditional Random Field Framework for Thai Morphological Analysis," in *Proceeding of the Fifth International Conference on Language Resources and Evaluation (LREC-2006)*, 2006.

[20] J. Chuleerat, *Dictionary-Based Thai CLIR: An Experimental Survey of Thai CLIR*, vol. 2406/2002: Springer Berlin/Heidelberg, 2002.

[21] W. Cavnar and J. Trenkle, "N-Gram Based Text Categorization," in *Proceedings of the 3rd Annual Symposium on Document Analysis and Information Retrieval*, Las Vegas, 1994, pp. 161-175.

[22] P. Majumder, M. Mitra, and B. B. Chaudhuri, "N-Gram: A Language Independent Approach to IR and NLP," in *International Conference on Universal Knowledge*, 2002.

[23] J. D. Cohen, "Highlights: Language - and Domain-Independent Automatic Indexing Terms for Abstracting," *Journal of The American Society for Information Science*, vol. 46, no. 3, pp. 162-174, 1995.

[24] W. B. Cavnar, "Using an N-Gram-Based Document Representation with a Vector Processing Retrieval Model," in *D.K. Harman editor, Proceedings of the Third Text REtrieval (TREC-3)*, 1994, pp. 269-277.

[25] H. E. Williams and J. Zobel, "Indexing and Retrieval for Genomic Databases," in *IEEE Transaction on Knowledge and Data Engineering*, 2002, pp. 63-78.

[26] J. Mayfield and P. McNamee, "Single N-Gram Stemming," in *Proceedings of the 26th Annual International Conference on Information Retrieval, ACM SIGIR*, Toronto, Canada, 2003, pp. 415-416.

[27] A. Califano and I. Rigoutsos, "FLASH: A Fast Look-Up Algorithm for String Homology," in *Proceedings of the 1st International Conference on Intelligent Systems for Molecular Biology*, Bethesda, Maryland, 1993.

[28] M. Yamamoto and K. W. Church, "Using Suffix Arrays to Compute Term Frequency and Document Frequency for All Substrings in a Corpus," *Computational Linguistics*, vol. 27, pp. 1-30, 2001.

[29] C. Jaruskulchai and C. Kruengkrai, "A practical text summarizer by paragraph extraction for Thai," in *Proceedings of the Sixth International Workshop on Information Retrieval with Asian Languages*, Sappro, Japan, 2003, pp. 9-16.

[30] W. Aroonmanakun, "Collocation and Thai word segmentation," in *Proceedings of the 5th SNLP & 5th Oriental COCOSDA Workshop*, 2002, pp. 68-75.

[31] S. Luksaneeyanawin, "A Thai Text to Speech System," in *Proceedings of the Conference on Electronics and Computer Research and Development*, 1992.

[32] K. Asanee, T. Chalathip, and S. Sapon, "A Statistical Approach to Thai Word Filtering," in *Proceedings SNLP'95, the 2nd Symposium on Natural Language Processing*, Bangkok, Thailand, August 2-4, 1995, pp. 398-406.

[33] V. Ruttikorn, S. Waraporn, J. Somsak, and T. Sakchai, "An Analysis on Correct Sentence Selection by Word's General Usage Frequency," in *Natural Language Processing: Multi-lingual Machine Translation and Related Topics*, 1994, pp. 291-300.

[34] S. J. Puglisi, W. F. Smyth, and A.Turpin, "Inverted Files Versus Suffix Arrays for Locating Patterns in Primary Memory," in *SPIRE 2006*, pp. 122-133.

[35] J. H. Lee and J. S. Ahn, "Using n-Grams for Korean Text Retrieval," in *Proceedings of the19th Annual International Conference on Information Retrieval, ACM SIGIR*, Zurich, Switzerland, 1996, pp. 216-224.

[36] K. L. Kwok, "Comparing Representations in Chinese Information Retrieval," in *Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Philadelphia, USA, 1997, pp. 34-41.

[37] H. Fujii and W. B. Croft, "A Comparison of Indexing Techniques for Japanese Text Retrieval," in *Proceedings of ACM SIGIR 16th Annual International Conference on Research and Development in Information Retrieval*, 1993, pp. 237-246.

[38] P. McNamee, "Knowledge-Light Asian Language," in *Proceedings of the Third NTCIR Workshop on Research in Information Retrieval, Automatic Text Summarization and Question Answering Text Retrieval, at the NTCIR-3. Workshop*, 2002.

[39] M. S. Kim, K. Y. Whang, J. G. Lee, and M. J. Lee, "n-Gram/2L: A Space and Time Efficient Two-Level n-Gram Inverted Index Structure," in *VLDB, Trondheim*, Norway, 2005, pp. 325-336.

[40] M. M. Hasan and Y. Matsumoto, "Chinese-Japanese Cross Language Information Retrieval: A Han Character Based Approach," in *Proceedings of the SIGLEX Workshop on Word Senses and Multi-linguality, ACL-2000*, Hong Kong, 2000, pp. 19-26.

[41] P. Srichaivattana, "Dictionary-Less Search Engine for the Collaborative Database," in *Proceeding of the 3rd International Symposium on Communications and Information Technologies*, , September, 2003.

[42] C. Jaruskulchai., "Thai Text Segmentation: Problems and Potential Solutions," in *The Sixth Annual Workshop on Science and Technology Exchange between Thai Professionals in North America and Thailand*, Edmonton, Alberta, Canada, 1996.

[43] R. Pankhuenkhat, "Thai Linguistics": Chulalongkorn, 1998.

[44] A. Kawtrakul, C. Thumkanon, and P. McFetridge, "Automatic Multilevel Indexing for Thai Text Information Retrieval," in *IEEE Asia Pacific Conference on Circuits and Systems*, 1998.

[45] B. Liu, *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data*. New York: Springer-Verlag Berlin Heidelberg, 2007.

[46] R. L. Haskin, "Special-Purpose Processors for Text Retrieval," *Database Engineering*, vol. 4, no. 1, pp. 16-29, September 1981.

[47] D. Gusfield, *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge: Cambridge University Press, 1997.

[48] R. Sukhahuta and D. Smith, "Information Extraction Strategies for Thai Documents," *International Journal of Computer Processing of Oriental Languages (IJCPOL)*, vol. 14, no. 2, pp. 153-172, 2001.

[49] D. Lewis, "Naive (Bayes) at Forty: The Independence Assumption in Information Retrieval," in *Proceeding of 10th European Conference on Machine Learning*, 1998, pp. 4-15.

[50] J. R. Quinlan, "Induction of Decision Trees," in *Machine Learning*, vol. 1, no. 1, pp. 81-106, 1986.

[51] V. Vapnik, *The Nature of Statistical Learning Theory*. New York: Springer, 1995.

[52] T. Joachims, "Text Categorization with Support Vector Machines: Learning with Many Relevant Features," in *Proceedings of the 10th European Conference on Machine Learning*, 1998, pp. 137-142.

[53] F. Peng, F. Feng, and A. McCallum, "Chinese Segmentation and New Word Detection using Conditional Random Fields," in *Proceedings of the 20th International Conference on Computational Linguistics (COLING)*, 2004.

[54] J. Lafferty, A. McCallum, and F. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *Proceedings of the Eighteenth International Conference on Machine Learning (ICML)*, 2001, pp. 282-289.

[55] V. Sornlertlamvanich, T. Charoenporn, and H. Isahara, "ORCHID: Thai Part-Of-Speech Tagged Corpus," in *Technical Report TR-NECTEC-1997-001*: Thai National Electronics and Computer Technology Center, December 1997.

[56] T. Theeramunkong, W. Chinnan, T. Tanhermhong, and V. Sornlertlamvanich, "Full-Text Search for Thai Information Retrieval Systems," in *The Fourth Symposium on Natural Language Processing*, Thailand, 2000.

[57] E. Adams, "*A Study of Trigrams and Their Feasibility as Index Terms in a Full Text Information Retrieval System*." PhD thesis, George Washington University, USA, 1991.

[58] Y. Ogawa and T. Matsudua, "Optimizing query evaluation in n-gram indexing," in *Proceedings of International Conference on Information Retrieval, ACM SIGIR*, Melbourne, Australia, 1998, pp. 367-368.

[59] H. E. Williams, "Genomic Information Retrieval," in *Proceedings of the 14th Australasian Database Conferences*, 2003.

[60] L. F. Chien, "Fast and Quasi-Natural Language Search for Gigabytes of Chinese Texts," in *Proceedings of 18th ACM SIGIR Conference on Research and Development in Information Retrieval*, New York, USA, 1995, pp. 112-120.

[61] T. Liang, S. Y. Lee, and W. P. Yang, "Optimal Weight Assignment for a Chinese Signature File," in *Journal of Information Processing and Management*, vol. 32, no. 2, pp. 227-237.

[62] Y. T. Lin, *Chinese English Dictionary of Modern Usage*. Hong Kong: Chinese University of Hong Kong Press, 1972.

[63] H. Jiao, Q. Liu, and H.-b. Jia, "Chinese Keyword Extraction Based on N-Gram and Word Co-Occurrence," in *2007 International Conference on Computational Intelligence and Security Workshops (CISW 2007)* China, 2007.

[64] Y. Ogawa, A. Bessho, M. Iwasaki, M. Nishimura, and M. Hirose, "A New Indexing and Text Ranking Method for Japanese Text Databases Using Simple-Word Compounds as Keywords," in *Proceedings of the Third International Symposium on Database Systems for Advanced Applications*, 1993, pp. 197-204.

[65] E. Miller, D. Shen, J. Liu, and C. Nicholas, "Performance and Scalability of a Large-Scale N-Gram Based Information Retrieval System," *Journal of Digital Information,* vol. 1, no. 5, pp. 1-25, 2000.

[66] J. D. Cohen, "Recursive Hashing Functions for n-Grams," *ACM Transactions on Information Systems,* vol. 15, no. 3, pp. 291-320, July 1997.

[67] U. Manber and G. Myers, "Suffix Arrays: A New Method for On-Line String Searches," *in the First Annual ACM-SIAM Symposium on Discrete Algorithms,* 1990, pp. 319-327.

[68] D. B. Paul and J. M. Baker, "The Design for the Wall Street Journal-Based CSR Corpus," in *Proceedings of DARPA Speech and Natural Language Workshop,* 1992, pp. 357-361.

[69] C. T. Meadow, B. R. Boyce, D. H. Kraft, and C. Barry, *Text Information Retrieval Systems*, 3rd ed., London, UK: Academic Press, 2007.

[70] D. E. Knuth, J. H. Morris Jr, and V. R. Pratt, "Fast Pattern Matching in Strings," *SIAM Journal on Computing*, vol. 6, no. 2, pp. 323-350, 1977.

[71] R. S. Boyer and J. S. Moore, "A Fast String Searching Algorithm," in *Communications of the ACM (CACM)*, vol. 20, no. 10, pp. 762-772, October 1997.

[72] D. M. Sunday, "A Very Fast Substring Search Algorithm," in *Communications of the ACM (CACM)*, vol. 33, no. 8, pp. 132-142, August 1990.

[73] A. V. Aho and M. J. Corasick, "Fast Pattern Matching: An Aid to Bibliographic Search," in *Communications of the ACM (CACM)*, vol. 18, no. 6, pp. 333-340, June 1975.

[74] S. Wu and U. Manber, "Agrep—a fast approximate pattern searching tool," in *USENIX Conference*, January 1992.

[75] L. A. Hollaar, K. F. Smith, W. H. Chow, P. A. Emrath, and R. L. Haskin, "Architecture and Operation of a Large, Full-Text Information-Retrieval System," in *Advanced Database Machine Architecture, D.K. Hsiao, Ed.*, Englewood Cliffs, New Jersey: Prentice-Hall, 1992, pp. 256-299.

[76] J. Vilo, *"Discovering Frequent Patterns from Strings: Department of Computer Science, University of Helsinki, Finland,"* Technical Report C-1998-9, May 1998.

[77] H. P. Luhn, "The Automatic Creation of Literature Abstracts," *IBM Journal of Research*, vol. 2, no. 4, pp. 159-165, 1958.

[78] P. Weiner, "Linear Pattern Matching Algorithms," in *IEEE 14th Annual Symposium on Switching and Automata Theory*, 1973.

[79] R. Giegerich and S. Kurtz, "A Comparison of Imperative and Purely Functional Suffix Tree Constructions," *Science of Computer Programming*, vol. 25, no. 2-3, pp. 187-218, 1995.

[80] L. Colussi, "Fastest Pattern Matching in Strings," *Journal of Algorithms*, vol. 16, no. 2, pp. 163-189, 1994.

[81] R. Sinha, S. J. Puglisi, A. Moffat, and A. Turpin, "Improving Suffix Array Locality for Fast Pattern Matching on Disk," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, 2008, pp. 661-672.

[82] F. M. Liang, "Word Hy-phen-a-tion By Com-put-er." Doctor of Philosophy, Stanford University, USA, 1983.

[83] M. Nelson, "Fast String Searching with Suffix Trees," *Dr Dobb's Journal*, 1996.

[84] G. A. Stephen, *String Searching Algorithms*, vol. 6: World Scientific Press, 2000.

[85] R. Sinha, "Using Compact Tries for Cache-Efficient Sorting of Integers," in *Proceedings of International Workshop on Efficient and Experimental Algorithms*, 2001, pp. 513-528.

[86] M. Jung, M. Shishibori, Y. Tanaka, and J.-i. Aoe, "A Dynamic Construction Algorithm for the Compact Patricia Trie using the Hierarchical Structure,' *Journal of Information Processing and Management*, vol. 38, no. 2, March 2002.

[87] D. R. Morrison, "PATRICIA—Practical Algorithm to Retrieve Information Coded in Alphanumeric," *Journal of the ACM (JACM)*, vol. 15, no. 4, pp. 514-534, 1968.

[88] E. M. McCreight, "A Space-Economical Suffix Tree Construction Algorithm," *Journal of the ACM (JACM)*, vol. 23, no. 2, pp. 262-272, 1976.

[89] E. Ukkonen, "On-Line Construction of Suffix Trees," *Algorithmica,* vol. 14, no. 3, pp. 249-260, 1995.

[90] R. Giegerich and S. Kurtz, "From Ukkonen to McCreight and Weiner: A Unifying View of Linear-Time Suffix Tree Construction," *Algorithmica,* vol. 19 no. 3, pp. 331-353, 1997.

[91] R. Giegerich, S. Kurtz, and J. Stoye, "Efficient Implementation of Lazy Suffix Trees," in *Proceedings of the Third Workshop on Algorithmic Engineering (WAE99),* 1999, pp. 30-42.

[92] G. Navarro, "A Guided Tour to Approximate String Matching," *ACM Computing Surveys,* vol. 33, no. 1, pp. 31-88, Mar. 2001.

[93] J. Vilo, *"Pattern Discovery from Biosequences,"* in Facutly of Science, Department of Computer Science. PhD thesis, University of Helsinki, Helsinki, November 2002.

[94] T. Chumwatana, K. W. Wong, and H. Xie, "Using Frequent Max Substring Technique for Thai Keyword Extraction used in Thai Text Mining," in *2nd International Conference on Soft Computing, Intelligent System and Information Technology (ICSIIT 2010),* Bali, Indonesia, 1-2 July 2010.

[95] T. Chumwatana, K. W. Wong, and H. Xie "Frequent Max Substring Mining for Indexing," *International Journal of Computer Science and System Analysis (IJCSSA), India,* 2008.

[96] W. B. Croft, D. Metzler, and T. Strohman, *Search Engines: Information Retrieval in Practice.* Boston: Addison-Wesley, 2010.

[97] L. Narupiyakul, C. Thomas, N. Cercone, and B. Sirinaovakul, "Thai Syllable-Based Information Extraction Using Hidden Markov Models," in *the 5th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing-2004)* Seoul, Korea, February 15-21, 2004, pp. 537-546.

[98] C. Haruechaiyasak, P. Srichaivatanna, S. Kongyoung, and C. Damrongrat, "Automatic Thai Keyword Extraction from Categorized Text Corpus," 2008.

[99] T. Yamashita and Y. Matsumoto, "Language-Independent Morphological Analysis," in *the Sixth Applied Natural Language Processing Conference,* April 2000, pp. 232-238.

[100] P. Mittrapiyanuruk and V. Sornlertlamvanich, "The Automatic Thai Sentence Extraction," in *the Fourth Symposium on Natural Language Processing,* May 2000, pp. 23-28.

[101] N. Suwanno, Y. Suzuki, and H. Yamazaki, "Extracting Thai Compound Nouns for Paragraph Extraction in Thai Text," in *IEEE International Conference on Natural Language Processing and Knowledge Engineering (IEEE NLP-KE '05),* 2005.

[102] T. Charoenporn, V. Sornlertlamvanich, and H. Isahara, "Building a Large Thai Text Corpus—Part-Of-Speech Tagged Corpus: ORCHID" in *Proceedings of NLPRS'97,* 1997.

[103] C.-H. Lee and H.-C. Yang, "A Web Text Mining Approach Based on Self-Organizing Map," in *the 2nd International Workshop on Web Information and Data Management,* Kansas City, Missouri, United States, 1999, pp. 59-62.

[104] J. Bakus, M. F. Hussin, and M. Kamel, "A SOM-Based Document Clustering Using Phrases," in *Proceedings of the 9th International Conference on Neural Information Processing (ICONIP'02).* vol. 5, 2002, pp. 2212-2216.

[105] D. R. Cutting, D. R. Karger, J. O. Pedersen, and J. W. Tukey, "Scatter/Gather: A Cluster-Based Approach to Browsing Large Document Collections," in *Proceedings of the 15th Annual International ACM SIGIR conference on Research and Development in Information Retrieval*, 1992, pp. 318-329.

[106] I. Matveeva, "Document Representation and Multilevel Measures of Document Similarity," in *Proceedings of the 2006 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, New York, June 04-09, 2006, pp. 235-238.

[107] C.-M. Tseng, K.-H. Tsai, C.-C. Hsu, and H.-C. Chang, "On the Chinese Document Clustering Based on Dynamical Term Clustering," in *AIRS 2005*, pp. 534-539.

[108] Y. Li, S. M. Chung, and J. Holt, "Text Document Clustering Based on Frequent Word Meaning Sequences," *Data and Knowledge Engineering (DKE)*, vol. 64, no 1, pp. 381-404, 2008.

[109] Y. Wang and J. E. Hodges, "Document Clustering using Compound Words," in *Proceedings of the 2005 International Conference on Artificial Intelligence (ICAI 2005)*, Las Vegas, Nevada, USA, 2005, pp. 307-313.

[110] J. Mathieu, "Adaptation of a Keyphrase Extractor for Japanese Text," in *Proceedings of the 27th Annual Conference of the Canadian Association for Information Science (CAIS-99)* Sherbrooke, Quebec, 1999, pp. 182-189.

[111] T. Chumwatana, K. W. Wong, and H. Xie, "An Automatic Indexing Technique for Thai Texts using Frequent Max Substring," in *Eighth International Symposium on Natural Language Processing, 2009 (SNLP '09)* Bangkok, Thailand, 2009.

[112] R. Feldman and J. Sanger, *The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data.* Cambridge University Press, 2006.

[113] R. C. Dubes and A. K. Jain, *Algorithms for Clustering Data*, Prentice Hall, 1988.

[114] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis.* John Wiley and Sons, 1990.

[115] M. Steinbach, G. Karypis, and V. Kumar, "A Comparison of Document Clustering Techniques," in *KDD Workshop on Text Mining*, 2000.

[116] Y. Zhao and G. Karypis, "Comparison of Agglomerative and Partitional Document Clustering Algorithms," in *The SIAM Workshop on Clustering High-Dimensional Data and Its Applications*, Washington, DC, April 2002.

[117] Z. Huang, "Extensions to the K-means Algorithm for Clustering Large Datasets with Categorical Values," in *Data Mining and Knowledge Discovery*, 1998, pp. 283-304.

[118] D. Dembele and P. Kastner, "Fuzzy C-means Method for Clustering Microarray Data," *Bioinformatics*, vol. 19, no. 8, pp. 973-980, 2003.

[119] L. J. Heyer, S. Kruglyak, and S. Yooseph, "Exploring Expression Data: Identification and Analysis of Coexpressed Genes," *Genome Research*, vol. 9, no. 11, pp. 1106-1115, 1999.

[120] C. C. Fung, K. W. Wong, H. Eren, R. Charlebois, and H. Crocker, "Modular Artificial Neural Network for Prediction of Petrophysical Properties from Well Log Data," *IEEE Transactions on Instrumentation & Measurement*, vol. 46, no. 6, pp. 1259-1263, December 1997.

[121] D. Myers, K. W. Wong, and C. C. Fung, "Self-Organising Maps Use for Intelligent Data Analysis," *Australian Journal of Intelligent Information Processing Systems*, vol. 6, no. 2, pp. 89-96, 2000.

[122] A. Blazejewski and R. Coggins, "Application of Self-Organizing Maps to Clustering of High-Frequency Financial Data," in *The Second Workshop on Australasian Information Security, Data Mining and Web Intelligence, and Software Internationalisation*, Dunedin, New Zealand, 2004.

[123] D. R. Hill, "A Vector Clustering Technique,' in *Mechanized Information Storage, Retrieval and Dissemination,* K. Samuelson, Ed. North-Holland, Amsterdam, 1968.

[124] J. J. Rocchio, *"Document Retrieval Systems—Optimization and Evaluation,"* PhD thesis, Harvard University, USA, 1966.

[125] G. Salton, A. Wong, and C. S. Yang, "A Vector Space Model for Automatic Indexing," *Communications of the ACM,* vol. 18, no. 11, pp. 613-620, 1975.

[126] O. Zamir, "*Clustering Web Documents: A Phrase-Based Method for Group Search Engine Results*," PhD thesis, University of Washington, USA, 1999.

[127] D. Mladenic and M. Grobelnik, "Word Sequence as Features in Text-Learning," in *Proceedings of the 17th Electrotechnical and Computer Science Conference (ERK-98)* Ljubljana, Slovenia, 1998.

[128] C. Kruengkrai and C. Jaruskulchai, "Thai Text Document Clustering Using Parallel Spherical K-means Algorithm on PIRUN Linux Cluster (in Thai)" in *The Fifth National Computer Science and Engineering Conference*, Chiang Mai University, Chiang Mai, Thailand, 2001, pp. 7-9 November.

[129] T. Kohonen, *Self-Organization and Associative Memory,* vol. 8. New York: Springer, 1984.

[130] K. Lairungruang, *"Automatic Thesaurus Construction with Term Context and Syntactic Analysis for Thai Text Retrieval,"* Master thesis, Mahidol University, Bangkok, Thailand, 2003.

[131] F. Archetti, P. Campanelli, E. Fersini, and E. Messina, "A Hierarchical Document Clustering Environment Based on the Induced Bisecting k-Means." vol. 4027 of Lecture Notes in Computer Science, H. L. Larsen, G. Pasi, D. O.Arroyo, T. Andreasen, and H. Christiansen, Eds.: Springer, 2006, pp. 257-269.

[132] K. Wang, M. Ester, and B. C. M. Fung, "Hierarchical Document Clustering using Frequent Itemsets," in *SIAM International Conference on Data Mining,* San Francisco, CA, United States, 2003, pp. 59-70.

[133] A. K. Jain and R. C. Dubes, *Algorithm for Clustering Data*. Englewood Cliffs, NJ: Prentice Hall, 1988.

[134] C. Thumkanon, A. Kawtrakul, Y. Poovorawan, F. Andres, K. Nakasiri, W. Manomaiphibul et al., "Research Resource Management for Thai Language Processing Service," in *International Conference on Computer Communication 1999 (ICCC'99)* Tokyo, 1999.

[135] L. Wieger, *Chinese Characters: Their Origin, Etymology, History, Classification and Signification.* The Catholic Mission Press, 1965.

[136] T. Chumwatana, K. W. Wong, and H. Xie, "Non-Segmented Document Clustering Using Self-Organizing Map and Frequent Max Substring Technique," in *16th International Conference on Neural Information Processing (ICONIP 2009),* Bangkok, Thailand, 2009.

[137] B. Phoophakdee and M. J. Zaki, "TRELLIS+: An Effective Approach for Indexing Genome-Scale Sequences Using Suffix Trees,' in *Pacific Symposium on Biocomputing 2008,* Kohala Coast, Hawaii, USA, 2008.

[138] H. Huo and V. Stojkovic, "A Suffix Tree Construction Algorithm for DNA Sequences," in *the Proceedings of the 7th IEEE International Conference on Bioinformatics and Bioengineering (BIBE 2007),* Boston, 2007, pp. 1178-1182.

[139] K. Sadakane and T. Shibuya, "Indexing Huge Genome Sequences for Solving Various Problems," in *Genome Informatics,* vol. 12, pp. 175-183, 2001.

[140] P. Baldi and G. W. Hatfield, *DNA Microarrays and Gene Expression: From Experiments to Data Analysis and Modeling*. Cambridge, United Kingdom: Cambridge University Press, 2002.

[141] M. Barsky, U. Stege, A. Thomo, and C. Upton, "Suffix Trees for Very Large Genomic Sequences," in *CIKM '09: Proceedings of the 18th ACM Conference on Information and Knowledge Management*, 2009.

[142] B. Phoophakdee and M. J. Zaki, "Genome-Scale Disk-Based Suffix Tree Indexing," in *Proceedings of the ACM SIGMOD International Conference on Management of Data. ACM*, 2007, pp. *833--844*.

[143] H. E. Williams and. J. Zobel, "Indexing and Retrieval for Genomic Databases," in *IEEE Transactions on Knowledge and Data Engineering*, 2002, pp. 63-78.

[144] H. E. Williams, *"Indexing and Retrieval for Genomic Databases,"* PhD thesis, RMIT University, Melbourne, Australia, 1998.

[145] M. Barsky, U. Stege, A. Thomo, and C. Upton, "A New Method for Indexing Genomes Using On-Disk Suffix Trees," in *CIKM '08: Proceedings of the 17th ACM Conference on Information and Knowledge Management*, 2008, pp. *649-658*.