

**UNIVERSITY OF TURKISH AERONAUTICAL ASSOCIATION  
INSTITUTE OF SCIENCE AND TECHNOLOGY**

**IMPROVING POSTING LISTS INTERSECTION WITH SKIP POINTERS**



**MASTER THESIS**

**Haitham Wajdi Hussein AL-OBAIDI**

**THE DEPARTMENT OF INFORMATION TECHNOLOGY**

**THE PROGRAM OF INFORMATION TECHNOLOGY**

**DECEMBER, 2017**

**UNIVERSITY OF TURKISH AERONAUTICAL ASSOCIATION INSTITUTE  
OF SCIENCE AND TECHNOLOGY**

**IMPROVING POSTING LISTS INTERSECTION WITH SKIP POINTERS**

**MASTER THESIS**

**Haitham Wajdi Hussein AL-OBAIDI**

**1406050032**


**THE DEPARTMENT OF INFORMATION TECHNOLOGY**

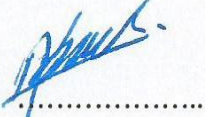
**THE PROGRAM OF INFORMATION TECHNOLOGY**

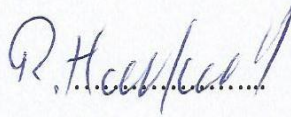
**Thesis Supervisor: Assist. Prof. Dr. Shadi AL-SHEHABI**

**Thesis Co-Supervisor: Assist. Prof. Dr. Abdül kadir GORUR**

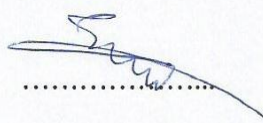
Türk Hava Kurumu Üniversitesi Fen Bilimleri Enstitüsü'nün 1406050032 numaralı Yüksek Lisans öğrencisi “Haithem Wajdi Hussein AL-OBAIDI” ilgili yönetmeliklerin belirlediğigerekli tüm şartları yerine getirdikten sonra hazırladığı “Improving Posting Lists Intersection with Skip Pointers” başlıklı tezini aşağıda imzaları bulunan jüri önünde başarı ile sunmuştur.

**Supervisor** Assist. Prof. Dr. Shadi AL SHEHABI   
Türk Hava Kurumu Üniversitesi .....

**Co-Supervisor** Assist. Prof. Dr. Abdul kadir GORUR   
Çankaya Üniversitesi .....

**Jury Members** Assoc. Prof. Dr. Reza HASSANPOUR   
Çankaya Üniversitesi .....

Assist. Prof. Dr. Hakan Ezgi KIZILÖZ   
Türk Hava Kurumu Üniversitesi .....

Assist. Prof. Dr. Shadi AL SHEHABI   
Türk Hava Kurumu Üniversitesi .....

## **STATEMENT OF NON-PLAGIARISM PAGE**

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.



**Haitham Wajdi Hussein AL-OBAIDI**

**DECEMBER, 2017**

## **ACKNOWLEDGEMENTS**

I would like to express my sincere gratitude to Assist. Prof. Dr. Shadi AL-SHEHABI and Assist. Prof. Dr. Abdül Kadir GORUR for all the guidance, knowledge and wisdom they provided at different stages of my educational journey. I would also express my appreciation to the candle that burnt to light our ways, my father. My gratitude also goes to my mother, who I learnt my first words from her. I would also thank all my family especially my wife for all the support they provided along the way. Without you all, I wouldn't be here today. Thank you.

## TABLE OF CONTENTS

ACKNOWLEDGEMENTS .....	v
TABLE OF CONTENTS .....	vi
LIST OF FIGURES.....	viii
LIST OF TABLES .....	ix
LIST OF ABBREVIATIONS .....	x
ABSTRACT .....	xi
ÖZET.....	xiii
Chapter One .....	1
Introduction .....	1
1.1. Data Indexing .....	1
1.2. Postings Lists.....	2
1.3. Skip Pointers.....	4
1.4. Problem Definition .....	5
1.5. Aim of the Study .....	6
Chapter Two.....	7
Literature Review .....	7
Chapter Three.....	13
Methodology .....	13
3.1. Existing Skip Methods .....	13
3.2. The Proposed Method: Dynamic Skip Pointers .....	18
Chapter Four.....	21
Experimental results.....	21

4.1. Experiment A .....	21
4.2. Experiment B.....	23
4.3. Experiment C.....	26
4.4. Experiment D .....	28
4.5. Experiment E.....	30
Chapter Five .....	33
Discussion .....	33
Chapter Six.....	36
Conclusion.....	36
References.....	38

## LIST OF FIGURES

Figure 1.1: Sample posting list.....	2
Figure 1.2: Sample inverted indexing dictionary.....	3
Figure 1.3: Postings lists for two words.....	5
Figure 2.1: Sample of Inverted indexing with dictionary and postings. ....	9
Figure 2.2: Sample postings list with skip pointers. ....	10
Figure 2.3: Sample hierarchy of a skip list. ....	10
Figure 3.1: Sample postings list with skip pointers. ....	15
Figure 4.1: Average execution time for the skip methods using stop words. ....	23
Figure 4.2: Average number of comparisons made by the skip methods using stop words. ....	23
Figure 4.3: Average execution time for the skip methods using frequent words. ....	25
Figure 4.4: Average number of comparisons by the skip methods using frequent words. ....	25
Figure 4.5: Average execution time for the skip methods using rare words. ....	27
Figure 4.6: Average number of comparisons made by the skip methods using rare words. ....	28
Figure 4.7: Average execution time for the skip methods using stop-frequent words combination.....	29
Figure 4.8: Average number of comparisons for the stop-frequent words combination.	30
Figure 4.9: Average execution time for the skip methods using stop-rare words combination.....	32
Figure 4.10: Average number of comparisons for the stop-rare words combination. ....	32



## LIST OF TABLES

Table 4.1: Time and comparisons count of the three skip methods to find the keywords 'in' & 'was'.....	22
Table 4.2: Time and comparisons count of the three skip methods to find the keywords 'the' & 'of'.....	22
Table 4.3: Average performance of the skip methods using stop words. ....	22
Table 4.4: Time and comparisons of the skip methods to find the words 'advantage' & 'meeting'.....	24
Table 4.5: Time and comparisons of the skip methods to find the words 'distance' & 'pass'.....	24
Table 4.6: Average performance of the skip methods using frequent words.....	24
Table 4.7: Time and comparisons of the skip methods to find the words 'huddle' & 'people'.....	26
Table 4.8: Time and comparisons of the skip methods to find the words 'moment' & 'uncle'.....	26
Table 4.9: Average performance of the skip methods using rare words.....	27
Table 4.10: Time and comparisons of the skip methods to find the words 'the' & 'associated'.....	28
Table 4.11: Time and comparisons of the skip methods to find the words 'in' & 'meeting'.....	29
Table 4.12: Average performance of the skip methods using a stop-frequent words combination.....	29
Table 4.13: Time and comparisons of the skip methods to find the words 'be' & 'continent'.....	30
Table 4.14: Time and comparisons of the skip methods to find the words 'it' & 'grins'. 31	
Table 4.15: Average performance of the skip methods using a stop-rare words combination.....	31

## LIST OF ABBREVIATIONS

WWW	World Wide Web
IR	Information Retrieval
SP	Skip Pointer
SL	Skip List
DS	Dynamic Skip



## **ABSTRACT**

### **Improving Posting Lists Intersection with Skip Pointers**

AL-OBAIDI, Haitham Wajdi Hussein

Master, Department of Information Technology

Supervisor: Assist. Prof. Dr. Shadi AL-SHEHABI

Co-Supervisor: Assist. Prof. Dr. Abdül Kadir GORUR

December 2017, 54 pages

The rapid accumulative growth in the number of digital documents imposes many challenges to the digital world. One of the main challenges created by this growth is to find documents related to a search query in a reasonable execution time. To reduce the time required to search for a certain word in all documents, inverted indexing is used, where the indexing is based on words instead of documents, so that, each word is indexed with a list of documents identification numbers that contain this word. Each document number is known as postings and these postings are stored in an ordered manner in lists known as postings lists.

When a search query is executed, the postings lists related to these words are retrieved in order to find documents that exist in all postings lists, as results of the search query, by finding intersections between related postings lists. As postings lists are stored in an ordered manner, certain positions are skipped during search using skip pointers, to accelerate the intersection process, by comparing the required value to a value in a remote position in order to decide the next step taken by the intersecting

algorithm. The distribution of these skip pointers over the postings lists is a key factor to accelerate the intersecting process and is different from one method to another. The performance of the existing methods needs to be improved in order to process longer postings lists while maintaining reasonable execution time.

In this study, a novel method based on skip pointer is proposed to provide faster results for the intersecting process, in order to meet up the growing number of documents. The proposed method is tested using different scenarios, and compared to the existing methods. The results of the conducted experiments show significant improvement in the number of comparisons, hence execution time, required to find intersections between postings lists of different sizes.

**Keywords:** Information retrieval; Skip pointers; Skip lists; Inverted indexing.

## ÖZET

### **Atlama İşaretçileriyle Kesişen Mesaj Gönderi Listelerini İyileştirme**

AL-OBAIDI, Haitham Wajdi Hussein

Yüksek Lisans, Bilgi Teknolojileri Bölümü

Tez Danışmanı: Yrd.Doç.Dr. Shadi AL-SHEHABI

Eş Danışman: Yrd.Doç.Dr. Abdül Kadir GORUR

Aralık 2017, 54 sayfa

Dijital belge sayısındaki hızlı artış dijital dünyayı birçok zorlukla karşı karşıya bırakmaktadır. Bu artışın yarattığı ana zorluklardan birisi makul bir süre içerisinde bir arama sorgusuyla ilgili belgeleri bulmaktır. Tüm belgelerde belirli bir kelimeyi bulmak için gereken süreyi kısaltma amacıyla ters indeksleme kullanılmaktadır. Burada indeksleme belgeler yerine sözcüklere dayalıdır, bu nedenle her kelime bu kelimeyi içeren bir belge kimlik numarası listesi ile indekslenir. Her belge numarası gönderi olarak adlandırılmakta ve bu gönderiler gönderi listeleri olarak bilinen listelerde sıralı bir şekilde saklanmaktadır.

Bir arama sorgusu yürütüldüğünde, arama sorgusunun sonucu olarak bütün gönderi listelerinde mevcut olan belgeleri bulmak için ilgili gönderi listeleri arasında kesişimler bulunarak bu kelimeler ile ilgili gönderi listeleri getirilir. Gönderi listeleri sıralı bir şekilde saklandığı için, kesişme işleminin yapıldığı algoritma tarafından atılacak bir sonraki adıma karar vermek ve kesişim sürecini hızlandırmak için gerekli değeri uzak bir konumdaki değer ile karşılaştırarak arama sırasında atlama işaretçileri kullanılarak belirli pozisyonlar atlanmaktadır. Bu atlama işaretçilerinin gönderi

listelerine dağılımı kesiştirme işlemini hızlandırmak için önemli bir faktördür ve bir yöntemden diğere farklılık arz etmektedir. Bir yandan makul yürütme süresi korunurken diğeryandan daha uzun gönderi listelerinin işlenebilmesi için mevcut yöntemlerin uygulamalarının geliştirilmesi gerekmektedir.

Bu çalışmada, gittikçe artan doküman sayısını karşılamak için, kesiştirme süreci için daha hızlı sonuç elde etmek amacıyla atlama işaretçilerine dayanan yeni bir yöntem önerilmiştir. Önerilen yöntem, farklı senaryolar kullanılarak test edilmiş ve mevcut yöntemlerle karşılaştırılmış. Yürütülen deneylerin sonuçları, farklı boyutlardaki gönderi listeleri arasındaki kesişimleri bulmak için gereken karşılaştırma sayısında, dolayısıyla yürütme süresinde önemli oranda bir gelişme göstermiştir.

**Anahtar Kelimeler:** Bilgi alımı ; Atlama işaretçileri; Atlama listeleri; Ters indeksleme.

## CHAPTER ONE

### INTRODUCTION

The skip pointers technique is widely used in different applications that require finding intersections in ordered lists, according to the acceleration it provides to such processes [1-3]. This technique provides shortcuts for the search process, so that, it is possible to compare ranges in order to figure out whether the required value may exist in that range of positions or not. Many other techniques are built based on the use of skip pointers, such as skip lists and other more complex methods that use skip pointer in multiple ways to accelerate the process of finding intersections [4-6].

#### **1.1. Data Indexing**

Data indexing aims to provide easier and faster way to retrieve information from indexed documents. For example, it takes plenty of time to search all the documents for a certain word, every time a search query is requested. Such process becomes even more time-consuming, when the number of documents is increased. With the rapidly growing number of online digital documents, it has become impractical to use the traditional techniques to search for a certain word in all these documents, every time a search phrase is queried. Thus, many studies have proposed different techniques to provide alternative techniques rather than scanning the documents per each search query.

One of the widely used techniques for faster retrieval of information, is the inverted indexing, where indexing is based on retrieved information rather than documents. In this technique, words are extracted from the indexed document, each word alongside with all the documents that this word appears in [7-9]. Using this method, it becomes easier to query the document that has a certain word, by simply retrieving the indexed data related to the word. Each document identification number, where the word appears in that document, is known as posting. The list of documents identification numbers for

an indexed word is known as postings list. While the table that includes the indexed words and the postings lists is known as the dictionary.

## 1.2. Postings Lists

Every indexed document is assigned with an identification number. These numbers are assigned in an ordered way, depending on the indexing sequence of these documents. So that, newer documents have larger identification number than the older ones. Thus, appending new documents to a list maintains the order of the documents identification numbers in that list. This technique is used to create the postings lists during the inverted indexing process.

Documents are scanned periodically to update postings lists, related to each word in the dictionary, which are created during inverted indexing. Documents that no longer exist are removed from all postings lists, while modified documents are scanned for changes, so that these documents are removed from the postings lists of words that no longer exists in the document, and postings are added to the postings lists related to words added to the document in the appropriate position depending on the document identification number. A sample postings list for the word “World” is shown in Figure 1.1. The frequency of this word in the scanned documents is shown next to the indexed word, then documents identification numbers from the dataset, which are shown inside the cells, are distributed in the 198 positions, which are shown under the cells that contain the document identification numbers.

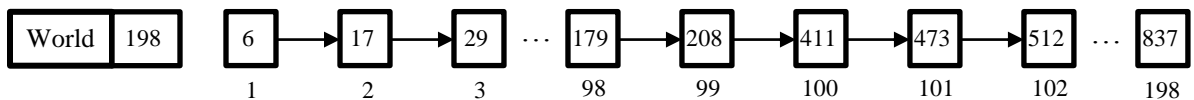


Figure 1.1: Sample posting list.

When a search is being queried for documents related to specific words, the results are concluded using the postings lists, without the need to go through the scan or indexing process. On the other hand, maintaining the order of the identification numbers in the postings lists assists accelerating the search process by using skip pointers. These



pointers accelerate the discovery of intersections among postings lists in order to accelerate the search process. These pointers have no role in the inverted indexing process, they are only placed on the postings lists resulted from inverted indexing, after the inverted indexing is completed. A sample inverted indexing dictionary, for multiple words, is shown in Figure 1.2.

term	doc. freq.	→	postings lists
ambitious	1	→	2
be	1	→	2
brutus	2	→	1 → 2
capitol	1	→	1
caesar	2	→	1 → 2
did	1	→	1
enact	1	→	1
hath	1	→	2
I	1	→	1
i'	1	→	1
it	1	→	2
julius	1	→	1
killed	1	→	1
let	1	→	2
me	1	→	1
noble	1	→	2
so	1	→	2
the	2	→	1 → 2
told	1	→	2
you	1	→	2
was	2	→	1 → 2
with	1	→	2

Figure 1.2: Sample inverted indexing dictionary.

### 1.3. Skip Pointers

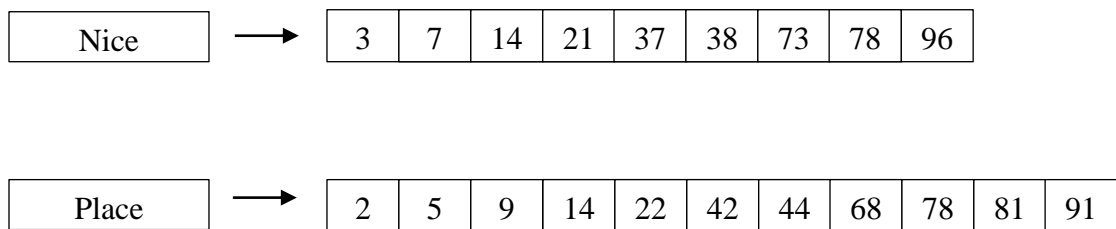
In a certain ascending list, if a number in a specific position is less than a certain number, then all numbers before that position are less than that certain number too. Based on this idea, skip pointers techniques rely on comparing the queried values to values in a certain position, so that, if the values in these positions are less than the queried values, all values prior to this position are neglected, based on the previous idea. The use of such technique helps to eliminate larger number of values using fewer comparisons.

The classic skip pointer technique places skip pointers, one skip pointer for every group of locations in the ordered list. The number of positions per each group is equal to the square root of the total number of items in that list. When a certain value is queried, the method goes through the list by checking the value in the position of the next skip pointer, before deciding the path it goes through, for the search process. This process is repeated every time a position with a skip pointer is reached. In case that the value in the position, where the next skip pointer is located, is less than the queries values, then the entire elements prior to this position are eliminated from the search process.

In order to find documents that contain more than one word, postings list for each word is retrieved in order to find documents that exist in all postings lists. This search is achieved by finding the intersections between two postings lists. The list that results from this intersection is then used to find intersections with other lists, in case that the search query includes more than two words. The search for intersections is continued until the last postings list is processed when the results of this intersection are the results of the search queried. These results include the identification numbers of all the documents that include all the words queried in the search.

Skip pointers based methods have the ability to eliminate multiple postings, which cannot be in the intersection results, from both lists by iterating through these lists removing all documents in one list with identification number less than the last searched number from the other list. This procedure cannot be achieved using other search methods, such as binary search, where the search for intersections must iterate through

one list and search for documents from that list in the other list, one by one. Thus, skip pointers based methods have the ability to find these intersections using fewer comparisons, hence, less execution time. For example, to search for intersection in the postings lists shown in Figure 1.3 using binary search method, the first posting from one of the postings lists is searched for in the other one. So that, the first posting for the word “Nice”, which is 3 is searched for in the second postings list. For the word “Place”, by dividing the list into two parts, then the value of the posting in the middle, which is 42 is compared to the required value. As the value 42 is larger than the required value 3, the right half of the list is neglected, and the remaining set is divided by two. The value in the middle of the remaining list, which is 9, is compared to the required value 3. As the 9 is larger than the 3, then the right half is also neglected, keeping only two values remain. Both values are not equal to the required value, thus, this value is not included in the intersection results. This process is repeated for every posting from one of the lists, until all values from that list is searched for in the other one. Such technique does not have the ability to eliminate multiple values from the first list, while the skip pointers methods do that, as the following chapters explain.



**Figure 1.3: Postings lists for two words.**

#### 1.4. Problem Definition

The rapid increase in the number of documents available on the internet each day is creating problems about finding documents related to a certain search query. Many methods are proposed to overcome the problem of retrieving information from a single document, so that, the process of finding intersections among the retrieved data, has become the bottleneck of searching these documents. Skip pointers is a widely used method, in different techniques, to find intersections between lists, using less number of

comparisons, hence, less time consumption. The huge numbers of documents impose challenges to the existing methods, where finding intersections for huge lists within a reasonable interval of time require supercomputers.

### **1.5. Aim of the Study**

The aim of this study is to propose a new algorithm that is capable of processing lists with an enormous number of entries in them. The proposed method is based on the skip lists and is capable of finding all the intersection using fewer comparisons and therefore, less execution time.

The remainder of this study is organized as follows: chapter two reviews literature related to the subject, which this study is investigating. Chapter three explains the existing methods, the way they process lists to find intersections, as well as the proposed method and the algorithm it uses to find intersections. Chapter four demonstrated the experiments conducted to measure and compare the performance of the proposed method with the existing method. Chapter five discusses the results of the experiments and the difference among these methods. Chapter six illustrates the conclusions acquired from this study.

## **CHAPTER TWO**

### **LITERATURE REVIEW**

The rapid growth in the number of documents on the World Wide Web makes it difficult to use the traditional techniques used earlier, when only limited number of documents are used to exist or shared in a certain physical or logical location [10, 11]. Information retrieval is one of the services that are suffering from this growth, where existing techniques are very time consuming when used to process the existing huge number of documents that contain the required information. Thus, many improvements are proposed for the information retrieval techniques in order to process the increasing number of documents and retrieve the required information within acceptable processing time [12].

Information Retrieval (IR) is the act of search, representation and manipulation of huge human-language data collections, such as electronic text. Many methods are proposed to accelerate the process of information retrieval from documents, in order to maintain the processing time within the acceptable limits despite the huge increasing number of documents online [13, 14]. Information retrieval techniques may be used to retrieve information from a document, or retrieve documents that have information related to a certain query.

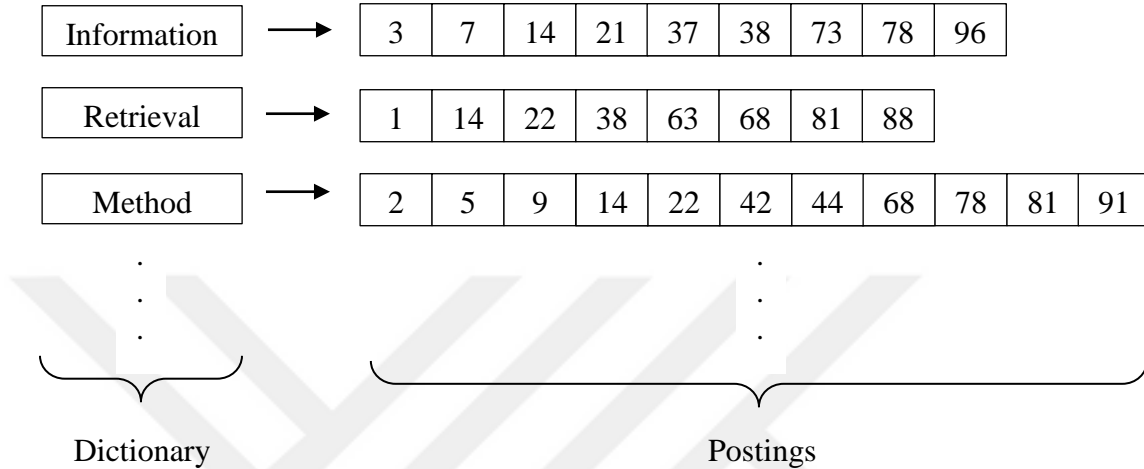
Most of the proposed methods focus on improving the process of retrieving data from every single document. For example, inverted indexing is targeted toward retrieving information about documents related to an ad hoc query by indexing information the opposite way the data is stored in. Each word is indexed alongside with all documents that this word appears in, and the number of time that it appears in that document. Thus, when a query that contains an indexed word, information about related documents are retrieved directly from inverse indexing tables, rather than searching all the document during every query processing [15].

Some other methods rely on estimating the language model used in each document, so that, when an ad hoc query is executed, the likelihood of a document related to this search is estimated depending on the concluded language model. This also allows faster information retrieval from the documents themselves. The method proposed in [16] uses the Bayesian decision theory to estimate a probabilistic ranking for each query to be in a certain document. The ranking is computed using model languages created for both the document and the search query being searched in the documents.

Information may be retrieved from one or more documents for each query. Then, the documents that have information related to that query are retrieved as search results for that query. Most search queries contain more than one word, which may or may not be sequential in the documents. Thus, it is not possible to search for that query in the document as one phrase. It is mandatory to search for each word a time in order to measure the relevance of each document to the search query. To do so, each word is searched in the available electronic documents, then, the intersection of related documents is retrieved as the most related results to the search query. These results may also be ranked according to the position of the words in the query and the frequency of each word's repetition [17-19].

A posting list is a list that contains the identification numbers of documents. Inverted indexes are dictionaries for the terms exist in a posting list, alongside with all postings that have this term in them. Information retrieval from a certain document is related to the size of that document, in other words, the more the data presented in the document, the more time required extracting data from that document. As demonstrated earlier, many methods are proposed to accelerate this process. On the other hand, refining the results by finding the intersection of documents that contain more words from the search query is related to the number of documents found to be related to that search query. For example, a two-word search query is executed and two lists are generated, where each list contains the documents that have information related to each word in the search query. Then, these results must be refined in order to display the documents that include both words in order to display these documents only, or give priority to these documents in an arranged search results, where most relevant results are displayed first. The

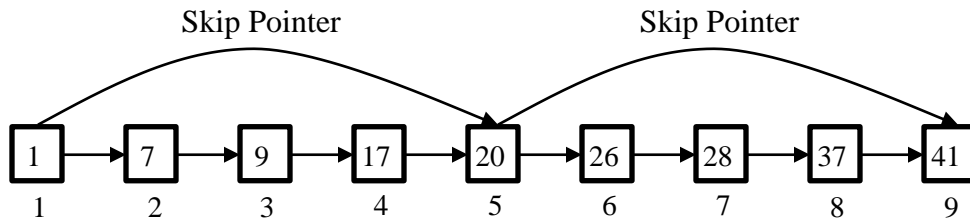
increased number of document imposes challenges to the query results refining process [20, 21]. An example of inverted indexing is shown in Figure 2.1, which shows words stored in the dictionary alongside with the postings where each word is found in.



**Figure 2.1: Sample of Inverted indexing with dictionary and postings.**

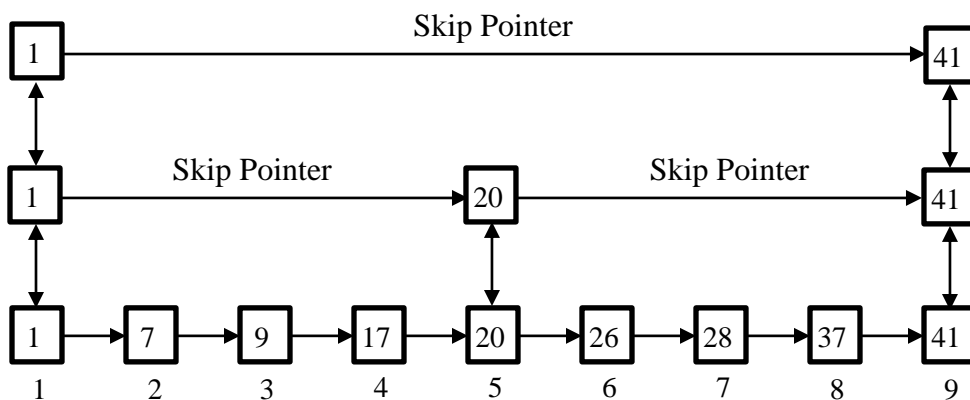
Finding intersections by iterating through one list and search for that document into the other list is very time-consuming, especially with the rapidly increasing number of documents uploaded to the internet every day. Thus, it is important to use faster methods that have the capability of providing the exact same results in shorter time. Such methods are based on taking larger steps in linked lists and check if these larger steps are useful or not. Such technique is known as Skip Pointers (SP) [22].

Skip pointers are shortcuts placed on different places over a postings list. When searching for a specific item in that list, it is possible to take these shortcuts, and compare the values they point at, to the item being searched from. Then, if that item is found to be located before the shortcut, it is possible to go back and search for it in the items between the last step before jumping to the shortcut, and the position of the shortcut. Otherwise, all items previous to the shortcut are neglected. In both cases, the number of steps required to find an item in the ordered list is less than the time required to search for that item by iterating through the list items, one by one [23]. A sample posting list is shown in Figure 2.2, where specific predefined positions have alternative larger steps to try in order to make a decision about the path it is taking.



**Figure 2.2: Sample postings list with skip pointers.**

A more complex method, based on skip pointers, is known as Skip Lists (SL). In this method, skip pointers are arranged in a pyramid-like hierarchy, where shortcuts in a certain level are the skip pointer for the lower level, which may sequentially include skip pointer, for a lower level. This method provides a path for the query being search about the position that queried value may be located in. In the example shown in Figure 2.3, the skip pointer in the third level may be used to directly conclude whether a certain value may exist in the posting list, which is located at the bottom of the hierarchy or not. Then, the second level directs the search operation to search in the right part of the posting list or the left one. In such simple hierarchy, half of the postings in the lists are eliminated from the search operation using only two comparisons.



**Figure 2.3: Sample hierarchy of a skip list.**



There are many applications based on using skip pointers and skip lists, in order to find intersections among linked lists, such as the method proposed for a storage system in [24] to provide a decentralized structure and the Big Table system [25] to manage storage and retrieval of huge data using a distributed storage. Skip lists are always widely used alongside with the inversely indexed material in search engines [26]. The reason behind using the skip lists and pointers in such technologies is their capability in processing huge number of documents faster than any other proposed methods [27].

The distribution of skip pointer in a posting list is an important factor that affected the number of steps taken by comparisons to reach a decision about the searched query, whether a value exists in this list or not. The less number of skip pointers means a larger number of values to search in using a step by step search, in case that the value at the end of the skip pointer is larger than the value being searched for. Otherwise, a larger number of skip pointers results in smaller portions of the postings list, which reduces the benefit of using skip pointers. The appropriate number of skip pointers results in faster intersection process, as it reduces the number of comparisons for each intersection process [26].

The classic skip pointers technique is simple, where the value being searched for is compared to the value that the skip pointer is pointing at before deciding the next step. If the value that the skip pointer is pointing at is less than the value being searched for, then the next position is the position next to where the skip pointer is pointing at. Otherwise, the next search position is the next position that the skip pointer is pointing from [22]. The skip pointers in this method are distributed in a way where the distance between one skip pointer and another is equal to the square root of the number of postings in this postings list, which means that the number of positions that are skipped by each skip pointer is related to the number of postings in a list, where larger lists produce larger number of postings skipped by each skip pointer. Thus, this method may also have some performance issues with a huge data.

An improved approach is proposed in [28], where the next position that skip pointer is pointing at is compared to the value being searched from, and the size of the step taking by the skip pointer is measured. The number of skipped position by the initial skip pointers, in this method, is equal to three halves of the square root of the number of postings in the list. If the value in the next position of the skip pointer is larger than the required value, the size of the next step is compared to a preset threshold in order to decide the position of the next step. If the step size is larger than the threshold value, the number of steps between the start and the end of the skip pointer, excluding the start and end positions, is divided by two and a skip pointer is created which points to the midway of the previous skip pointer.

Then, the value stored in the position where the new skip pointer is pointing at is compared to the required value in order to decide the position that a step by step search starts from. If the value in the position that the new skip pointer is pointing at is less than the value being searched for, then the previous procedure is repeated, until the queried value is larger than the value in the position the skip pointer is pointing to, or the number of cells between the start and the end of the skip pointer is less than the predefined threshold. Otherwise, when the queried value is larger than the value where the skip pointer is point at, or the number of skipped postings is less than the predefined threshold, a step by step search starts at this point. This method has a limited capacity to reduce the size of the portion, where a step by step search is conducted, when applied to find intersections among posting lists.

## **CHAPTER THREE**

### **METHODOLOGY**

The main benefit behind using skip pointers is to reduce the time required to find intersections between two posting lists that contain a huge number of postings. This time reduction is achieved by using less number of comparisons to find a certain value from one list in another. To do so, skip pointers provide shortcuts to farther values, so that it is possible for the algorithm to compare the value in the position of the next pointer, and the value being searched for. If the value in that position is less than the value being searched for, then it is possible for the search operation to neglect all values between the current position and the position of the next skip pointer. The procedures followed by algorithms, when the value in the position of the next skip pointer is larger than the required value, are different from one method to another.

#### **3.1. Existing Skip Methods**

In order to provide a better illustration for the proposed improvement in the way skip pointers are used to accelerate an intersection process between two postings lists, it is important to discuss, in details, how skip pointers work. Skip pointers are located on different locations in the posting list, so that, when a search operation is executed, the value in the position where the skip pointer is pointing at is compared to the required value, in order to decide whether the required value may be in the range between the current position and the position where the next skip pointer is located at, or is pointing at. This allows skipping multiple positions without the need to go through them one by one, which allows faster processing.

In the classic method, skip pointers are distributed in the postings list, where the number of skipped position is equal to the square root of the number of postings in that list. As the identification numbers of the documents in that postings list are arranged in an ascending number, it is possible for the search algorithm to decide the position of the

next step, whether to be where the skip pointer is pointing at, or the position next to current position. The decision is based on the comparison between the value being searched for and the value in the location where the skip pointer is pointing at. If the value being searched for is larger than the value in the far location, then all values in the positions until the next skip pointer are neglected, because the values are arranged in an ascending order, and the value where the skip pointer is pointing is less than the queried value, which makes it impossible for that value to be in these positions. The pseudo code for this algorithm is shown in Algorithm 3.1.

**Algorithm 3.1: Postings lists intersection using skip pointers method.**

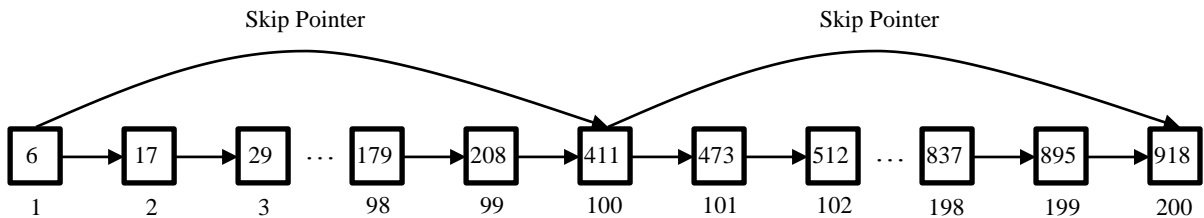
```

IntersectWithSkips( $p_1, p_2$ )
1   $intersections \leftarrow \langle \rangle$ 
2  while  $p_1 \neq null$  and  $p_2 \neq null$ 
3  if  $docID(p_1) = docID(p_2)$ 
4      then ADD( $intersections, docID(p_1)$ )
5           $p_1 \leftarrow next(p_1)$ 
6           $p_2 \leftarrow next(p_2)$ 
7  else if  $docID(p_1) < docID(p_2)$ 
8      then if  $hasSkip(p_1)$  and  $(docID(skip(p_1)) < docID(p_2))$ 
9          then while  $hasSkip(p_1)$  and  $(docID(skip(p_1)) \leq docID(p_2))$ 
10             do  $p_1 \leftarrow skip(p_1)$ 
11             else  $p_1 \leftarrow next(p_1)$ 
12     else if  $hasSkip(p_2)$  and  $(docID(skip(p_2)) < docID(p_1))$ 
13         then while  $hasSkip(p_2)$  and  $(docID(skip(p_2)) \leq docID(p_1))$ 
14             do  $p_2 \leftarrow skip(p_2)$ 
15             else  $p_2 \leftarrow next(p_2)$ 
16  return intersections

```

For example, if a value of 179 is being searched for in the postings list shown in Figure 3.1, then after comparing the value in the first position with the value being searched for, the value in the position where the skip pointer is pointing at is compared to the value being searched, because the value in the first position is less than the value being searched for. When the value in the 100<sup>th</sup> position is found to be greater than the value being searched for, the conclusion now is that if the value exists in the postings list, then it should be in the positions between the 1<sup>st</sup> and 100<sup>th</sup>. A step by step search is then started to find the required value in that range.

Another case is when, for example, the value being searched for is 895. In this case, the search also starts from the first position, then, as the value being searched for is larger than the value in that position, the next value compared to the required value is where the skip pointer is pointing at, which is the 100<sup>th</sup> position. As the value in that position is still less than required value, all values prior to this position are neglected, and the next position used in comparison is where the second skip pointer is pointing at. As it is larger than the required value, the step by step search then starts from the 100<sup>th</sup> position, until the required value, or a larger value, is found in step by step search.



**Figure 3.1: Sample postings list with skip pointers.**

The improved skip method, proposed in [28], where the number of positions skipped by every skip pointer is equal to three halves of the square root of the total number of postings in the posting list. This method uses the same technique when moving forward, until a skip pointer, which points to a position that holds a larger value than the value being searched for, is reached. At this step, the algorithm does go back to where the skip pointer leaves from, it goes to the midway between the start and the end positions of the skip pointer. This value is compared to the queried value in order to decide the next step.

If the value in the mid position of the skip pointer is found to be larger than the queried value, a new midway is selected between the position of lastly checked value and the start of the skip pointer. This process is repeated until the number of positions between a newly set skip pointer, which is selected at the midpoint of the last skip pointer, becomes less than a predefined threshold, or when the value in that position becomes less than the queried value, where in this case a step by step search is started. The algorithm used to find intersections between two postings lists using the improved skip pointers method is shown in Algorithm 3.2.

**Algorithm 3.2: Postings lists intersection using improved skip pointers method.**

```

IntersectWithImSkips( $p_1, p_2$ )
1   $intersections \leftarrow \langle \rangle$ 
2  while  $p_1 \neq null$  and  $p_2 \neq null$ 
3  if  $docID(p_1) = docID(p_2)$ 
4      then ADD( $intersections, docID(p_1)$ )
5           $p_1 \leftarrow next(p_1)$ 
6  else if  $docID(skip(p_1)) \leq docID(p_2)$ 
7      then while  $docID(skip(p_1)) \leq docID(p_2)$ 
8          do  $p_1 \leftarrow skip(p_1)$ 
9  else  $min = p_1 + 1$ 
10          $max = skip(p_1) - 1$ 
11         while ( $min \leq max$  and not( $found$ ))
12              $mid = (min + max)/2$ 
13             If  $docID(mid) > docID(p_1)$ 
14                 then  $max = mid - 1$ 
15             else if  $docID(mid) < docID(p_2)$ 
16                 then  $min = mid + 1$ 
17             else  $found = true$ 
18                  $p_1 = min$ 
19  else if  $docID(p_1) < docID(p_2)$ 

```

```

20     then if  $docID(p_1) - docID(p_2) < midP$ 
21          $p_1 \leftarrow next(p_1)$ 
22     else
23         If  $docID(skip(p_1)) \leq docID(p_2)$ 
24     then while  $docID(skip(p_1)) \leq docID(p_2)$ 
25         do  $p_1 \leftarrow skip(p_1)$ 
26     else  $min = p_1 + 1$ 
27          $max = skip(p_1) - 1$ 
28         while ( $min \leq max$  and  $not(found)$ )
29              $mid = (min + max)/2$ 
30             If  $docID(mid) > docID(p_1)$ 
31                 then  $max = mid - 1$ 
32             else if  $docID(mid) < docID(p_2)$ 
33                 then  $min = mid + 1$ 
34             else  $found = true$ 
35                  $p_1 = min$ 
36                 ADD( $intersections, docID(p_1)$ )
37         else  $p_2 \leftarrow next(p_2)$ 
38 return intersections

```

In summary, the classic skipping technique takes forward skips, equal to the square root of the total number of postings in the posting list, in order to find whether the queried value may be in that range of positions or not. In case it is in that position, a step by step search is started from the position next to the position where the last skip pointer starts. While in the improved method, where pointers skip a number of positions equal to three halves of the square root of the total number of postings in the posting list. When a position pointed at by a skip pointer has a value larger than the value queried for, the midway position between the start position and the end position of the skip pointer is selected for comparison with the queried value. This process is repeated until the number of positions skipped by the skip pointer is less than a predefined threshold,

or the value in the position where the latest skip pointer is pointing at is less than the queried values, where a step by step search is started. Thus, in both methods the step by step search start as soon as a skip pointer, with a starting position that has a less value than the queried value and pointing at a position that has a larger value than the queried one, is found.

### **3.2. The Proposed Method: Dynamic Skip Pointers**

As the number of the documents is rapidly increasing, and with the significant number of studies that proposed methods to improve the information retrieval process from each document, creating the postings lists has shown significant improvement. Thus, the bottleneck that is limiting the execution of search queries, is finding intersections among these lists. As the use skip lists is also based on skip pointers, it is important to improve the performance of the skip pointers technique even more, in order to process huge lists within a reasonable amount of time. As demonstrated in the previous section, the classic method skips forward positions until it reaches a skip pointer that points to a position with a larger value than the queried value, where a step by step search is started. While the existing improved method also skips forward until it reaches the same position as the classic techniques, then it starts to move forward one step at a time, and use smaller skip pointers to reach for a position that holds a smaller value than the queried one.

Although the existing improved technique has the ability to process much more larger lists than the classic technique in the same period of time, the increased number of documents is imposing a new challenge to this technique, which compares the values in the position located halfway, the last skip pointer that points to a larger value than the queried one, is pointing to. When the number of postings in the posting lists goes higher, these steps become huge, and it is possible to neglect a matching value located very close to the position where the last pointer is pointing at.

To override such problems, this study proposes a new method to find intersections in postings lists, based on the skip pointers technique. The new method has the ability to adapt different sizes of postings lists, where the number of skipped positions is



dynamically adjusted depending on the number of remaining postings in the postings list and the number of positions skipped in the last skip action taken by the algorithm.

Unlike the classic skip pointers technique, the proposed method does not use pre-distributed pointers on the postings list. The number of skipped steps is calculated dynamically during the execution of the algorithm; depending on the position of the current pointer and the position of the last detect position, where a larger value, compared to the queried value, is found. The number of skipped position is equal to half the count of position in between the last selected position to compare their values to the queried value. In the beginning, if the required value is larger than the value in the current position, half of the postings are skipped in order to check the value in the midway position between where the pointer is standing, and the end of the list. In case that this value, in the midway, is larger than the queried value, for example, then the position next to where the pointer at, is set as the minimum of the range, where the queried value should be in, if exists. Then the position before the position where the last skip pointer pointed at, is considered as the maximum of that range.

After setting the minimum and maximum positions of the range that may include the queried value, the midway of that range is selected as the new position that skip pointer is pointing at. This process is repeated until the queried value is found, or the size of the skipped steps becomes equal to two. If any of these cases occurs, a new skip pointer is created, by computing the midpoint of the last position that holds less value than the queried value and the end of the postings list. This dynamic skipping used in this technique assures different skips when postings lists of different sizes are used. When a larger list is used, then the skip pointer automatically accommodates to the size of the list, as the size of the list is always used to update the skip pointer. Then, by eliminating items from one list, based on the values searched for from the other list, the step size starts to get smaller as it gets smaller when specific ranges are concluded to search for the queried value in. Algorithm 3.3 shows the procedure executed by the proposed method in order to find intersections between two postings lists.

**Algorithm 3.3: Dynamic skip pointers postings lists intersection method.**

```
IntersectWithMDSkips( $p_1, p_2$ )
1   $intersections \leftarrow \langle \rangle$ 
2  while  $p_1 \neq null$  and  $p_2 \neq null$ 
3  if  $docID(p_1) = docID(p_2)$ 
4      then ADD( $intersections, docID(p_1)$ )
5           $p_1 \leftarrow next(p_1)$ 
6           $p_2 \leftarrow next(p_2)$ 
7           $skip_{p_1} \leftarrow (list1.size - p_1)/2$ 
8           $skip_{p_2} \leftarrow (list2.size - p_2)/2$ 
9  else if  $docID(p_1) < docID(p_2)$ 
10     then if  $hasSkip(p_1)$  and  $(docID(skip(p_1)) < docID(p_2))$ 
11         then while  $hasSkip(p_1)$  and  $(docID(skip(p_1)) \leq docID(p_1))$ 
12             do  $p_1 \leftarrow skip(p_1)$ 
13              $skip_{p_1} \leftarrow (skip_{p_1} - p_1)/2$ 
14         else
15             If  $skip_{p_1} \leq 2$  then  $skip_{p_1} = (list1.size - p_1)/2$ 
16             else  $skip_{p_1} = skip_{p_1}/2$ 
17              $p_1 \leftarrow next(p_1)$ 
18     else
19         then if  $hasSkip(p_2)$  and  $(docID(skip(p_2)) < docID(p_1))$ 
20             then while  $hasSkip(p_2)$  and  $(docID(skip(p_2)) \leq docID(p_1))$ 
21                 do  $p_2 \leftarrow skip(p_2)$ 
22                  $skip_{p_2} \leftarrow (skip_{p_2} - p_2)/2$ 
23             else
24                 If  $skip_{p_2} \leq 2$  then  $skip_{p_2} = (list2.size - p_2)/2$ 
25                 else  $skip_{p_2} = skip_{p_2}/2$ 
26                  $p_2 \leftarrow next(p_2)$ 
27  return intersections
```

## CHAPTER FOUR

### EXPERIMENTAL RESULTS

#### 4.1. Experiment A

In order to evaluate the performance of the proposed method, the classic skip pointers method, as well as the improved skip pointers method, is implemented alongside with the proposed method. All methods are implemented using Java programming language. Then, a different combination of keywords is searched for, in order to measure and compare the performance of all the methods discussed earlier. The words are classified into three categories that are stop words, which are words like ‘is, are and the’, frequent words and rare words. Words are classified into these categories according to the frequency of the appearance of these words in the literature. Sample literature is selected to execute the experiments from the Project Gutenberg [29]. A total of 642 digital books are downloaded and converted into 900000 digital documents, by using each paragraph of the downloaded books to create a separate document. A computer with an Intel® Core™ i7-7500Q CPU @ 1.6GHZ with 8GB of memory running using Linux Ubuntu 12.4 operating system.

In this experiment, two search queries are executed, where two stop words are used in each search query. The search queries are executed three times using 300000, 600000, and 900000 documents sequentially. The time required to process the postings lists for each of these queries is measure as well as the number of comparisons made to find the intersections between lists. The first two words searched for, in this experiment, are ‘in’ and ‘was’. Time taken by each algorithm, and the number of comparisons made to come up with the intersections lists is summarized in Table 4.1.

**Table 4.1: Time and comparisons count of the three skip methods to find the keywords 'in' & 'was'.**

Searched Docs.	Classic Method		IM Method		DS Method		Intersections
	Time (ms)	Comparisons	Time (ms)	Comparisons	Time (ms)	Comparisons	
300000	0.108	257771	0.072	256933	<b>0.07</b>	<b>228896</b>	80154
600000	0.108	525146	<b>0.093</b>	524850	0.109	<b>468372</b>	165618
900000	0.102	786735	0.127	785050	<b>0.084</b>	<b>686780</b>	241266

Then, another combination of stop words is used to test the performance of each of the three skip methods. The used words are 'the' and 'of'. The time consumed by every method to find intersections between the postings list for these words, as well as the number of comparisons made by each method to find these intersections are summarized in Table 4.2.

**Table 4.2: Time and comparisons count of the three skip methods to find the keywords 'the' & 'of'.**

Searched Docs.	Classic Method		IM Method		DS Method		Intersections
	Time (ms)	Comparisons	Time (ms)	Comparisons	Time (ms)	Comparisons	
300000	<b>0.036</b>	255990	0.088	255990	0.133	<b>255171</b>	185175
600000	<b>0.04</b>	517739	0.099	517739	0.168	<b>516600</b>	376089
900000	<b>0.049</b>	765768	0.112	765768	0.169	<b>763600</b>	567037

Then, the average of both scenarios is computed to conclude the average performance of the skip methods to find intersections of postings list for documents that have the selected stop words. The average performance is shown in Table 4.3 and illustrated visually in Figure 4.1 and Figure 4.2.

**Table 4.3: Average performance of the skip methods using stop words.**

Searched Docs.	Classic Method		IM Method		DS Method	
	Time (ms)	Comparisons	Time (ms)	Comparisons	Time (ms)	Comparisons
300000	<b>0.0566</b>	247675.6	0.0738	247445	0.119	<b>241085</b>
600000	<b>0.0672</b>	499439.8	0.098	499352	0.1494	<b>486738</b>
900000	<b>0.0694</b>	746392.4	0.1124	746052.4	0.147	<b>724056.2</b>

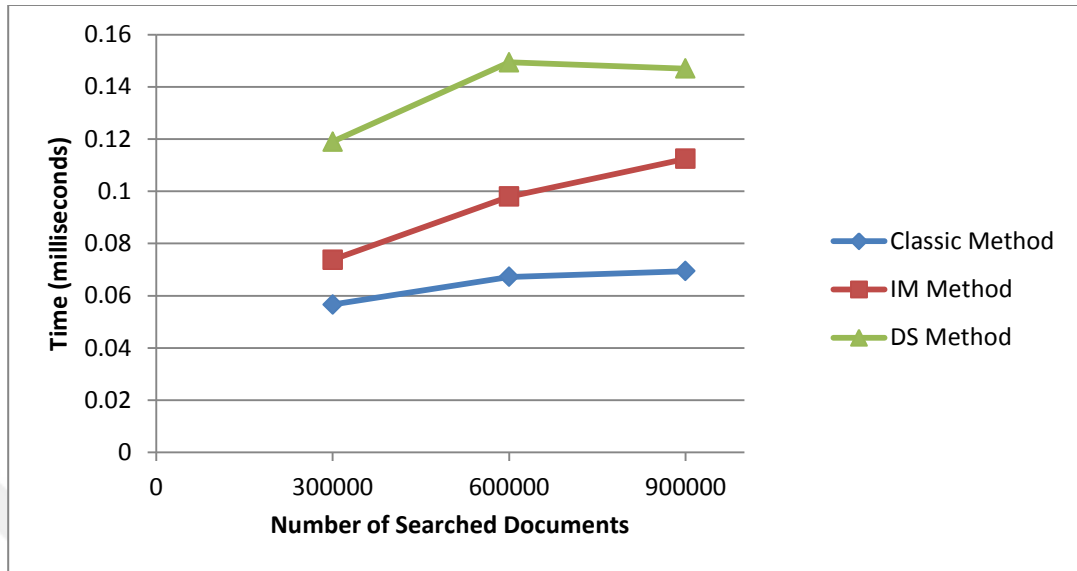


Figure 4.1: Average execution time for the skip methods using stop words.

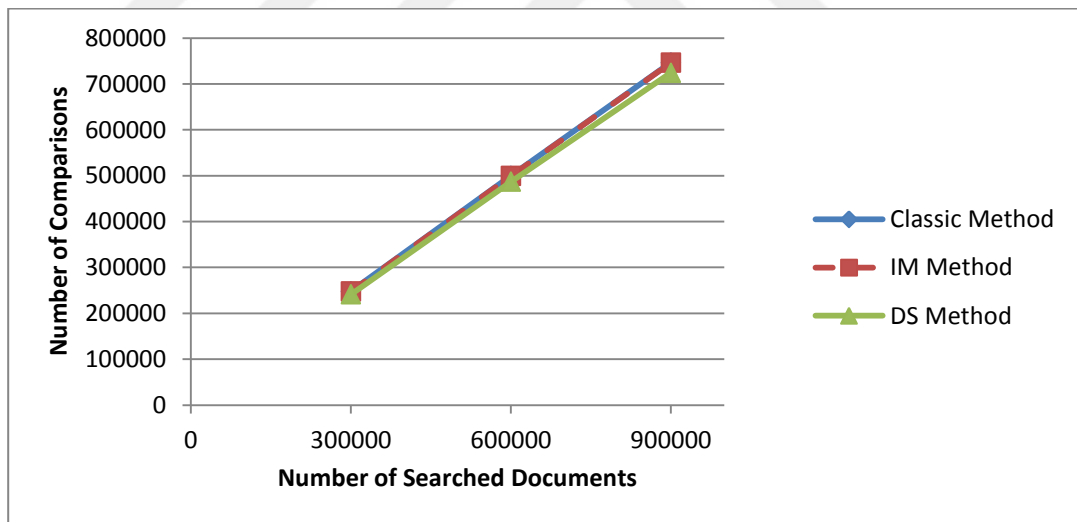


Figure 4.2: Average number of comparisons made by the skip methods using stop words.

## 4.2. Experiment B

In this experiment, two frequent words are queried in the search, where a frequent word is a word found frequently in the documents included in the experiments. The words selected for the first scenario of this experiment are ‘advantage’ and ‘meeting’. The number of comparisons made by each skip method and the time consumed by each

method in order to find intersections in the postings lists of these words are shown in Table 4.4.

**Table 4.4: Time and comparisons of the skip methods to find the words 'advantage' & 'meeting'.**

Searched Docs.	Classic Method		IM Method		DS Method		Intersections
	Time (ms)	Comparisons	Time (ms)	Comparisons	Time (ms)	Comparisons	
300000	0.006	2406	0.006	2406	<b>0.004</b>	<b>1986</b>	4
600000	0.01	5243	0.011	5243	<b>0.007</b>	<b>4244</b>	14
900000	0.015	7959	0.012	7959	<b>0.01</b>	<b>6480</b>	18

Then, the same scenario is repeated, by searching for two frequent words. The words selected for this scenario are 'distance' and 'pass'. Comparisons made by each skip method as well as the time taken by these methods to find the intersections between the postings lists generated for each word are measures, and summarized in Table 4.5.

**Table 4.5: Time and comparisons of the skip methods to find the words 'distance' & 'pass'.**

Searched Docs.	Classic Method		IM Method		DS Method		Intersections
	Time (ms)	Comparisons	Time (ms)	Comparisons	Time (ms)	Comparisons	
300000	0.016	11815	0.015	11815	<b>0.014</b>	<b>7686</b>	104
600000	0.027	25194	0.033	25194	<b>0.02</b>	<b>16142</b>	231
900000	0.033	39041	0.041	39041	<b>0.027</b>	<b>25104</b>	360

The average performance for each skip methods, when two frequent word are queries, is calculated and shown in Table 4.6.

**Table 4.6: Average performance of the skip methods using frequent words.**

Searched Docs.	Classic Method		IM Method		DS Method	
	Time (ms)	Comparisons	Time (ms)	Comparisons	Time (ms)	Comparisons
300000	0.0098	6264	0.0104	6257.8	<b>0.0072</b>	<b>4172.8</b>
600000	0.0182	17188.2	0.022	17188.2	<b>0.0158</b>	<b>13294.6</b>
900000	0.021	20455.8	0.026	20441.2	<b>0.0148</b>	<b>13248.2</b>

For better illustration, average times required by each skip method are shown in Figure 4.3, while number of comparison executed to find intersections are shown in Figure 4.4.

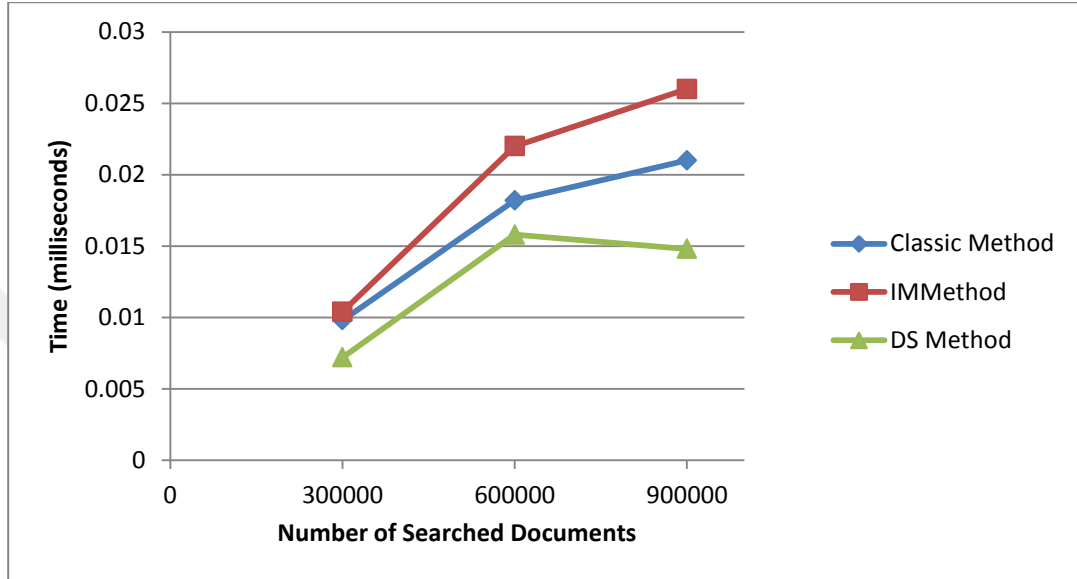


Figure 4.3: Average execution time for the skip methods using frequent words.

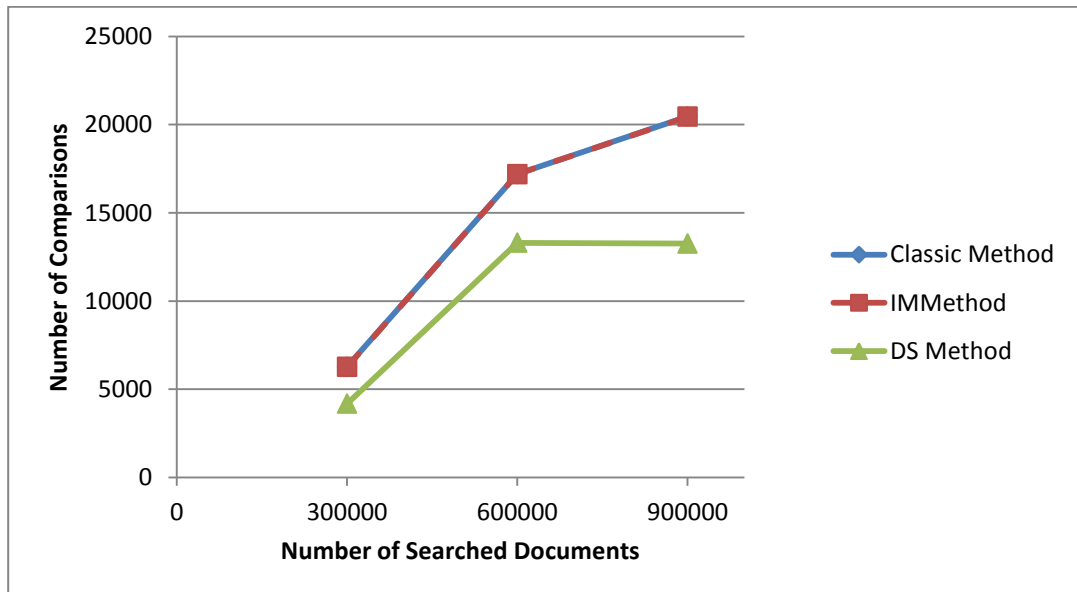


Figure 4.4: Average number of comparisons by the skip methods using frequent words.

### 4.3. Experiment C

In this experiment, two different combinations of words that are rarely seen in the documents included in the experiments, are used for querying the documents, in order to collect the postings lists, for the intersection process using the skip methods. The first section of the experiment is executed using the words ‘huddle’ and ‘people’. The number of comparisons made by each intersection method based on skip pointers, and the time consumed by each method to execute these comparisons, are shown in Table 4.7.

**Table 4.7: Time and comparisons of the skip methods to find the words 'huddle' & 'people'.**

Searched Docs.	Classic Method		IM Method		DS Method		Intersections
	Time (ms)	Comparisons	Time (ms)	Comparisons	Time (ms)	Comparisons	
300000	0	2751	0	2751	0	920	2
600000	0	6225	0.016	6225	0	1630	3
900000	0.015	10277	0.016	10277	0	2279	6

Next, a different combination of rarely used word is queried in order to create postings lists that are intersected using skip pointer techniques. The words that are used in this combination are ‘moment’ and ‘uncle’. The execution time of each method, as well as the number of comparisons made between the two postings lists, in order to find the common documents between these postings lists, are summarized in Table 4.8.

**Table 4.8: Time and comparisons of the skip methods to find the words 'moment' & 'uncle'.**

Searched Docs.	Classic Method		IM Method		DS Method		Intersections
	Time (ms)	Comparisons	Time (ms)	Comparisons	Time (ms)	Comparisons	
300000	0	4569	0	4569	0	2480	14
600000	0.015	11653	0.016	11606	0	5573	36
900000	0.016	16882	0.031	16882	0	7803	48

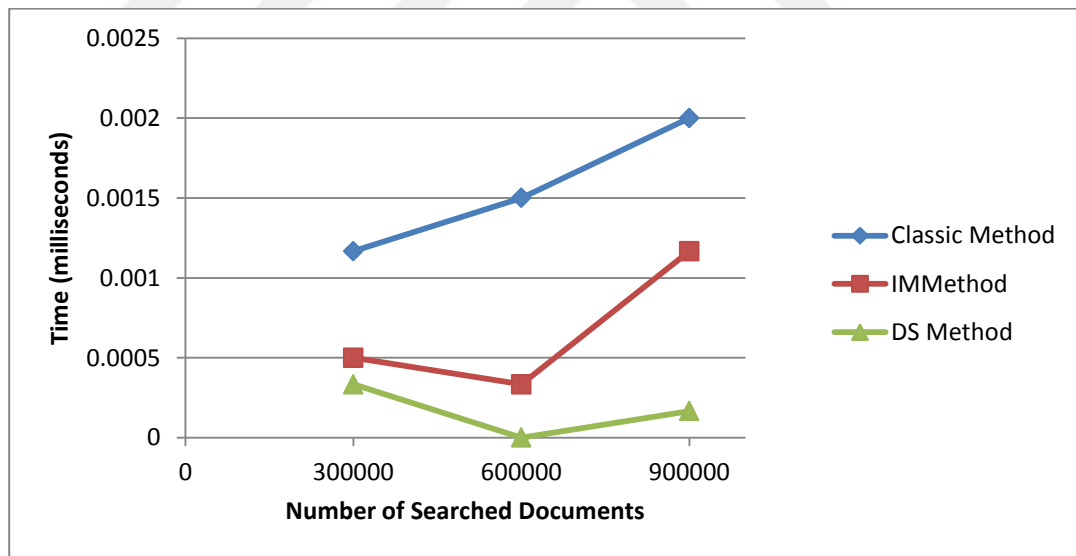
Finally, the average time and number of comparison executed by each skip method in this experiment are shown in Table 4.9. These values are considered as the average performance of the skip methods, when rarely used words are queried.



**Table 4.9: Average performance of the skip methods using rare words.**

Searched Docs.	Classic Method		IM Method		DS Method	
	Time (ms)	Comparisons	Time (ms)	Comparisons	Time (ms)	Comparisons
300000	0.0012	3307.3333	0.0005	3307.3333	<b>0.0003</b>	<b>1614.3333</b>
600000	0.0015	8002.0000	0.0003	7986.3333	<b>0.0000</b>	<b>3397.0000</b>
900000	0.0020	12346.6667	0.0012	12346.6667	<b>0.0002</b>	<b>5010.6667</b>

For better illustration, these values are shown in Figures 4.5 and 4.6. Where Figure 4.5 shows the average execution time of each method when used to find intersections between postings lists generated for documents that contain rarely used words, and Figure 4.6 shows the number of comparisons used by each method in order to find these intersections.



**Figure 4.5: Average execution time for the skip methods using rare words.**

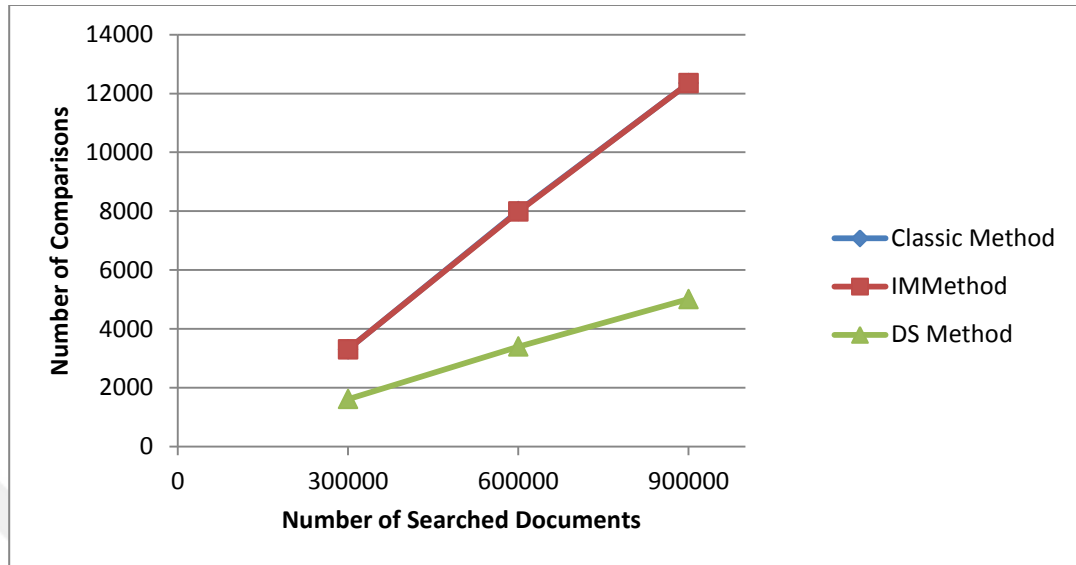


Figure 4.6: Average number of comparisons made by the skip methods using rare words.

#### 4.4. Experiment D

The performance of each method is measured, in this experiment, using two different combinations of one stop word and one frequent word. In the first scenario, the words ‘the’ and ‘associated’ are used to measure the time requires by each method to execute the required number of comparison, according to each method. The measured execution time and number of comparisons made by each method in this scenario, are shown in Table 4.10.

Table 4.10: Time and comparisons of the skip methods to find the words 'the' & 'associated'.

Searched Docs.	Classic Method		IM Method		DS Method		Intersections
	Time (ms)	Comparisons	Time (ms)	Comparisons	Time (ms)	Comparisons	
300000	0.059	91454	0.057	83358	<b>0.02</b>	<b>14659</b>	1614
600000	0.064	246742	0.073	224782	<b>0.024</b>	<b>30892</b>	3323
900000	0.063	400826	0.1	377941	<b>0.037</b>	<b>46503</b>	4931

Moreover, a different combination of one stop word and one frequent word is used, and the execution time and number of comparison for this combination are shown in Table 4.11.

**Table 4.11: Time and comparisons of the skip methods to find the words 'in' & 'meeting'.**

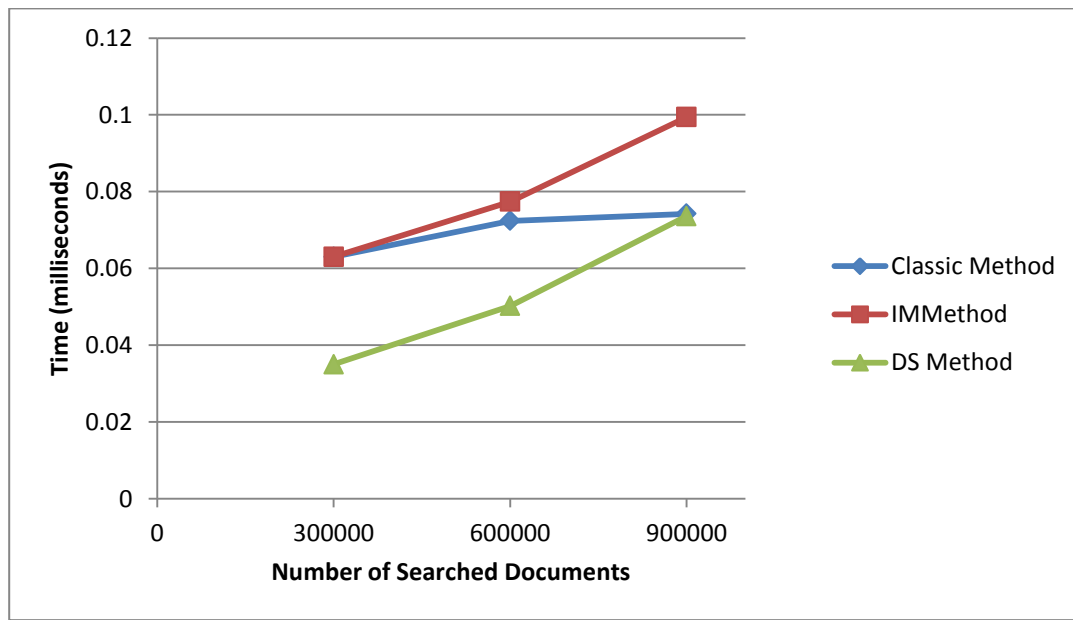
Searched Docs.	Classic Method		IM Method		DS Method		Intersection-ns
	Time (ms)	Comparisons	Time (ms)	Comparisons	Time (ms)	Comparisons	
300000	0.054	142002	0.06	128870	<b>0.017</b>	<b>15188</b>	1122
600000	0.067	331264	0.087	303071	<b>0.03</b>	<b>33559</b>	2643
900000	0.063	534030	0.113	495592	<b>0.039</b>	<b>49185</b>	3661

The average performance of finding intersection between two lists, one resulted from searching documents that contain a certain stop word, and the other list is for the documents that contain a certain frequent word, are shown in the Table 4.12.

**Table 4.12: Average performance of the skip methods using a stop-frequent words combination.**

Searched Docs.	Classic Method		IM Method		DS Method	
	Time (ms)	Comparisons	Time (ms)	Comparisons	Time (ms)	Comparisons
300000	0.063	135690.8	0.063	129369	<b>0.035</b>	<b>42310</b>
600000	0.0724	303757.4	0.0774	290143.2	<b>0.0502</b>	<b>89763</b>
900000	0.0742	471583.4	0.0994	453008.2	<b>0.0736</b>	<b>138368.4</b>

These values, of the average performance, are also illustrated visually. Figure 4.7 shows the execution time of each skip method to find intersections, while Figure 4.8 shows the number of intersections necessary to find these intersections.



**Figure 4.7: Average execution time for the skip methods using stop-frequent words combination.**

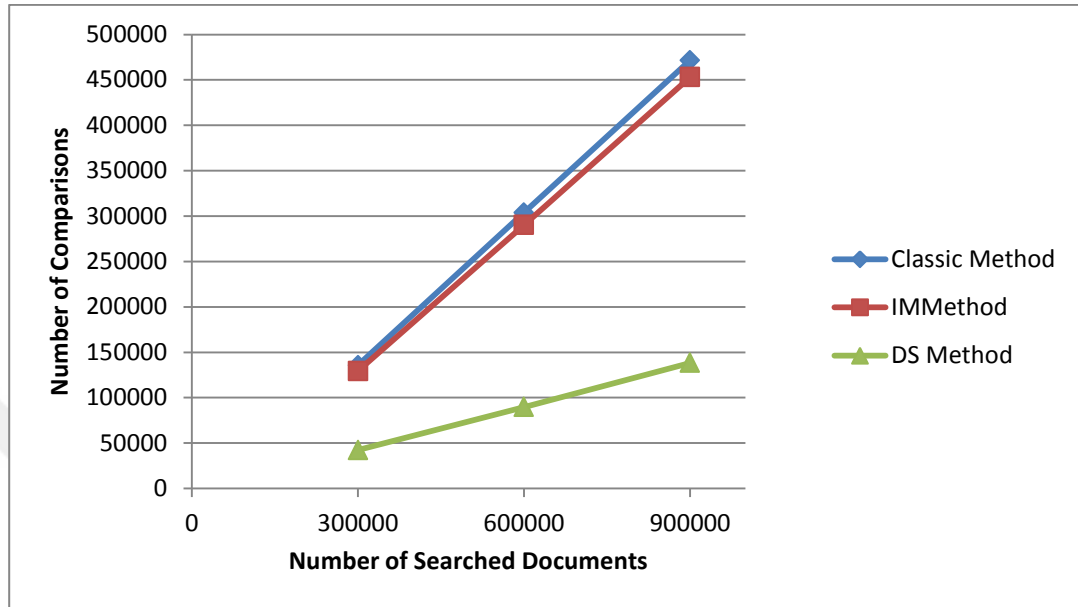


Figure 4.8: Average number of comparisons for the stop-frequent words combination.

#### 4.5. Experiment E

The scenarios used in this method are based on using one stop word and one rarely used word in order to create two postings lists, one for each. Then, the performance of each skip method is tested by measuring the number of comparisons that each method makes, in order to find the intersections, and the consumed by these methods to execute the comparisons, and find the intersections. Table 4.13 shows the number of comparisons that each method requires in order to find intersections in the postings lists, and the execution time to search for the words ‘be’ and ‘continent’.

Table 4.13: Time and comparisons of the skip methods to find the words 'be' & 'continent'.

Searched Docs.	Classic Method		IM Method		DS Method		Intersect ions
	Time (ms)	Comparisons	Time (ms)	Comparisons	Time (ms)	Comparisons	
300000	0.027	41446	0.052	35783	<b>0.009</b>	<b>5177</b>	268
600000	0.047	56701	0.058	53826	<b>0.013</b>	<b>8041</b>	468
900000	0.067	169984	0.068	152064	<b>0.014</b>	<b>16534</b>	750

Then, the test is repeated using different combination of one stop word and one rare word. The words used in this scenario are ‘it’ and ‘grins’. The performance summary of the skip methods used to find the intersections between the postings lists, one list for each word, are shown in Table 4.14.

**Table 4.14: Time and comparisons of the skip methods to find the words 'it' & 'grins'.**

Searched Docs.	Classic Method		IM Method		DS Method		Intersecti- ons
	Time (ms)	Comparisons	Time (ms)	Comparisons	Time (ms)	Comparisons	
300000	0.006	2247	0.025	1927	<b>0</b>	<b>228</b>	6
600000	0.015	13496	0.034	8644	<b>0.001</b>	<b>603</b>	25
900000	0.023	19353	0.036	10525	<b>0.002</b>	<b>889</b>	39

The average performances of each skip methods, when used to find intersections between lists, one created for a stop word and the other is created for a rarely used word, are summarized in Table 4.15.

**Table 4.15: Average performance of the skip methods using a stop-rare words combination.**

Searched Docs.	Classic Method		IM Method		DS Method	
	Time (ms)	Comparisons	Time (ms)	Comparisons	Time (ms)	Comparisons
300000	0.02	25137.4	0.0318	21424.2	<b>0.0056</b>	<b>3395.2</b>
600000	0.0266	56002.6	0.0374	50198.2	<b>0.0068</b>	<b>6476.4</b>
900000	0.0342	104377.4	0.05	91453.6	<b>0.0094</b>	<b>9879.6</b>

These measures are also illustrated visually in the Figure 4.9 and 4.10. Where Figure 4.9 demonstrated the timed consumed by each method to process the lists created for such combination, while Figure 4.10 shows the number of comparisons that each method requires to find these intersections.

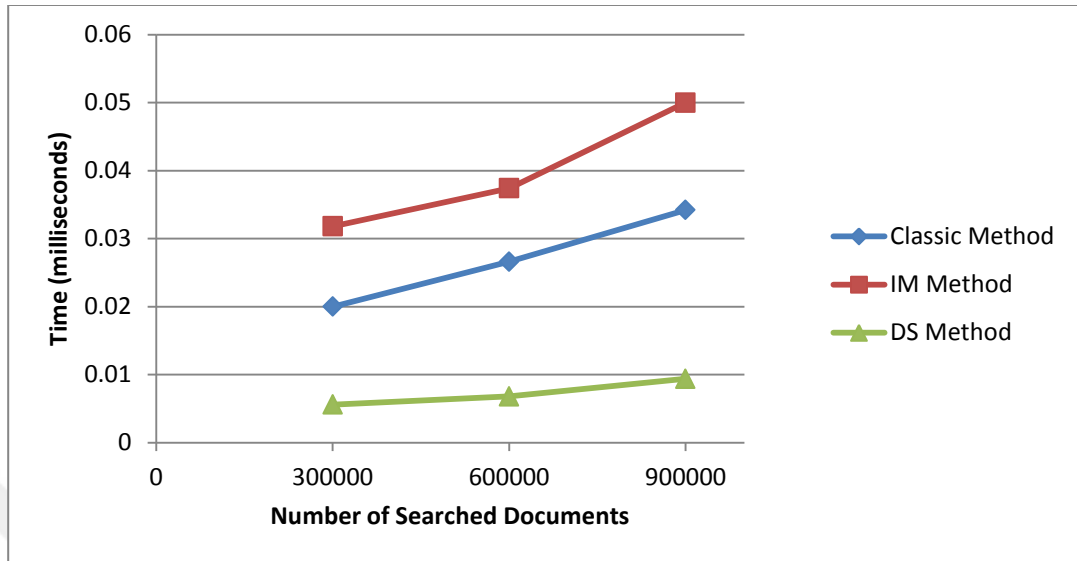


Figure 4.9: Average execution time for the skip methods using stop-rare words combination.

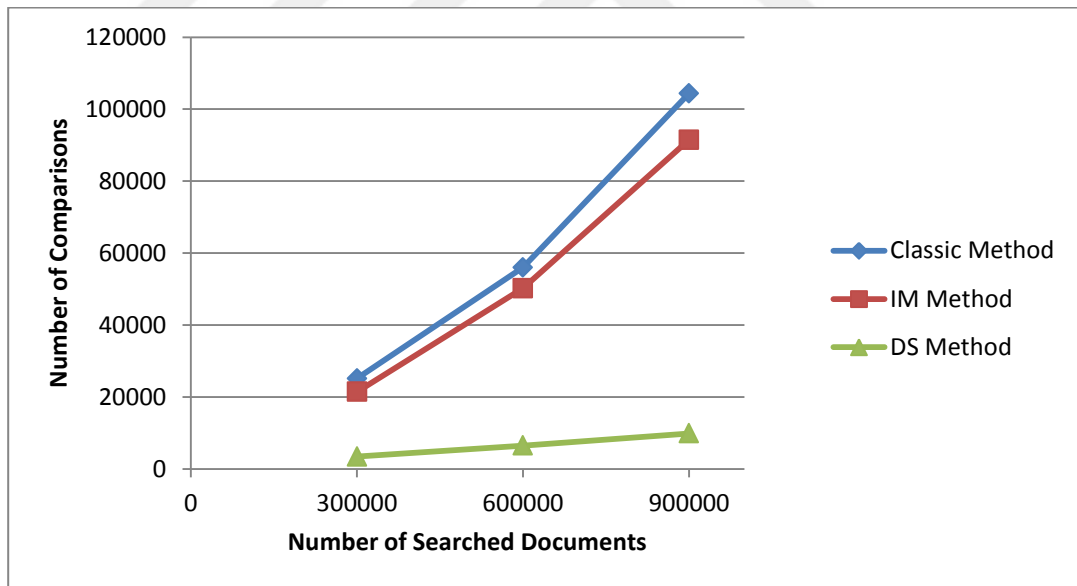


Figure 4.10: Average number of comparisons for the stop-rare words combination.

## CHAPTER FIVE

### DISCUSSION

The average performance of the proposed method in Experiment A, shows that despite the longer execution time, it has found the same number intersections by executing less number of comparisons. In the first part of the experiment, the proposed method has outperformed both methods in two occasions, where the IM method has outperformed the proposed method only when 600000 documents are used for the search. While in the second part of the same experiment, the classic method consumed less time, than both the IM and the proposed method, in order to find the intersections between the postings lists. Although the proposed method executed less number of comparisons, the time consumed by each comparison in the proposed method is higher than that in the classic method. Thus, when postings lists are created for a stop words combination, the enormously high number of documents in each posting lists makes it more efficient to go directly in a step by step search.

The use of two stop words is rarely applicable when documents are searched according to their contents. These words do not add any improvements to the search schema, but it requires more time to process such queries. Thus, in most search queries that include stop words, stop words are either neglected or used together with the next word, so that the results of the search are more related to the required query, and is executed in reasonable time interval [30-32]. Moreover, the proposed method is also able to process these words in shorter time, compared to the other two methods, when a moderately used stop word is included in the query, such as the word 'was', which is used in the first part of Experiment A. Eventually, the classic method has better overall performance, compared to the IM method and the proposed method, in finding intersections between two high density lists, as shown in Table 4.3.

In experiment B, the high performance of the proposed method is well illustrated when an intensive reasonable search is executed. The search queries that are tested in this experiment included two frequently used words, in the documents included in the search. This experiment shows the high capability of the proposed method in handling the intersection of postings lists that are generated for search queries that include frequently used words in the literature being searched in, as shown in Table 4.6. The proposed method has outperformed the other two methods in different combinations of such words, where the number of comparisons as well as the execution time are less than the classic and the IM methods.

Moreover, the performance of the proposed method is compared to the other two methods by searching for two rarely used word, in the literature included in the search, in Experiment C. The results show that the proposed method has outperformed the classic and the IM methods with a relatively more difference, compared to the results of earlier experiments in this study. The proposed method consumes less time and require less comparisons, in order to find the same number of intersections in both scenarios experimented in Experiment C. The overall performance of the compared methods, shown in Table 4.9 shows that, despite the huge difference in performance, the IM method has closed performance than the classic method, compared to the proposed method.

Eventually, experiments D and E measure the performance of the compared methods, when used to find intersections between postings lists, one created for a stop words, while the other is created for a frequent word or rare word, consequentially. The results of all the scenarios tested in these experiments show the high performance of the proposed method in comparison to the other two methods. The average performance of the compared methods when a stop word is used with a frequently used word, summarized in Table 4.12, show that the proposed method has significantly better performance than the other methods. While the difference in the measured performance shows the higher superiority of the proposed method when a combination of a stop word and a rare word is used, as shown in Table 4.15.



In summary, the results show that the proposed method has an overall better performance than the other two methods in real life situation, including some intensive search queries. The difference of the performance between the proposed method, from one side, and the methods used in the experiments, for comparisons, is noticed to get bigger when the density of documents in one, or both, of the postings lists is decreased. In other words, the superiority of the method is noticed to be increased regardless to the number of documents in the postings lists, but is related to the average difference between the identification number of adjacent postings in the postings lists. This is caused by the relatively more complex calculations done in the proposed method compared to the other methods, wherein most of the comparisons, the proposed method computes the midpoint of the last two postings where the skip point left from and points to. Such calculation eventually returns the next cell where the skip pointer is pointing from, in dense lists. Thus, it is simpler to use the traditional method of going to the next posting. This effect only appears when the non-practical queries of searching a combination of two stop words.

## CHAPTER SIX

### CONCLUSION

There are many applications that require finding intersections between two ordered lists [33-35]. One of the most important applications of these methods is to find documents that include a combination of words. The accumulatively increasing number of documents imposes challenges to the existing methods, in order to perform the queried search and return the results in a reasonable interval of time. Many methods are proposed to speed up the process of finding documents related to a single word. These methods create postings, where each posting represents the identification number of a document that has this word. These postings are grouped into lists known as postings lists.

In order to find the results of a multiple-word query, it is important to find the documents that exist in the postings lists of each word. In other words, they represent the results of intersecting these postings lists. Even in search queries that include more than two words, most methods are based on finding intersections between two postings lists then pass the new list, generated from the results of the intersection, to be compared to the other postings list, and so on. To accelerate this process of finding these intersections, multiple techniques are proposed based on skip pointers. Where skip pointers are located on different locations on the list, to provide an alternative route with larger step size for the specific locations in postings list.

The classic method predefines the locations of the skip pointer, by distributing them on the postings lists, using a step size equal to the square root of the size of the postings list. When a position with a skip list is reached during the search, the position of the next skip pointer is compared to the value being searched for. If the value at the next skip pointer is less than the required value, the postings prior to the position of the next skip pointer are neglected. This process is repeated until the end of the list, or when the value

in the position of the next skip pointer is larger than the required value, then a step by step search is started.

An improvement for this method is proposed in an earlier study where the same procedure is executed while the value in the position where the next skip pointer is pointing at, is larger than the queried value, until a position reached where the next skip pointer points at a position that has a value larger than the queried value. At this point, the value in the midway position is compared to the queried value in order to decide whether to start a step by step search, in case that the queried value is larger than the value in the midway position. Or, the midway position of the last midway and the last checked position is selected as a new skip point. This procedure is repeated until a value equal to, or less than, the queried value is found, or a predefined threshold is crossed.

In this study, a new method is proposed to find intersections between two postings lists, based on the skip pointers technique. The proposed method places a skip pointer at the midway between the last compared position, with less value than the queries value, and the last position, with a value higher than the queried value. Then, the value in that position is compared to that position in order to decide the part of the postings list that may include the queried value. This procedure is repeated until the queried value is found in the list, or the difference between the last two positions is equal to, or less than, two.

The experiments conducted in this study shows the superiority of the proposed method in different combination of words. The improvement in the performance of the proposed method, compared to the classic and IM methods, is noticed to have better performance when one or both of the postings lists being processed has led dense postings, regardless to the number of postings in that list. Less dense postings lists are lists that have a larger average difference between values stored in adjacent positions.

In future studies, it is recommended to test the proposed method in a multi-layer hierarchy in order to find intersections among multiple postings lists, by processing multiple pairs of postings lists simultaneously, which may reduce the time consumed to find intersections among these lists.

## REFERENCES

- [1] S. Vigna, "Quasi-succinct indices," in *Proceedings of the sixth ACM international conference on Web search and data mining*, 2013, pp. 83-92.
- [2] J. Lin and A. Trotman, "The role of index compression in score-at-a-time query evaluation," *Information Retrieval Journal*, vol. 20, pp. 199-220, 2017.
- [3] J. Wang, C. Lin, R. He, M. Chae, Y. Papakonstantinou, and S. Swanson, "MILC: inverted list compression in memory," *Proceedings of the VLDB Endowment*, vol. 10, pp. 853-864, 2017.
- [4] E. K. F. Dang, R. W. P. Luk, and J. Allan, "Fast forward index methods for pseudo-relevance feedback retrieval," *ACM Transactions on Information Systems (TOIS)*, vol. 33, p. 19, 2015.
- [5] I. Shlyakhter, P. C. Sabeti, and S. F. Schaffner, "Cosi2: an efficient simulator of exact and approximate coalescent with selection," *Bioinformatics*, vol. 30, pp. 3427-3429, 2014.
- [6] B. Ye, "A Set Intersection Algorithm Via x-Fast Trie," *JCP*, vol. 11, pp. 91-98, 2016.
- [7] B. B. Cambazoglu, E. Kayaaslan, S. Jonassen, and C. Aykanat, "A term-based inverted index partitioning model for efficient distributed query processing," *ACM Transactions on the Web (TWEB)*, vol. 7, p. 15, 2013.
- [8] W. Jung, H. Roh, M. Shin, and S. Park, "Inverted index maintenance strategy for flashSSDs: Revitalization of in-place index update strategy," *Information Systems*, vol. 49, pp. 25-39, 2015.
- [9] D. Stalnaker and R. Zanibbi, "Math expression retrieval using an inverted index over symbol pairs," in *DRR*, 2015, p. 940207.
- [10] L. M. Abualigah, A. T. Khader, M. A. Al-Betar, and M. A. Awadallah, "A krill herd algorithm for efficient text documents clustering," in *Computer Applications & Industrial Electronics (ISCAIE), 2016 IEEE Symposium on*, 2016, pp. 67-72.
- [11] M. Shakiba, N. Ale Ebrahim, M. Danaee, K. Bakhtiyari, and E. Sundararajan, "A Comprehensive Comparison of Educational Growth within Four Different Developing Countries between 1990 and 2012," 2016.

- [12] S. Dumais, E. Cutrell, J. J. Cadiz, G. Jancke, R. Sarin, and D. C. Robbins, "Stuff I've seen: a system for personal information retrieval and re-use," in *ACM SIGIR Forum*, 2016, pp. 28-35.
- [13] A. Berger and J. Lafferty, "Information retrieval as statistical translation," in *ACM SIGIR Forum*, 2017, pp. 219-226.
- [14] S. Büttcher, C. L. Clarke, and G. V. Cormack, *Information retrieval: Implementing and evaluating search engines*: Mit Press, 2016.
- [15] W. Sun, B. Wang, N. Cao, M. Li, W. Lou, Y. T. Hou, *et al.*, "Verifiable privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, pp. 3025-3035, 2014.
- [16] J. Lafferty and C. Zhai, "Document language models, query models, and risk minimization for information retrieval," in *ACM SIGIR Forum*, 2017, pp. 251-259.
- [17] M. Zhitomirsky-Geffet, J. Bar-Ilan, and M. Levene, "Testing the stability of "wisdom of crowds" judgments of search results over time and their similarity with the search engine rankings," *Aslib Journal of Information Management*, vol. 68, pp. 407-427, 2016.
- [18] F. Shoeleh, M. Azimzadeh, A. Mirzaei, and M. Farhoodi, "Similarity based Automatic Web Search Engine Evaluation," in *Telecommunications (IST), 2016 8th International Symposium on*, 2016, pp. 643-648.
- [19] R. Bodenheim, J. Butts, S. Dunlap, and B. Mullins, "Evaluation of the ability of the Shodan search engine to identify Internet-facing industrial control devices," *International Journal of Critical Infrastructure Protection*, vol. 7, pp. 114-123, 2014.
- [20] V. Ivanov, B. Palyukh, and A. Sotnikov, "Efficiency of genetic algorithm for subject search queries," *Lobachevskii Journal of Mathematics*, vol. 37, pp. 244-254, 2016.
- [21] A. Kalinin, U. Cetintemel, and S. Zdonik, "Searchlight: Enabling integrated search and exploration over large multidimensional data," *Proceedings of the VLDB Endowment*, vol. 8, pp. 1094-1105, 2015.
- [22] C. Manning, P. Raghavan, and H. Schütze, "Introduction to information retrieval/Christopher D," ed: Cambridge University Press, Cambridge, England, 2009.
- [23] L. Qian, Z. Ji, Z. Fu, Q. Wu, and G. Song, "Pre-judgment and Incomplete Allocation Approach for Query Result Cache," *Chinese Journal of Electronics*, vol. 25, pp. 1101-1108, 2016.

- [24] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, vol. 44, pp. 35-40, 2010.
- [25] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, *et al.*, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, p. 4, 2008.
- [26] P. Boldi and S. Vigna, "Compressed perfect embedded skip lists for quick inverted-index lookups," in *String Processing and Information Retrieval*, 2005, pp. 25-28.
- [27] F. Mei, Q. Cao, F. Wu, and H. Li, "A Concurrent Skip List Balanced on Search," in *International Workshop on Advanced Parallel Processing Technologies*, 2017, pp. 117-128.
- [28] F. Aeini, F. Mahmoudi, and N. UsefiFard, "Improved Skips for Faster Postings List Intersection," 2012.
- [29] *Free ebooks - Project Gutenberg.* Available: [http://www.gutenberg.org/robot/harvest?filetypes\[\]=txt](http://www.gutenberg.org/robot/harvest?filetypes[]=txt)
- [30] A. Kilgarriff, "Using corpora as data source for dictionaries," *The Bloomsbury Companion to Lexicography*. London: Bloomsbury, pp. 77-96, 2013.
- [31] N. Raghuvanshi and J. Patil, "A brief review on sentiment analysis," in *Electrical, Electronics, and Optimization Techniques (ICEEOT), International Conference on*, 2016, pp. 2827-2831.
- [32] T. Danisman and A. Alpkocak, "Feeler: Emotion classification of text using vector space model," in *AISB 2008 Convention Communication, Interaction and Social Intelligence*, 2008, p. 53.
- [33] G. Tolosa, E. Feuerstein, L. Becchetti, and A. Marchetti-Spaccamela, "Performance improvements for search systems using an integrated cache of lists+ intersections," *Information Retrieval Journal*, vol. 20, pp. 172-198, 2017.
- [34] M. Ilić, D. Rančić, and P. Spalević, "COMPARISON OF DATA MINING ALGORITHMS, INVERTED INDEX SEARCH AND SUFFIX TREE CLUSTERING SEARCH," *Facta Universitatis, Series: Automatic Control and Robotics*, vol. 15, pp. 171-185, 2016.
- [35] S. Kumar and P. Gupta, "Comparative Analysis of Intersection Algorithms on Queries using Precision, Recall and F-Score," *International Journal of Computer Applications*, vol. 130, 2015.