

N-GRAM-BASED SEARCH PROCEDURE

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By
Szymon Woznica

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

June 2009

Fargo, North Dakota

North Dakota State University
Graduate School

Title

N-GRAM-BASED SEARCH PROCEDURE

By

SZYMON WOZNICA

The Supervisory Committee certifies that this *disquisition* complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

North Dakota State University Libraries Addendum

To protect the privacy of individuals associated with the document, signatures have been removed from the digital version of this document.

ABSTRACT

Woznica, Szymon, M.S., Department of Computer Science, College of Science and Mathematics, North Dakota State University, June 2009. N-gram-based Search Procedure. Major Professor: Dr. Anne Denton.

Efficient querying and discovery of meaningful patterns in data becomes more and more important with accelerating growth of data published every day on the Internet. Tree pruning-based algorithms used in most popular search programs have troubles when dealing with infrequent query strings, limiting the number of returned results that might be of interest to the user. Furthermore, the existing tools are not capable of finding data patterns that could inform the user about the frequency of occurrence and location of a specific set of words in large, user-defined sets of textual data, in an efficient manner.

In this paper, we present a new search tool, which is based on n-grams and modern software technologies. Our tool can efficiently index word n-grams existing in large sets of user-defined, textual data and subsequently assist users in querying the text corpus, helping them to find hidden patterns and their locations in the input data, effectively. We describe an algorithm for extracting word n-grams with a parameter “n” equal to two, three and four, and demonstrate how it can be leveraged by the end-user of the search tool to mine data in a new way. The presented tool offers a unique feature that allows the user to search a set of n-grams, extracted from abstracts of biomedical publications obtained from the U.S. National Library of Medicine (NLM), filtering the search result by words existing in the English language.

The data tier of the search tool is based on the Microsoft SQL Server 2008 supported by a set of Common Language Runtime (CLR) functions and Transact Structured Query Language (T-SQL) based stored procedures, whereas the business logic and the user interface utilizes C# .NET 3.5 libraries to support regular expression patterns, database connection (LINQ to SQL) and multithreaded system operations.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my adviser, Dr. Anne Denton whose constant support and encouragement kept me motivated through my graduate program and for all her valuable suggestions and for always finding time to discuss my work.

I wish to thank other members of my graduate committee, Dr. Kendall Nygard, Dr. Weiyi (Max) Zhang, and Dr. Edward Deckard for their time and helpful suggestions.

Special thanks to my parents, wife, daughter and friends for their support and motivating me to complete my paper and my graduate program.

TABLE OF CONTENTS

ABSTRACT.....	iii
ACKNOWLEDGEMENTS	v
TABLE OF CONTENTS.....	vi
LIST OF TABLES	ix
LIST OF FIGURES.....	x
CHAPTER 1. INTRODUCTION	1
1.1. Overview	1
1.2. N-grams	2
1.3. Problem Statement.....	3
1.4. Proposed Tool.....	4
1.4.1. Overview.....	4
1.4.2. Data Tier	5
1.4.3. Business Logic Tier	6
1.4.4. Presentation Tier – Graphical User Interface.....	6
1.5. Organization of the Paper	8
CHAPTER 2. BACKGROUND	9
2.1. Existing N-gram-based Search Tools.....	9

2.2. MEDLINE® Database	10
2.3. Microsoft SQL Server	14
2.4. LINQ to SQL.....	14
2.5. Common Language Runtime Functions.....	15
2.6. User-defined Functions and Cross Apply mechanism	16
CHAPTER 3. APPLICATION DESIGN.....	17
3.1. Overview	17
3.2. Application Architecture	17
CHAPTER 4. IMPLEMENTATION.....	19
4.1. Input Data Reading.....	19
4.2. N-gram Extraction Algorithm	21
4.3. N-gram Search.....	30
4.4. Result Analysis.....	33
4.5. Full-Text Index Subtleness.....	34
CHAPTER 5. APPLICATION PERFORMANCE ANALYSIS.....	36
5.1. Search Effectiveness Analysis.....	36
5.2. Efficiency Analysis - Overview	38
5.2.1. Database Population and N-gram Extraction.....	38
5.2.2. Search.....	41

5.2.3. Browsing Results	46
CHAPTER 6. CONCLUSION.....	48
6.1. Conclusion.....	48
6.2. Future Work and Limitations	49
REFERENCES.....	50
APPENDIX 1. A SAMPLE FRAGMENT OF XML FILE.....	54

LIST OF TABLES

<u>Table</u>	<u>Page</u>
1. Search Tool Effectiveness Analysis.....	37
2. Properties of the input files	39
3. Scalability test results of the n-gram extraction component	39
4. Scalability test results of the search component for bi-grams.....	42
5. Scalability test results of the search component for tri-grams	43
6. Scalability test results of the search component for quad-grams	45
7. Scalability test results of the result browser component	47

LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
1. Tier interaction diagram.....	5
2. A sample interface of PubMed basic search module	11
3. A sample interface of PubMed advanced search module	12
4. A sample representation of the PubMed search results	13
5. Search tool data model	18
6. Code snippet for method for reading XML file	20
7. A sample set of records in the Abstract table.....	21
8. The T-SQL stored procedure to extract n-grams	22
9. Scalability of the n-gram extraction algorithm	25
10. Code snippet for the helper class “BiGram”	26
11. Code snippet for the table definition used by the CLR function.....	27
12. Code snippet for the “extractBiram” CLR function	27
13. Regular expression pattern used to identify word unigrams.....	28
14. Code snippet for a function that links the CLR function to SQL Server	28
15. Input data manager window	30
16. Search module window	31
17. Code snippet of the LINQ query called for the bigram database search	32
18. Result browser module window.....	34
19. Task 1. Add abstracts to database	40
20. Task 2. Extract n-grams	41

21. Efficiency test results – searching bi-grams.....	43
22. Efficiency test results – searching tri-grams	44
23. Efficiency test results – searching quad-grams	46
24. Efficiency test results – browsing results.....	47

CHAPTER 1. INTRODUCTION

1.1. Overview

Rapidly increasing growth rate of traffic and data available through the Internet [1] as well as public access to large quantities of machine-readable text, makes mining such data and extracting useful information a challenging task. Feeding a general purpose data search engine with a word or phrase is often not the most efficient way to search. Regularly occurring spelling mistakes in search queries (specifically those made in less frequently used words and phrases, such as common biological names) can greatly bias or narrow the number of results returned by the pure Boolean information retrieval programs [2]. Mining user-defined sets of textual data is often limited to scanning text and searching for exact matches of the query strings. Such techniques become very inefficient when dealing with large amount of non-indexed data. These issues continuously encourage computer scientists to develop new, more efficient algorithms and to design computer tools that aid their users in finding information that is relevant to their needs. One group of the techniques for information retrieval from large datasets which is recently gaining more popularity and attention of companies developing broadly used web-based search programs is based on n-grams. Converting text corpus to a set of n-grams has multiple advantages, for example, it allows embedding it in a vector space, i.e. representing as a histogram, thus allowing the textual set of data to be compared to other textual data in an efficient manner [3]. Other, statistical methods, such as z-scores are used to compare sequences by examining, based on their standard

deviation measure, how much each n-gram differs from its mean occurrence in a large set of textual data. N-grams have been also used to measure the likelihood that two sequences come from the same source set [3].

1.2. N-grams

An n-gram is a sub-sequence of n items from a given sequence [3, 4]. N-gram-based algorithms are used in a broad range of research topics in computer science. Most commonly we can find them in machine learning algorithms, spelling of misspelled words, compression algorithms, speech recognition, OCR (optical character recognition), and information retrieval. The last topic (information retrieval) includes searching for documents, for information within documents and for metadata about documents, as well as searching relational databases and the World Wide Web [5]. Two types of n-grams can be distinguished: letter n-grams and word n-grams. In this study we use n-grams of alphanumeric words with n in the range of two to four words. Those n-grams are usually called two-grams (or di-grams), three-grams (or tri-grams), and four-grams (or quad-grams), respectively. So, the sentence: “This is a red fox” would be composed of the following n-grams:

- bi-grams: This is, is a, a red, red fox
- tri-grams: This is a, is a red, a red fox
- quad-grams: This is a red, is a red fox

Typically, if we slice a sentence of length k words into n-grams, it will have a maximum of $k-1$ bi-gram instances, $k-2$ tri-gram instances, $k-3$ quad-gram instances,

and so on. Often, however, the number of unique n-grams is much lower, because the same n-grams occur multiple times and only their frequency values increase. A major benefit of using n-gram-based search algorithms is that single errors present in the input data [6] do not significantly affect overall frequencies in textual patterns.

1.3. Problem Statement

It's both interesting and valuable for the scientists to discover and understand data patterns, existing in a large number of scientific documents, published regularly in the Internet. In the recent years, many information retrieval and web mining tools leveraging n-gram-based algorithms have been presented, to manage and use data available on the web [7]. However, the popular, general purpose search tools lack the functionality that would provide comprehensive, domain specific information beyond a location of documents that are matching a specific keyword, in textual set of data [8]. The existing tools do not offer both high efficiency of indexing corpora, and finding word n-grams that are unique to a domain-specific, user-defined sets of data. One of the existing techniques used in n-gram-based search tools to provide better search efficiency, pruning of infrequent n-grams, has the important disadvantage. It doesn't store information about n-grams occurring infrequently (less often than a defined support threshold value) what may greatly influence quality of the search results.

The ability of the computer program to search and efficiently filter search results based on the existence of the query terms in the English language can help to

significantly narrow the amount of information presented to the user [9]. As a result, it can uncover interesting patterns that are domain specific and would otherwise be hidden. Our tool aims to overcome the described functionality and performance issues, at the same time providing the users high quality information on frequent data patterns and their locations in a specific, user-defined text corpus.

1.4. Proposed Tool

1.4.1. Overview

This paper presents word n-grams based search tool that aids users in efficient information retrieval from a large set of data. The proposed search system is fast, accessible and modular. The data to be searched is first provided in the Extensible Markup Language (XML) format, and then processed by extracting n-grams and their frequencies. The resulting data sets are indexed to deliver quick search results as well as to provide the user with a full-text data context of the n-gram search result obtained from the user-defined textual input.

The architecture of our tool is based on the three-tier model, which is one of the commonly used software architecture patterns. The three-tier model consists of the following three core tiers: data, business logic, and the presentation tier (user interface). Each tier is developed and maintained as an independent module. This approach has the advantage that it allows for developing modular software – in such approach, any tier can be replaced or upgraded independently as requirements or

technologies change [10]. The tier interaction of our tool is shown in Figure 1 and described in the following subchapters.

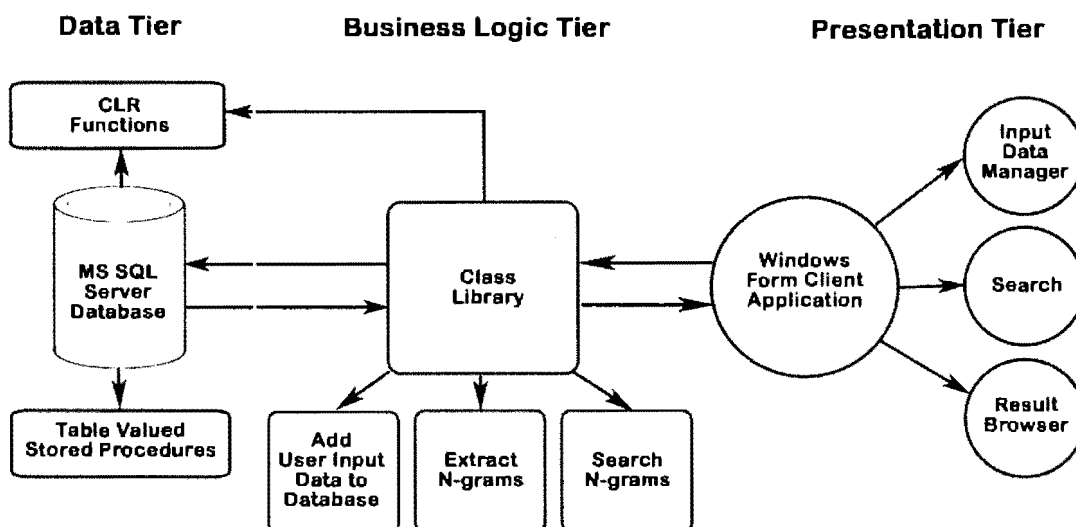


Figure 1. Tier interaction diagram.

1.4.2. Data Tier

The database utilized by the tool is hosted by Microsoft SQL Server and first populated with the user-defined input data and then with indexed word n-grams and their frequencies, as well as an English dictionary and a list of English stop words. Tables holding data defined by the tool user are synchronized using both T-SQL stored procedures and CLR table-valued functions. The database content is searched using a set of T-SQL stored procedures returning tabular n-gram results to the user.

1.4.3. Business Logic Tier

Most of the search and result filtering process is done on the database server (in the data tier) to minimize the number of data records that are being sent to the client side. This approach also makes the tool more flexible by allowing changes to be made to the database structure and database-located T-SQL stored procedures without requiring the user to make changes in the high-level language compiled code. With stored procedures, these sorts of changes can be hidden to an extent, because the parameter list and the table-valued results returned by stored procedures represent their contract, and changes can be made as long as the contract is met.

The business logic tier of our tool takes the main responsibility for sending database queries, collecting query results and processing them in a way that can be presented to the end-user in form of lists, text, numbers, and graphs.

1.4.4. Presentation Tier – Graphical User Interface

The core functionality of the system is accessible through a Windows Form-based C# .NET application. The tool's presentation tier consists of the following three low cohesive modules:

- Input data manager
- Search module
- Result browser

The user is allowed to use any set of abstracts using the input data manager module. The two restrictions put on the input file are that it must be provided to the program in XML format and include `<AbstractText>` and `</AbstractText>` XML tags in order to recognize the target data. Once, the input file is read, all bi-, tri-, and quad-grams and their respective frequencies are extracted and indexed quickly.

Once, the input data is indexed, the search module becomes active and the user is allowed to specify search parameters and perform searches by entering as little as one character of the first n-gram. The user can choose the parameter “n” for the n-grams to be two, three, or four.

The search results browser module becomes available to the user, once the n-gram search is completed. The module has three major functions:

- To allow for browsing n-grams found during the n-gram search phase: The user can see all matching n-grams as well as their corresponding frequencies in the input data.
- To view the histogram of the n-gram data: Histograms are constructed to plot data density [11]. It shows as bars, what proportion of the n-gram search result fall in each of several categories corresponding to the number of matching n-grams.
- To allow the user to select an n-gram and find out how it is used in the full text input data. The program provides a means to the user to explore the selected n-gram in a wider, more understandable context.

1.5. Organization of the Paper

The remainder of the paper is as follows: Chapter 2 provides a literature review of the relevant existing tools and techniques used in the information retrieval area. Chapter 3 explains details of a proposed algorithm and an architectural design of our search tool. Chapter 4 gives an overview of implementation of the developed application and explains its various features. Chapter 5 shows general experimentation setup and results of data indexing and search performed on the test data. Chapter 6 summarizes the work done, and provides conclusions and suggestions for future work.

CHAPTER 2. BACKGROUND

2.1. Existing N-gram-based Search Tools

There are many search tools that utilize n-grams to improve users' search experience. Google Inc. [12] uses n-grams in a variety of its research and development projects, including data mining of large data sets. Google's n-gram database consists of a huge amount of word n-grams collected from the web. Google Search however, uses algorithms that are not based solely on n-grams to provide the most relevant search term suggestions to their users. Rather than n-grams, Google's Search engine implements the content-based data classification techniques [13]. Google Search displays the first ten (most popular) search suggestions and doesn't allow the user to view the frequency count of the suggested word n-grams.

Sekine [14] proposed a tool for linguistic knowledge discovery, based on n-grams with "n" parameter equal to nine. The tool could work with an arbitrary number of wildcards. It also had low hardware requirements and provided the search results in a fraction of a second. However, to index 120 millions of n-grams using five 4GB-memory machines, two months were needed. Increasing the amount of random access memory to 64GB reduced the processing time to one week. This fact, virtually limits the tool to be used with one static set, rather than dynamically changing sets of data.

A system for querying n-grams produced from one trillion words that were drawn from the Internet was presented by Hawker et al. [15]. The authors created a

search tool that worked with the word n-grams with a parameter “n” in the range from one to five. To reduce the size of the n-gram dataset, the collection was filtered by a cut-off frequency of 200 units for unigrams, and 40 items for bigrams to 5-grams. However, filtering the n-gram database made the tool difficult to use, when searching for word n-grams having low frequency count value.

An approach to develop a tool based on word n-grams to improve the user’s search experience was made by Cui *et. al.* [16]. An algorithm presented by the authors aimed to extract correlation between search query terms and document text by analyzing server logs containing complete characteristic of the user browsing activity. The algorithm was based on the information on the documents that were selected by the users after reading a set of the search results for a particular set of query terms. To measure the “query – result set” correlation, word n-grams were extracted from a complete set of available documents and then compared to a set of words used as the search query. Before measuring correlation, stop words were eliminated from the searched set to improve the overall accuracy of the search results.

2.2. MEDLINE® Database

N-gram based algorithms usually work best with large sets of textual data. One of such sets is MEDLINE®. It is the authoritative repository of abstracts from literature published in medical and biomedical journals. MEDLINE currently contains over 18 million abstracts, covering a wide range of disciplines within health

sciences (broadly interpreted), from biochemistry to public health [17]. The oldest documents available at MEDLINE library were published in the late 40's of the 20th century. All abstracts in the database are publicly-accessible for reading, basic searching and downloading using PubMed – a native MEDLINE web-based search tool [18]. This tool is in most cases effective for simple searches, but does not provide an effective means of working with complex search scenarios. Figure 2 illustrates a sample interface of the PubMed basic search module, whereas Figure 3 shows a sample interface of the PubMed advanced search module.



Figure 2. A sample interface of PubMed basic search module.

The basic search module gives only a very basic functionality that allows the user to search for a keyword or a set of keywords. The query is then processed on the server side and a set of abstracts containing the searched terms is returned to the user. The interface of the advanced module of the search application provides more flexibility in specifying different search criteria. The user can search by author, journal, publication date or different categories. However, even the advanced module doesn't give much flexibility that would allow narrowing the search space or giving the user more information about search terms found in the abstract texts (Figure 3). The interface of PubMed, however, can be used to find abstracts within a subject area that is of interest to the user. The obtained set of data, once saved, can be used with other search and processing tools.

Search by Author, Journal, Publication Date, and more

Fill in any or all of the fields below, as needed.

All of these (AND) Any of these (OR)

Author

Journal

Publication Date to present
 (yyyy/mm/dd - month and day are optional)

[Click here](#) Add More Citation Search Fields

Limit by Topics, Languages, and Journal Groups

Full Text, Free Full Text, and Abstracts

Links to full text Links to free full text Abstracts

Humans or Animals

- Humans
 Animals

Gender

- Male
 Female

Type of Article

- Clinical Trial
 Editorial
 Letter
 Meta-Analysis
 Practice Guideline

Languages

- English
 French
 German
 Italian
 Japanese

Subsets

- Journal Groups**
- Core clinical journals
 Dental journals
 Nursing journals

Ages

- All Infant: birth-23 months
 All Child: 0-18 years
 All Adult: 19+ years
 Newborn: birth-1 month
 Infant: 1-23 months

Figure 3. A sample interface of PubMed advanced search module.

PubMed is a pure Boolean retrieval engine [17], which often makes it difficult to use, since it requires entering a complete search term, instead just part of it. A single misspelled letter in a search string submitted to PubMed can result in zero returned articles. It is a critical issue, specifically when users don't know the exact, correct spelling of more complex non-english terms (such as common biological terms) that frequently occur in the health science field. Figure 4 gives the sample search result representation of PubMed search engine. The interface provides information about the number of documents that contain specified search terms. However, it doesn't give the users any option to search for a set of words, based on their existence in the English language. There is also no quick way to find and browse documents based on set of words related to the used search keywords.

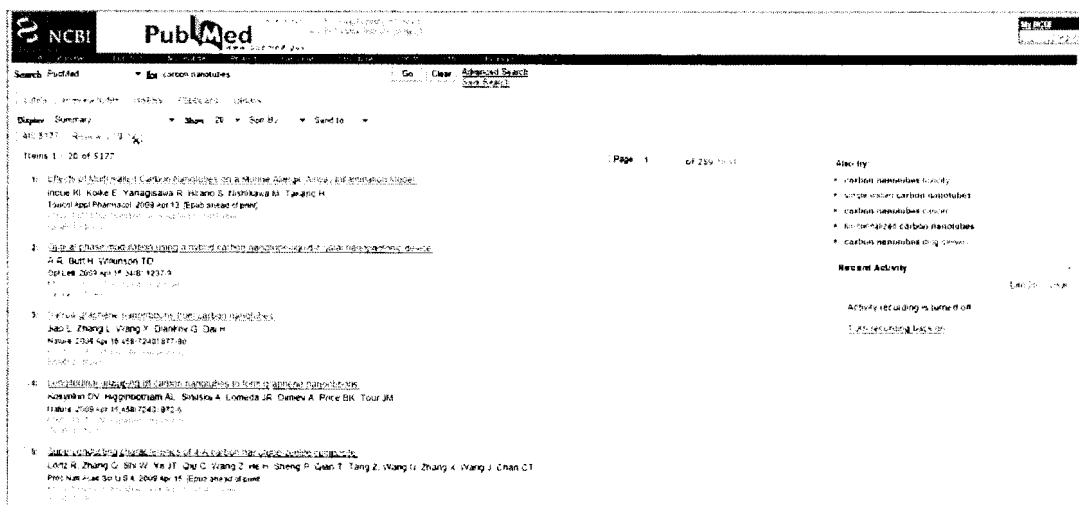


Figure 4. A sample representation of the PubMed search results.

2.3. Microsoft SQL Server

Microsoft SQL Server is a relational model database server developed at Microsoft Corporation. The most popular programming language used by this software is Transact Structured Query Language (T-SQL). The most current version of SQL Server is SQL Server 2008 (code-name: Katmai) [18]. One of the key features added to this version of the software is the Full-text index functionality that has been integrated with database engine. Full-text indexing is a powerful and fast way to reference the context of a character based column on a table that exists on the SQL Server 2008 [20]. MS SQL Server and full-text index technology were successfully utilized by Agrawal [21] in a system for a keyword-based search of large databases.

2.4. LINQ to SQL

Microsoft .NET Language Integrated Query (LINQ) is a recently introduced technology that allows a user to write queries in a uniform way in any high-level programming language. LINQ takes full advantage of strong typing that is strictly enforcing data type rules with no exceptions [22]. All data types are known at compile time. By using strong typed code, more errors can be caught at compile-time, than in the case of weak typed code, resulting in fewer run-time exceptions. Even though there are still some performance implications with LINQ, this technology has a number of advantages such as [23]:

- type-safe data access

- compile-time syntax checking
- lazy query execution
- intellisense/auto-complete
- shorter syntax
- convenient way to specify queries
- SQL-like
- allows to define anonymous types (*var* types)

2.5. Common Language Runtime Functions

The Microsoft .NET Common Language Runtime (CLR) provides a shared type system, intermediate language and dynamic execution environment for the implementation and inter-operation of multiple source languages. It consists of an intermediate language (IL), an Execution Engine (EE) which can execute IL and manages a variety of runtime services such as: storage management, debugging, security, etc. It also provides a set of shared .NET Framework libraries. There are reports of using the CLR with a variety of programming languages, including C#, Visual Basic, C++, Eiffel, Cobol, Standard ML, Mercury and Scheme and Haskell [24, 25].

The CLR has been successfully integrated inside the SQL Server Database Management System by Acheson *et al.* [26]. The authors emphasize the ability of such integration to write business logic of the application in the form of functions, stored procedures, triggers, data types, and aggregates using modern programming

languages. The presented design allowed running application code inside the database in a secure, reliable, scalable, and efficient manner. Such approach is described as extremely suitable to the design of application architectures that require business logic to execute in the data tier and to avoid the cost of shipping data to a business logic tier (middle tier) process in order to process data outside the database. The application code is deployed inside the database using assemblies. Once the assembly is saved in the database, the users can use SQL DDL statements which act as procedures or table-valued functions to expose entry points of the code contained within the assembly [26].

2.6. User-defined Functions and Cross Apply mechanism

Pushing down computations to the database tier has multiple benefits, such as reduced data transfer costs and faster data access [27, 28, 29, 30]. The most popular way to do that is by using User Defined Functions (UDFs) written as T-SQL statements [31]. The special type of UDFs are table-valued UDFs. Such functions return a table instead of a single value as in the case of stored procedures. When a RETURN command is executed, the records inserted into the variable of the function, are returned as its tabular output [32].

One of the functions introduced to T-SQL and supported by MS SQL Server 2008 is CROSS APPLY. This function can apply a table-valued function to a table (or row-set) in order to apply the table-value function to each row of the table, unites the resulting row sets, and joins the input table in an efficient manner [31].

CHAPTER 3. APPLICATION DESIGN

3.1. Overview

This chapter presents the architecture of the proposed Search tool, which can be used for searching n-gram database with various features. The tool allows also for browsing search results in full-text input data context. The tool utilizes multiple programming technologies to help achieve excellent performance and search accuracy. The three-tier architecture model is used in this project. In such architecture, the application is developed, executed, and maintained as independent modules, what allows any of the tiers to be modified or replaced independently.

3.2. Application Architecture

The database is responsible for storing input data and indexing it for efficient retrieval and processing. Our tool consists of six separate tables stored in the MS SQL 2008 database. The data in two of them are not changeable (dictionary and stop word tables), whereas the content of the remaining four tables holds the input data as well as n-grams that are extracted from these data. These tables can be changed by the user at the run-time. There are no foreign-key relations among those tables. However, the tables are dependent on each other and must be considered as a whole from the business logic perspective. The data tables used by the search tool are shown in Figure 5. A table that holds user-defined input data uses a full-text index and the remaining tables have unique values indexed as and marked as their atomic or composite primary keys (key icons in the figure).

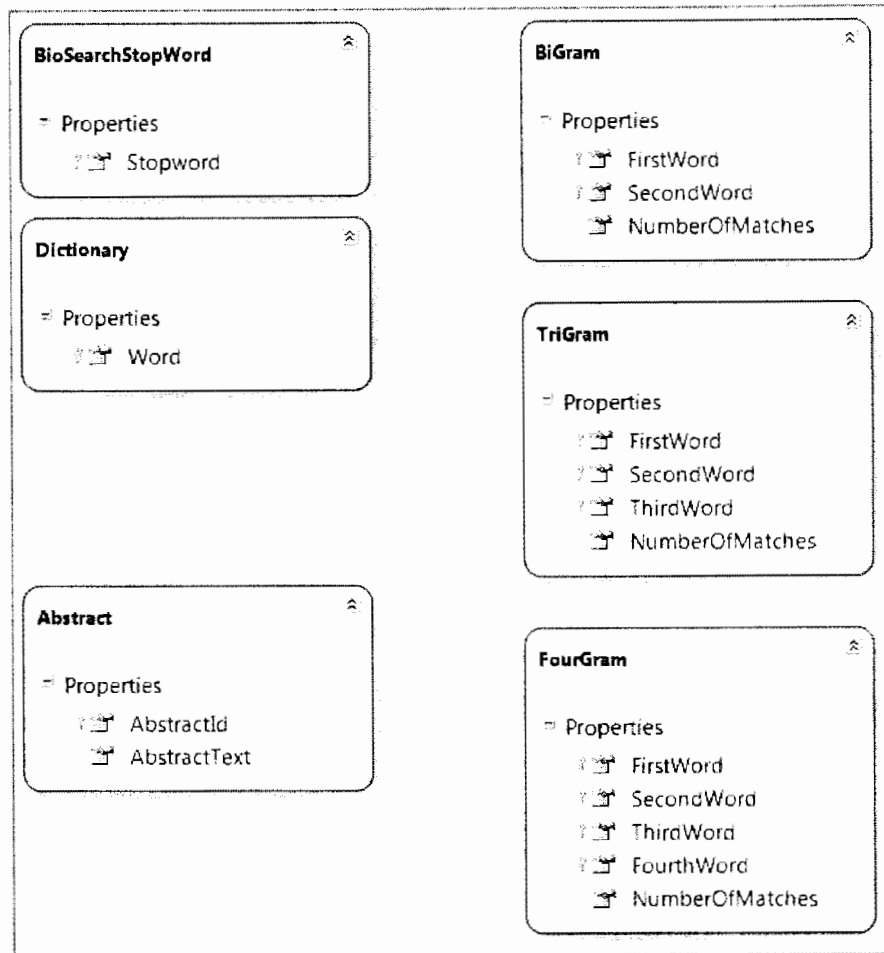


Figure 5. Search tool data model.

The user interacts with the tool using a windows form-based graphical user interface (GUI). The tool's GUI acts as a center point where the user can provide and mine data of interest. For ease of use, the data are provided and managed, processed, and presented in three separate windows, respectively. Regardless of the number of n-grams that the search is based on, the user uses the same interface for both search options and browsing the search results.

CHAPTER 4. IMPLEMENTATION

4.1. Input Data Reading

The algorithm of our tool accepts data provided in the XML format that contains one or more pairs of XML tags: `<AbstractText>` and `</AbstractText>`. These tags indicate where the data to be read begins and ends, respectively. If a file doesn't contain any tags with such name, no data will be read. A sample fragment of the XML file is shown in Appendix 1.

Once, the data file is provided, it is read by the code leveraging the .NET 3.5 Class Library class – `XmlTextReader`. This class represents a data reader that provides fast, forward-only, non-cached, access to XML data [33]. The fact the class provides forward-only access, means that every element of the XML data structure, must be read from the beginning of the file until the end (or until the desired XML element is reached). Therefore, the tool requires the reading of the entire XML file. Since the class does not provide any kind of file validation (the class assumes that the file being read is valid), try-catch exception code was written to provide exception message in case the XML is not valid. Since the class does not provide a means of reading a specific type of XML tags, each pair of tags contained in an XML document is read to determine whether the XML tag currently being read is the one that is needed (in our case, the one containing the abstract). In our application this is accomplished by creating `XmlTextReader` object and then iteratively calling (within a do-while loop) the `Read` method as long as it returns the *true* flag. At each

call to the Read method, the XML file is read, one XML element at the time. The element's name is also specified in the openXML method (Figure 6), so that the application can distinguish between various element types and read the one that is needed. The way the method is implemented allows for easy modifications to change or add additional XML tags. In such case only one additional "if" statement needs to be added to expand functionality of the application according to the user needs.

```
public string openXML(string fileName)
{
    int count = 0;
    string result = "";
    XmlTextReader reader = new XmlTextReader(fileName);

    while (reader.Read())
    {
        XmlNodeType nodeType = reader.NodeType;

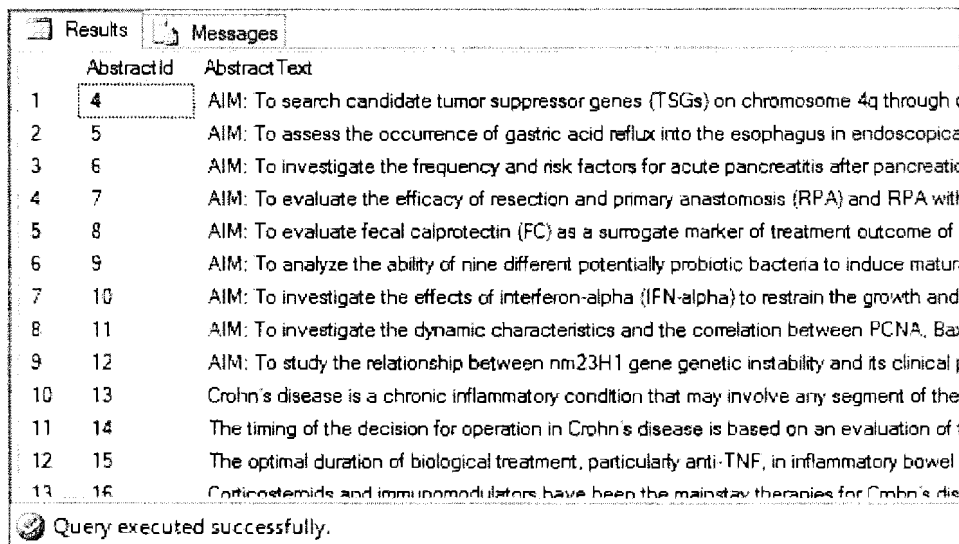
        //create sql connection and a new record
        BioSearchDataClassesDataContext dataContext = new BioSearchDataClassesDataContext();
        Abstract newAbstract = new Abstract();

        switch (nodeType)
        {
            case XmlNodeType.Element:
                if (reader.Name == "AbstractText")
                {
                    result = reader.ReadInnerXml();

                    //insert text to database
                    newAbstract.AbstractText = result;
                    dataContext.Abstracts.InsertOnSubmit(newAbstract);
                    dataContext.SubmitChanges();
                    this.counter = count++;
                }
                break;
        }
    }
    return result;
}
```

Figure 6. Code snippet for method for reading XML file.

Once each abstract is extracted from the input file, it is inserted into the Abstract table. Then, the full-text index is placed on the record for quick search of n-grams in the paper abstract text. A sample of several records of the Abstract table is shown in Figure 7.



	AbstractId	Abstract Text
1	4	AIM: To search candidate tumor suppressor genes (TSGs) on chromosome 4q through c
2	5	AIM: To assess the occurrence of gastric acid reflux into the esophagus in endoscopica
3	6	AIM: To investigate the frequency and risk factors for acute pancreatitis after pancreatic
4	7	AIM: To evaluate the efficacy of resection and primary anastomosis (RPA) and RPA with
5	8	AIM: To evaluate fecal calprotectin (FC) as a surrogate marker of treatment outcome of
6	9	AIM: To analyze the ability of nine different potentially probiotic bacteria to induce matur
7	10	AIM: To investigate the effects of interferon-alpha (IFN-alpha) to restrain the growth and
8	11	AIM: To investigate the dynamic characteristics and the correlation between PCNA, Bax
9	12	AIM: To study the relationship between nm23H1 gene genetic instability and its clinical
10	13	Crohn's disease is a chronic inflammatory condition that may involve any segment of the
11	14	The timing of the decision for operation in Crohn's disease is based on an evaluation of t
12	15	The optimal duration of biological treatment, particularly anti-TNF, in inflammatory bowel
13	16	Corticosteroids and immunomodulators have been the mainstay therapies for Crohn's dis

Query executed successfully.

Figure 7. A sample set of records in the Abstract table.

4.2. N-gram Extraction Algorithm

The n-gram extraction module of this application has the main responsibility of input data management, which has two major functions: to read abstracts in to the database and to allow the user to extract n-grams once the input data is read from the file. Implementation of the first of these two functions is described in 4.1. The second one is realized by creating an instance of the “ExtractNGrams” class and calling its method in a separate thread. That method then calls the T-SQL stored

procedure that contains code to extract n-grams. Calling the “ExtractAllNGrams” stored procedure is done with the help of LINQ to SQL class libraries. This stored procedure consists of three similar components whose responsibilities are to extract two, three, and four-grams. The stored procedure is shown in Figure 8.

```

BEGIN
--Extract all bi-grams
create table #TempAllBiGrams (FirstWord VARCHAR(50), SecondWord VARCHAR(50))

insert into #TempAllBiGrams (FirstWord, SecondWord)
select b.FirstWord, b.SecondWord from MSLAPTOP.dbo.Abstracts a
cross apply MSLAPTOP.dbo.extractBiGrams(a.AbstractText) as b

insert into MSLAPTOP.dbo.BiGrams (FirstWord, SecondWord, NumberOfMatches)
select t.FirstWord, t.SecondWord, count(*) as Matches from #TempAllBiGrams t
group by t.FirstWord, t.SecondWord
drop table #TempAllBiGrams

--Extract all tri-grams
create table #TempAllTriGrams (FirstWord VARCHAR(50), SecondWord VARCHAR(50),
ThirdWord VARCHAR(50))

insert into #TempAllTriGrams (FirstWord, SecondWord, ThirdWord)
select b.FirstWord, b.SecondWord, b.ThirdWord from MSLAPTOP.dbo.Abstracts a
cross apply MSLAPTOP.dbo.extractTriGrams(a.AbstractText) as b

insert into MSLAPTOP.dbo.TriGrams (FirstWord, SecondWord, ThirdWord, NumberOfMatches)
select t.FirstWord, t.SecondWord, t.ThirdWord, count(*) as Matches from #TempAllTriGrams t
group by t.FirstWord, t.SecondWord, t.ThirdWord
drop table #TempAllTriGrams

--Extract all quad-grams
create table #TempAllFourGrams (FirstWord VARCHAR(50), SecondWord VARCHAR(50),
ThirdWord VARCHAR(50), FourthWord VARCHAR(50))

insert into #TempAllFourGrams (FirstWord, SecondWord, ThirdWord, FourthWord)
select b.FirstWord, b.SecondWord, b.ThirdWord, b.FourthWord
from MSLAPTOP.dbo.Abstracts a
cross apply MSLAPTOP.dbo.extractFourGrams(a.AbstractText) as b

insert into MSLAPTOP.dbo.FourGrams (FirstWord, SecondWord, ThirdWord, FourthWord,
NumberOfMatches)
select t.FirstWord, t.SecondWord, t.ThirdWord, t.FourthWord, count(*) as Matches from
#TempAllFourGrams t
group by t.FirstWord, t.SecondWord, t.ThirdWord, t.FourthWord
drop table #TempAllFourGrams
END

```

Figure 8. The T-SQL stored procedure to extract n-grams.

The logic that underlies the code for indexing n-grams implemented in the stored procedure shown in Figure 8 is as follows:

1. Create a temporary table “TempAllBiGrams” to store all bi-grams,
2. Extract all bi-grams from each abstract in the database (in the abstract table) with the help of the CLR function – “extractBiGrams”,
3. Insert bi-grams extracted in step 2 into a temporary table created in step 1,
4. Group bi-grams with the same first and second words and count bi-grams in each group,
5. Insert obtained bi-grams with their frequency count into the permanent, indexed “BiGrams” table,
6. Release the memory by removing the temporary table created in step 1,
7. Repeat steps 1 to 6 for tri-grams and quad-grams, analogously.

One of the two closure properties of itemsets – the downward closure property – states that all subsets of a frequent itemset are also frequent [34]. Since the tool gives the user the ability to search for n-grams which frequency count is equal to one, implementing the downward closure property in our application wouldn't be of benefit. However, if the support threshold (the minimum n-gram frequency count) would be more than one, then the algorithm could be modified to reduce the computer memory use. In such case, the tool would first look up the bi-gram table, and omit adding tri-grams and quad-grams (to the corresponding tables), which, as their component, have bi-grams with the frequency count lower, than the minimum

support threshold. The disadvantage of implementing the downward closure property in our tool would be a need of performing multiple table scans for bi-gram, and tri-gram frequency count verifications in step 7. It could result in significantly higher CPU usage and much longer n-gram extraction processing times.

An alternative to the proposed logic that could be used to extract n-grams is as following:

1. Locate one bi-gram in the abstract text
2. Check if that bi-gram exists in the bi-gram table
3. If yes, increase the frequency count by one
4. If not, add that bi-gram to the bi-gram table with the frequency count equal to one
5. Repeat steps 1 to 4 for all n-grams located in the abstract text for all abstracts in the abstract table

However, the complexity of such algorithm would be $O(n^2)$, an equivalent to insertion sorting, which is very inefficient when used with large sets of data. Also such algorithm would require multiple I/O hard disk operations that are significantly (approximately one million times) slower than the processing times for operations done in the random access memory (RAM). Since, in the proposed tool, the n-gram table is scanned and sorted only once (at the end of the extraction phase), its complexity is reduced to $O(n \log n)$, which is an equivalent to one of much more efficient sorting algorithms such as merge sorting.

To test the scalability of the n-gram extraction algorithm, we measured the time needed to extract bi-, tri-, and quad-grams from six sets of abstract data of sizes from 10,000 to 60,000, each time increasing the set size by 10,000 abstracts. The n-gram extraction scalability test result showing the $O(n \log n)$ complexity of our algorithm is shown in Figure 9.

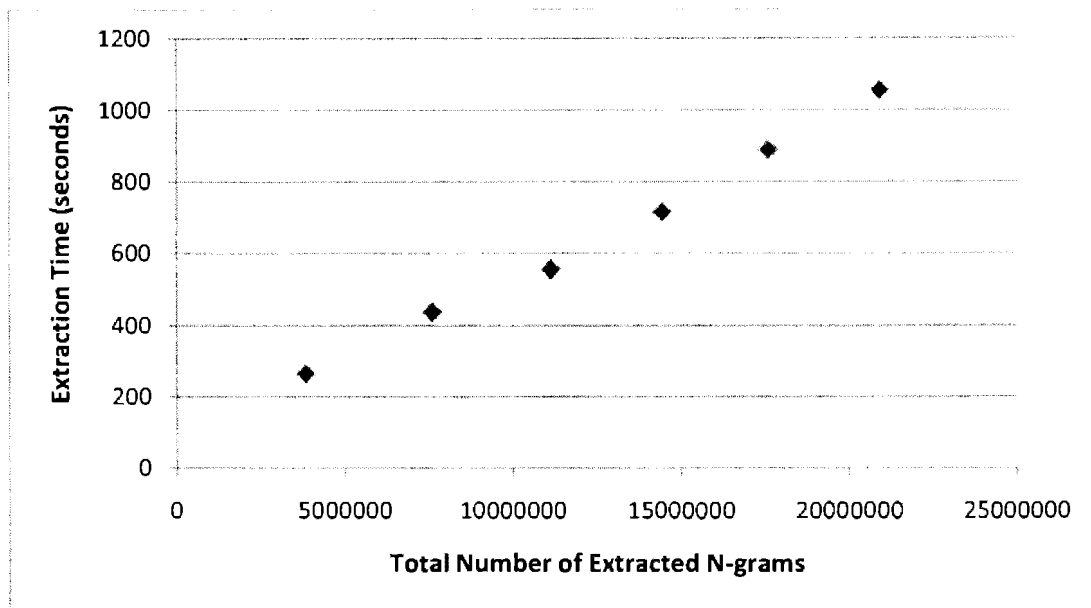


Figure 9. Scalability of the n-gram extraction algorithm.

The CLR, table-valued functions described in the n-gram extraction algorithm, namely: `extractBiGrams`, `extractTriGrams`, and `extractFourGrams` were written and compiled in the C# programming language and Visual Studio 2008 integrated development environment (IDE). Those functions were used in the efficient n-gram extraction process and allowed to leverage the rich programming model provided by the Microsoft .NET Framework.

Each of the CLR functions that were created consists of three separate parts:

- Helper class that defines an n-gram class data fields - words
- Table definition for the returned results
- The function that takes an input string as a parameter and returns a list of n-grams in the string that is provided

The code snippet for the helper class is provided in Figure 10. “BiGram” is a trivial class that defines how the extracted n-grams are presented in the n-gram list returned by the function. The properties of this class are defined using a constructor taking two arguments: `firstWord` and `secondWord` that represent the first and second word of each bi-gram respectively.

```
public class BiGram
{
    private string FirstWord = "";
    private string SecondWord = "";

    public BiGram(string firstWord, string secondWord)
    {
        this.FirstWord = firstWord;
        this.SecondWord = secondWord;
    }
}
```

Figure 10. Code snippet for the helper class “BiGram”.

The code snippet for table definition used by the CLR function to return extracted n-grams is shown in Figure 11. The code fills in one row of the n-gram table each time the function is called, which is the case when one abstract is processed.

```

public static void FillTVFRowForBiGrams(object row, out SqlString _firstWord, out SqlString
_secondWord)
{
    BiGram biGram = (BiGram)row;
    _firstWord = biGram.FirstWord;
    _secondWord = biGram.SecondWord;
}

```

Figure 11. Code snippet for the table definition used by the CLR function.

The main part of the algorithm that contains the logic used to extract n-grams from the input string is shown in Figure 12. It represents a static function that accepts a string as an argument, creates a new .NET list, creates a collection of matching n-grams for the input string, and finally loops through all n-grams making sure they are not longer than 50 characters and adding them to the created list that is returned by the function.

```

public static IEnumerable extractBiGrams(SqlString abstractText){
    System.Collections.Generic.List<BiGram> bigramList = new
    System.Collections.Generic.List<BiGram>();

    MatchCollection singleWordCollection = Regex.Matches((string)abstractText, regExPattern);

    for (int i = 0; i < singleWordCollection.Count - 1; i++){
        if (singleWordCollection[i].Length < 50 && singleWordCollection[i + 1].Length < 50){
            bigramList.Add(new BiGram(singleWordCollection[i].Value, singleWordCollection[i +
            1].Value, 1));
        }
    }

    return bigramList;
}

```

Figure 12. Code snippet for the “extractBiramS” CLR function.

The regular expression pattern used to identify word unigrams in the input string (defined as `regExpPattern` variable in Figure 12) is shown in Figure 13. This pattern identifies all words in a provided string and is used to create a collection of n-grams for a given string of words. The regular expression pattern used in this project is fully compliant with the .NET Framework [35]. That means it can be used with any .NET programming language such as C#, C++ or Visual Basic .NET.

```
(?<=\w*)\w+|\w+\.\w+(?=\s$)
```

Figure 13. Regular expression pattern used to identify word unigrams.

Once the CLR function is compiled, it is loaded as an assembly by the SQL Server. Before the function was used, the T-SQL function was written to link the external assembly and translate the result to the form that can be understood by the SQL Server. The code snippet for that function is provided in Figure 14.

```
FUNCTION [dbo].[extractBiGrams](@absText [nvarchar](max))
RETURNS TABLE (
    [FirstWord] [nvarchar](50) NULL,
    [SecondWord] [nvarchar](50) NULL)
WITH EXECUTE AS CALLER AS
EXTERNAL NAME [TsqlCrLfFunctions].[ClrFunctions].[extractBiGrams]
```

Figure 14. Code snippet for a function that links the CLR function to SQL Server.

As shown in Figure 14, the maximum size of an abstract text (`absText`) is set to the “max” value (not size limited variable) and the maximum length of each of the n-gram words is set to 50 characters. Although both the abstract text and a group of

potential n-grams are of an unknown length, the length of each n-gram was limited to utilize the available random access and HDD memory more efficiently and to make the result presentation more readable. The decision of setting threshold to 50 was made based on the length of the longest English word that exists in English dictionary [36]. The longest word in major English dictionary consists of 45 characters. Ten percent was reserved for potential special characters (such as numbers) included in n-grams.

Our program provides a means for the user to manage the input data and the n-gram extraction process using AddData window (Figure 15). The module controls three functions of the program. Using the AddData window controls, the user can:

- Add input data (Abstracts) to the program database
- Remove all input data (Abstracts and extracted n-grams) from the program database
- Start the n-gram extraction process once the input data is loaded

The window also shows the current status of the input data, including information on how many abstracts is currently in the database and how many has been added since the program was started. The program also informs the user when the process of extracting n-grams is finished.

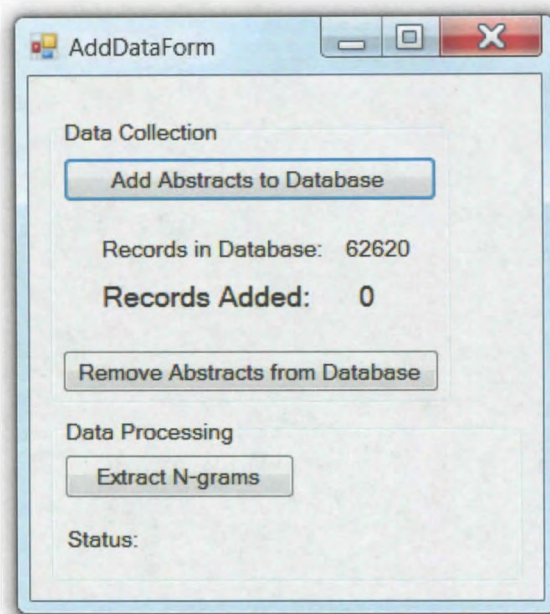


Figure 15. Input data manager window.

4.3. N-gram Search

The main function of the n-gram search module is to allow the user to specify search criteria and to start the search n-gram process. The user can interact with the application using the search form (main window of the application, i.e. the windows that shows up when the program is started). The main window of the application is shown in Figure 16.

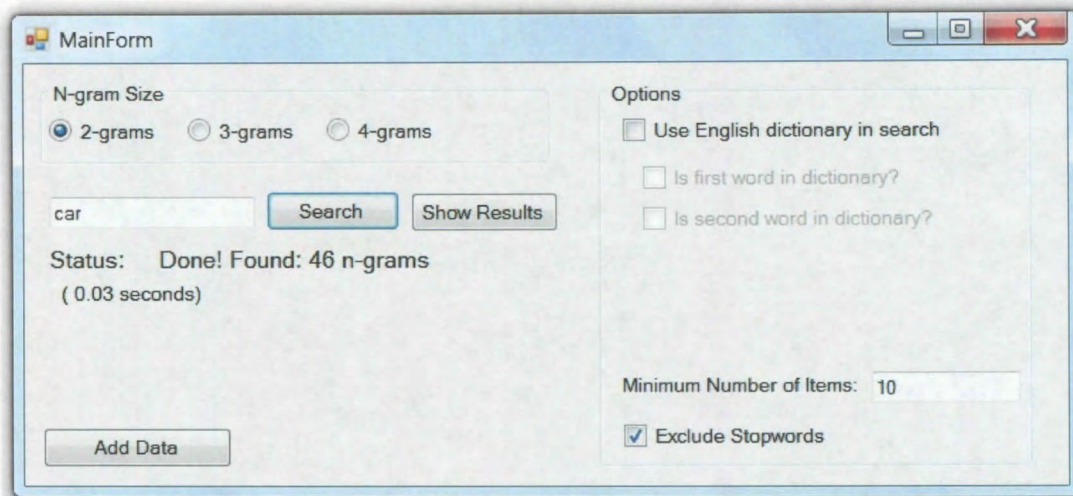


Figure 16. Search module window.

Our application requires the user to enter as little as one character of the first word n-gram as a search query. Moreover, the application allows for specifying which word in an n-gram sub-sequence should be matched against words that exist in English dictionary. The user is also allowed to set the application to ignore search results that include at least one of the English stop words [37]. We also provide a means for the user to specify the minimum frequency of n-grams that should be included in the final result set. The user has a choice to select a number of words (“n” parameter of the n-gram) on which the search will be performed. The possible word n-grams that the program accepts consist of two, three and four words.

The search process is done in a separate thread, so the user can interact with the application and observe the search process status at all times in the main window. Our tool performs the database search by calling one of the stored procedures using

the LINQ to SQL .NET Framework component. A code snippet of the LINQ query code executed when the program performs search on the bi-gram table is shown in Figure 17.

```
var items = from p in dataContext.searchBiGrams(searchWorkerHelper.startsWith,
        searchWorkerHelper.minimumNumberOfItems, searchWorkerHelper.isStopwordExcluded,
        searchWorkerHelper.isDictionaryIndependent, searchWorkerHelper.isFirstWordInDictionary,
        searchWorkerHelper.isSecondWordInDictionary)
        select p;
```

Figure 17. Code snippet of the LINQ query called for the bigram database search.

The “searchBiGrams” T-SQL stored procedure accepts six parameters that specify:

- Search string text
- Minimum frequency of n-gram that it needs to have to be included in the result set
- If the stop words should be included in the result set
- If the search process should be performed with use of the English dictionary
- If the first word in the searched bi-grams exists in the English dictionary, when search is performed with use of the English dictionary
- If second word in the searched bi-grams exists in the English dictionary, when search is performed with use of the English dictionary

Once, the required parameters are provided by the user and read by the application, our tool makes a connection with the database-located stored procedure and perform the search. Then, the result table containing searched n-grams is returned to the client and available to view by the user in the result window of the application.

4.4. Result Analysis

Figure 18 shows the window of our application where the search process results are displayed. There are three main functions of this module. First of all, it allows for browsing n-grams found during the n-gram search phase. The user can see all matching n-grams as well as their corresponding frequencies in the input data. All n-grams are sorted by the n-gram frequency in a descending order. Secondly, the user is allowed to view the histogram of the n-gram data. The histogram shows the number of n-grams that occur correspondingly often. The heights of the bars on the histogram correspond to the number of matching n-grams. If the distribution of the histogram is skewed to the right (positive skew [38]), then there is relatively low number of n-grams with high frequency count values. Finally, the user can select a word n-gram and find all its references in the full text input data. This way, the user can explore the selected n-gram in a wider, more understandable context of the original input data.

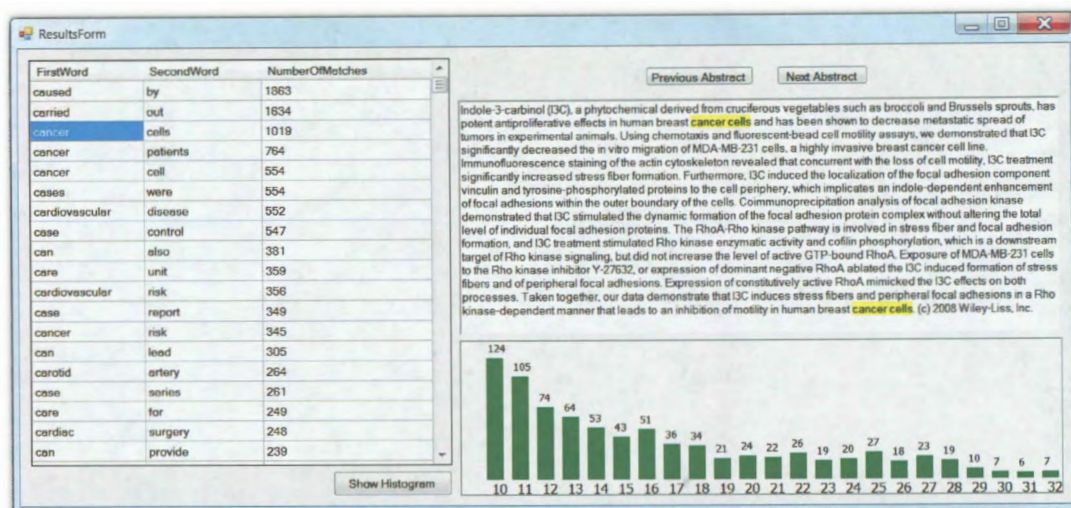


Figure 18. Result browser module window.

4.5. Full-Text Index Subtleness

Microsoft full-text search technology allows the user of our search tool to find n-grams from the n-gram search result list and browse the abstracts to see the n-grams in the full context almost instantly (in a fraction of the second) – much faster than it would take to scan search large database without using this feature. Full-text search, however, has one minor drawback in this application, namely, it treats, by default, a “.” (period character) in a special way. When, a sentence ends with “.” and a next sentence starts with a capital letter, full-text search interprets the period as an end of the sentence delimiter and does not index the two words with the “.” between them together. For this reason full-text search technology cannot be used to search n-grams being part of two different sentences. It is not the issue for all other non-alphanumeric characters as full-text search places a marker in the index and treats

those characters the same way as a whitespace character. The described issue is minor in our application since there is no (or very rarely) a need for finding word n-grams that do not belong to the same sentence.

CHAPTER 5. APPLICATION PERFORMANCE ANALYSIS

5.1. Search Effectiveness Analysis

To demonstrate the search effectiveness of the tool, several search tasks were evaluated. We tested the tool for the quality of the search results with different set of options enabled. In the presented scenario, the word “nucleoside” was used as the search query for the tests. Nucleosides are chemical structures that can be bound to the phosphate group to create new structures such as: nucleosides monophosphates, nucleosides diphosphates, or nucleosides triphosphates [39]. Using our search tool, the user can find out how often those forms occur in a specific, user-defined set of publication abstracts. Table 1 presents a list of tasks and the search results corresponding to each of them. In the first task, neither dictionary-based nor stop word-based filtering was used. It resulted in returning a relatively large set of records (84 bi-grams) that were mostly irrelevant to our search goal. In the next situation (Task 2), we used the dictionary filtering mechanism to return all matching records that for both words of the bigram exist in the English language. It allowed for reducing the number of results to 52, but filtered out the records, that we hoped to be returned in our sample search scenario. In Task 3, the dictionary filtering option was also enabled. In this case however, the option allowing for search for records with only the first word existing in the English language was selected. As the result, eight bigrams were returned. As shown in the table, the first several results were relevant to our search goal. Finally, we used stop words filtering option (Task 4), what resulted in returning by the tool intended bigrams, with their corresponding

frequency counts, as the first three positions. We also used the same parameters as in Task 4, for the tri-gram search. As a result, the search tool returned more detailed, content specific information.

Task No.	Dictionary Options	Exclude Stop words?	No. of Results	Sample Set of Returned Search Results		
1.	Not used	No	84	nucleoside	reverse	18
				nucleoside	analogues	14
				nucleoside	transporter	12
				nucleoside	analogue	12
				nucleoside	analog	10
				nucleoside	and	7
				nucleoside	transporters	6
				nucleoside	diphosphate	6
				nucleoside	naive	6
			
2.	Used – all words are in dictionary	No	52	nucleoside	reverse	18
				nucleoside	analogues	14
				nucleoside	analogue	12
				nucleoside	transporter	12
				nucleoside	analog	10
			
3.	Used – first word is in dictionary, second is not	No	8	nucleoside	triphosphates	4
				nucleoside	diphosphates	3
				nucleoside	5	2
				nucleoside	monophosphates	2
				nucleoside	NNRTI	1
				nucleoside	phosphotriesters	1
				nucleoside	triphosphatase	1
				nucleoside	imidazoquinolinamines	1
4.	Used – first word is in dictionary, second is not	Yes	7	nucleoside	triphosphates	4
				nucleoside	diphosphates	3
				nucleoside	monophosphates	2
				nucleoside	NNRTI	1
				nucleoside	phosphotriesters	1
				nucleoside	triphosphatase	1
				nucleoside	imidazoquinolinamines	1
5.	Used – first word is in dictionary, second and third are not	Yes	2	nucleoside	triphosphates	LNA 1
				nucleoside	triphosphatase	NTPase 1

Table 1. Search Tool Effectiveness Analysis

5.2. Efficiency Analysis - Overview

The efficiency analysis of the search tool can be measured by the time needed to process the specific amount of data. The application's data is processed in a different way by all three major components of our tool. First, the input file is read and the database is populated with the publication abstracts as well as the n-grams are extracted. Secondly, in the main – search component of our tool – the n-gram database is searched and results are provided to the user. And finally, the user browses the result n-grams and uses the full-text n-gram search functionality.

Scalability, a desirable property of a system or process, indicates its ability to handle growing amounts of data in a graceful manner [40]. Although the time needed to do a search on small amounts of data when utilizing the processing power of modern computers is (most of the time) extremely short, when programs are fed tens or hundreds of megabytes of data to process, programs need to be very designed to work efficiently. In the ideal case, an application should work fast with small or large amounts of data. In this chapter, we investigate the scalability of each component of our system by comparing the times needed to accomplished specific tasks with increasing amount of data provided to the system.

5.2.1. Database Population and N-gram Extraction

The scalability measure of the n-gram extraction component can be viewed as the time needed to load the abstract texts from the input file provided in the XML format to the Microsoft SQL Server database as well as the time needed to extract all word

n-grams from the database. For the testing purpose, we loaded three different input files containing abstracts downloaded from the PubMed website. The abstract files contain a collection of all abstracts published on the PubMed website in the months of December 2008, January 2009, and February 2009. Properties of the input data are summarized in Table 2.

Set	Date of Publish	Size of raw file (MB)	Number of Abstracts	Number of Bi-grams	Number of Tri-grams	Number of Quad-grams
Set 1	December 2008	443	62 620	2 992 102	7 956 064	10 904 580
Set 2	January 2009	390	60 568	1 838 539	6 154 718	12 373 728
Set 3	February 2009	506	64 347	3 038 073	8 105 305	11 124 560

Table 2. Properties of the input files

To test how the program responds to the increasing amount of data, the tests were conducted using Set 1 (62,620 abstracts) and then using cumulative data from Set 1 and Set 2 (123,188 abstracts), and finally using the dataset containing all three subsets of data, that is Set 1, Set 2, and Set 3 (187535 abstracts). The amount of time needed to perform the tasks specific to the Input Data Manager component are presented in Table 3. The table shows both the actual time spent on the task and the average amount of time spent on processing one thousand abstracts.

Task No.	Task Description	Time needed to complete the task (seconds)					
		Set 1		Set 1 + Set 2		Set 1 + Set 2 + Set 3	
		Actual	Per 1000 Abstracts	Actual	Per 1000 Abstracts	Actual	Per 1000 Abstracts
1.	Add abstracts to database	1598	25.5	3472	28.18	6121	32.6
2.	Extract n-grams	1542	24.6	2842	23.1	44.81	23.9

Table 3. Scalability test results of the n-gram extraction component

The diagram presenting the time that the program needed to complete task 1 performed with each experimental set is shown in Figure 19. It can be observed that as more data is added, scaling of the system was nearly linear – approximately the same amount of time was needed to add each thousand abstract to the database. The small decrease in speed is caused most probably by the growing size of the full-text index with the increase of data present in the Abstract table.

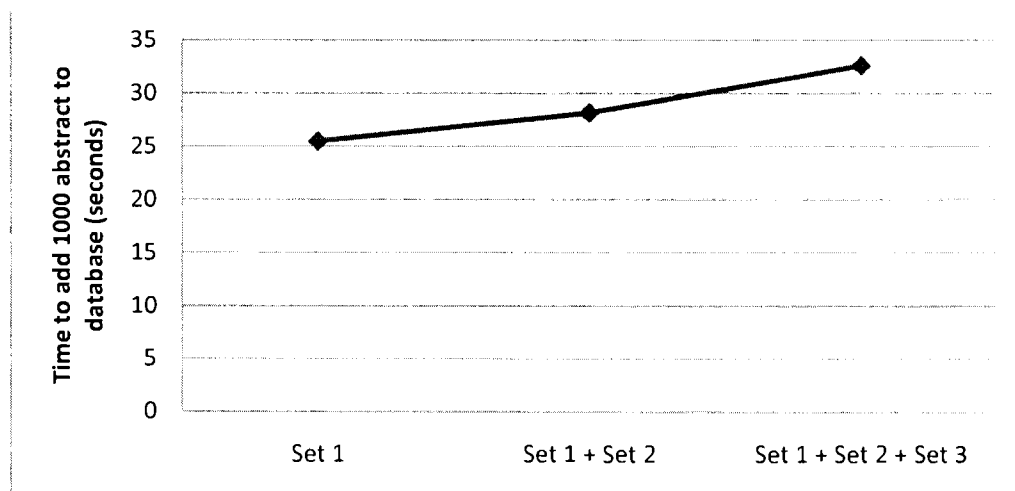


Figure 19. Task 1. Add abstracts to database.

Figure 20 shows that once the abstracts are added, it takes nearly the same amount of time to extract the word n-grams. It means that the system scales linearly. Linear scalability, relative to load, means that with fixed resources, efficiency of the system decreases at a constant rate [41].

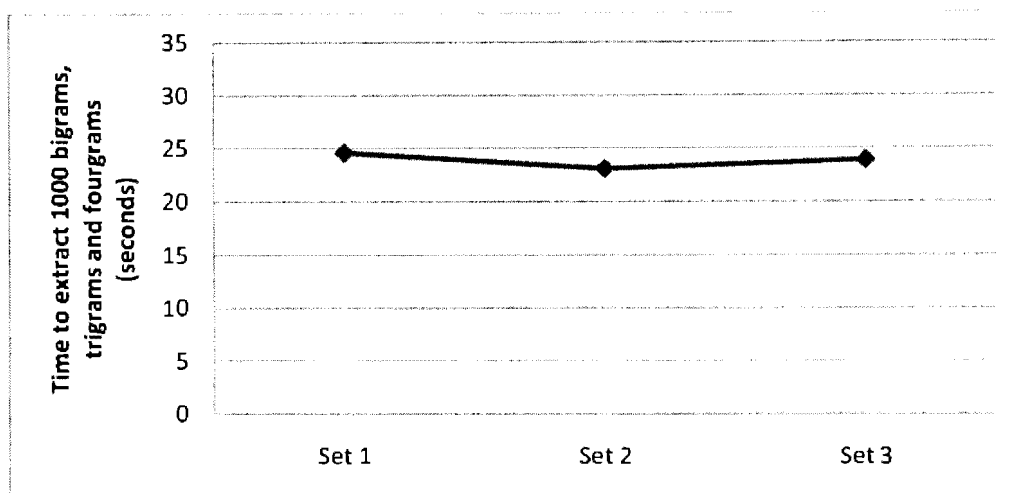


Figure 20. Task 2. Extract n-grams.

5.2.2. Search

The level of efficiency of the search module of our tool is the one that will have the biggest impact on the satisfaction level of the search tool user. While it is not likely, that the user will be adding new abstracts to the program database very often, the search module is designed to be utilized frequently to direct search queries to the system. Thus, high efficiency of this module will be of a critical importance.

The efficiency level of the search module of our system was tested with different combinations of features supported by the tool. Table 4 presents six test cases with respective times needed for completing the test tasks when searching bi-grams. The string “car” was used as the input string for all tests of the search and result browser modules as it consists of the first three letters of “carbon nanotubes” – a term that

occurs relatively often in biomedical publications. The minimum number of search results to display was set to one for all search scenarios.

Task No.	# of words	Dictionary options	Exclude Stop words?	Time needed to complete the task in seconds and number of the results returned					
				Set 1		Set 1 + Set 2		Set 1 + Set 2 + Set 3	
				Actual	Per 1000 Abstracts	Actual	Per 1000 Abstracts	Actual	Per 1000 Abstracts
1.	2	Not used	No	0.07 (9328)	0.001	0.11 (15243)	0.0009	0.16 (20010)	0.0008
2.	2	Not used	Yes	0.08 (8006)	0.001	0.12 (13255)	0.001	0.17 (17537)	0.0009
3.	2	Used - all words are in dictionary	No	0.38 (6593)	0.006	0.50 (10425)	0.004	0.57 (13382)	0.003
4.	2	Used - all words are not in dictionary	No	0.12 (301)	0.001	0.18 (544)	0.001	0.23 (772)	0.001
5.	2	Used - all words are in dictionary	Yes	0.40 (5753)	0.006	0.58 (9266)	0.004	0.62 (11993)	0.003
6.	2	Used - all words are not in dictionary	Yes	0.12 (260)	0.001	0.18 (455)	0.001	0.25 (656)	0.001

Table 4. Scalability test results of the search component for bi-grams

Figure 21 illustrates the test results presented in Table 4. It is clear, that our tool performs much better in tasks 1, 2, 4 and 6, that is, when the program doesn't search for words that exist in the English dictionary. The more abstracts provided, the shorter is the time to process n-gram data extracted from each abstract. This property helps to deal efficiently with large sets of the input data by storing each n-gram and its frequency only once.

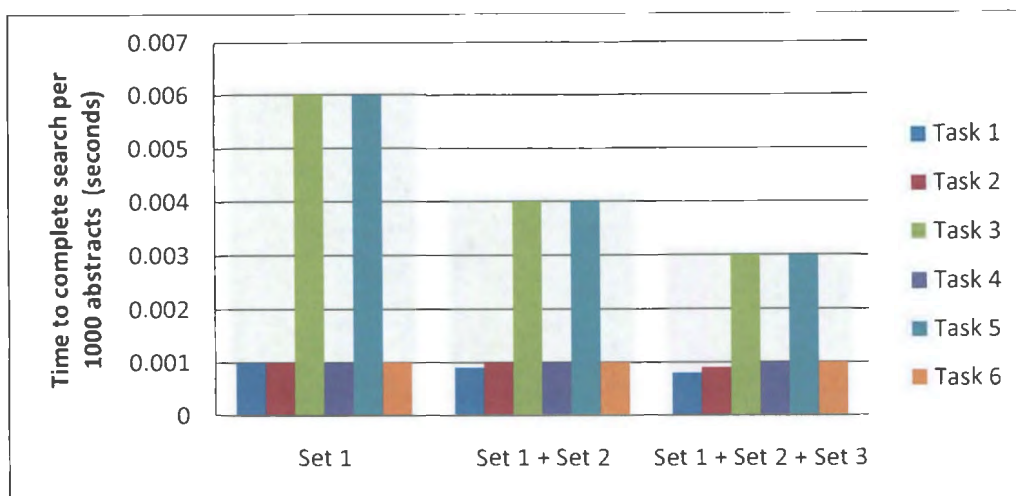


Figure 21. Efficiency test results – searching bi-grams.

The test results performed with our tool for searching tri-grams are shown in Table 5. The combinations of other parameters used in this test were the same as when testing the tool with bi-grams.

Task No.	# of words	Dictionary options	Exclude Stop words?	Time needed to complete the task in seconds and number of the results returned					
				Set 1		Set 1 + Set 2		Set 1 + Set 2 + Set 3	
				Actual	Per 1000 Abstracts	Actual	Per 1000 Abstracts	Actual	Per 1000 Abstracts
1.	3	Not used	No	7.04 (22043)	0.11	15.84 (39797)	0.12	20.23 (55536)	0.11
2.	3	Not used	Yes	3.86 (12204)	0.07	14.58 (22347)	0.11	18.67 (31469)	0.09
3.	3	Used - all words are in dictionary	No	7.81 (16457)	0.12	14.92 (29244)	0.12	20.79 (40317)	0.11
4.	3	Used – all words are not in dictionary	No	6.46 (111)	0.10	7.62 (216)	0.06	8.52 (307)	0.04
5.	3	Used - all words are in dictionary	Yes	9.48 (9116)	0.15	14.24 (16475)	0.12	8.97 (23017)	0.05
6.	3	Used – all words are not in dictionary	Yes	3.63 (73)	0.06	8.80 (133)	0.07	8.12 (187)	0.04

Table 5. Scalability test results of the search component for tri-grams

Figure 22 illustrates the test results presented in Table 5. Distribution of the search time, shows that the tool performed the best when the number of the returned results was the smallest. These results suggest that the time needed for the tool to load a large number of records from the indexed database to the computer random access memory is significantly longer, than in case when smaller number of records needed to be loaded. To accommodate this behavior, the user can choose to reduce the maximum number of results being returned to increase the speed of search.

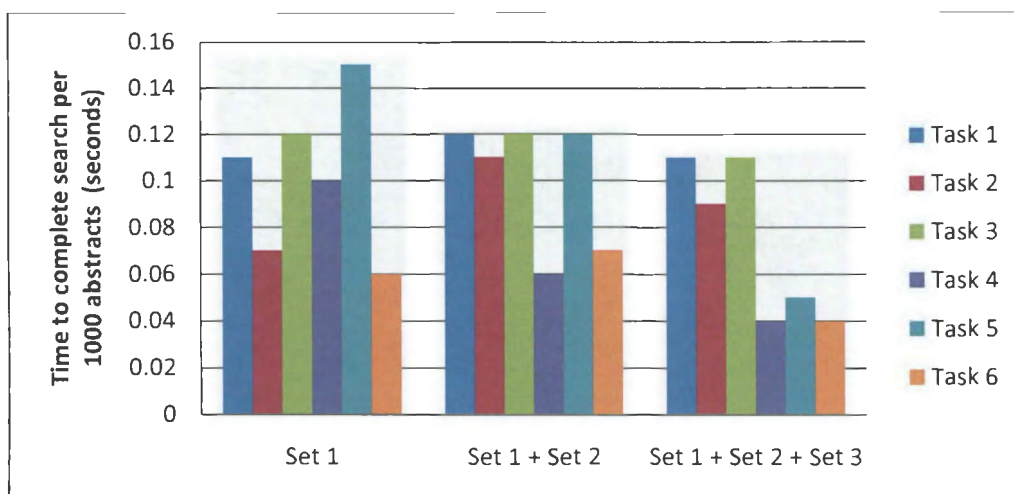


Figure 22. Efficiency test results – searching tri-grams.

The results of the efficiency tests performed with quad-grams are summarized in Table 6. The test procedures were the same as those for testing bi-grams and tri-grams. Also, in this case we tested the speed of search with optional features enabled and disabled. It can be seen, that the time needed to search quad-grams was much

longer than to search bi-gram database. This might be caused by the higher complexity of the primary key. In the case of quad-grams, the primary key is a composite key that consists of all four n-grams.

Task No.	# of words	Dictionary options	Exclude Stop words?	Time needed to complete the task in seconds and number of the results returned					
				Set 1		Set 1 + Set 2		Set 1 + Set 2 + Set 3	
				Actual	Per 1000 Abstracts	Actual	Per 1000 Abstracts	Actual	Per 1000 Abstracts
1.	4	Not used	No	2.84 (28071)	0.05	23.16 (53304)	0.18	41.84 (76356)	0.22
2.	4	Not used	Yes	2.70 (9941)	0.04	30.96 (19128)	0.25	31.60 (27537)	0.17
3.	4	Used - all words are in dictionary	No	6.55 (20136)	0.10	29.39 (37940)	0.24	40.35 (53898)	0.22
4.	4	Used - all words are not in dictionary	No	4.85 (51)	0.08	22.62 (96)	0.18	30.33 (124)	0.16
5.	4	Used - all words are in dictionary	Yes	6.26 (6924)	0.09	35.77 (13228)	0.29	36.39 (18914)	0.19
6.	4	Used - all words are not in dictionary	Yes	3.64 (27)	0.06	25.79 (48)	0.21	35.94 (60)	0.19

Table 6. Scalability test results of the search component for quad-grams

The general trend that implies high degree of our system's linear scalability is illustrated in Figure 23. Although times needed to search quad-grams are always higher than for bi-grams, the average time needed to complete the search (per 1000 records of the input data) was significantly shorter for the smallest dataset. It performed approximately equally fast with the second and third testing set. Since, the

number of n-grams stored in the index is much larger for the four-gram table, more time is needed to perform the search.

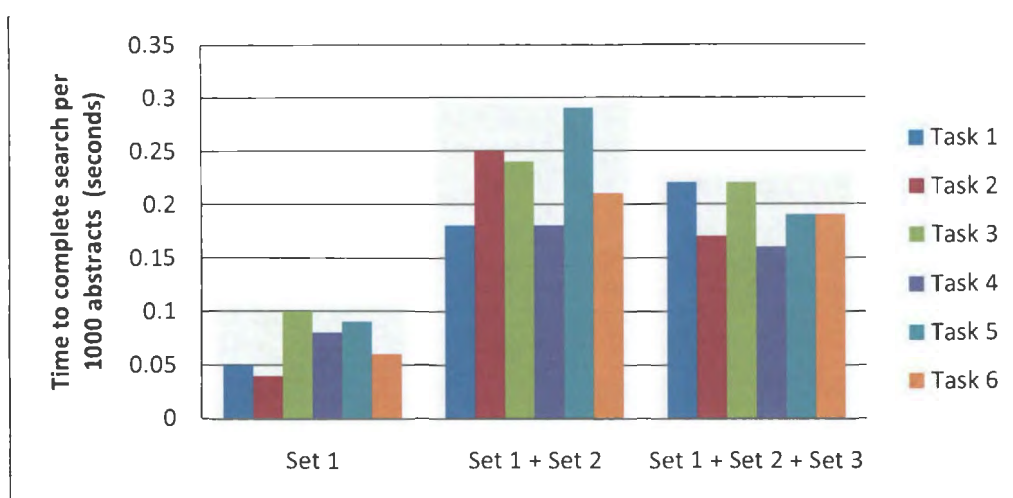


Figure 23. Efficiency test results – searching quad-grams.

5.2.3. Browsing Results

The user of our tool has the ability to browse n-grams selected in the search process and browse the full text abstracts for the selected n-gram. The abstract database contains often hundreds of thousands of records. Thus, the only option to search and retrieve the abstract quickly is by utilization of full-text index implemented in our tool with the help of a user-defined stored procedure. The scalability test results of full-text search algorithm used in our tool is presented in Table 7.

Time needed to complete the task (in seconds)		
Set 1	Set 1 + Set 2	Set 1 + Set 2 + Set 3
0.14	0.36	0.62

Table 7. Scalability test results of the result browser component

Figure 24 illustrates the results summarized in Table 7. It indicates how the number of records in the database influenced the efficiency of the result browser component. The results indicate that the time needed to find all matching n-grams in full text abstract increases with the increase in the amount of data provided by the user. The time growth has a linear nature with the growth of the size of data. If the input data set is doubled, it takes twice as long to find the reference of all matching n-grams using search and the full-text index.

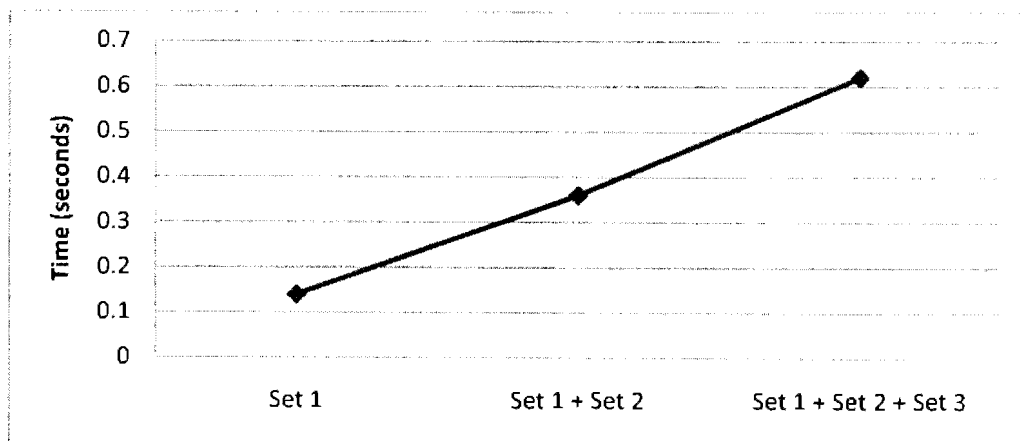


Figure 24. Efficiency test results – browsing results.

CHAPTER 6. CONCLUSION

6.1. Conclusion

In the age of rapid increase in the amount of data stored in the electronic format, efficient methods of information retrieval used by the users of search tools has become a major challenge for software developers. This paper presents an algorithm and implementation of a scalable windows-based application that aids its users in searches performed on large data sets using n-grams.

Our contribution is an application that allows its users to first: efficiently extract bi-, tri-, and quad-grams from the XML file and then to search those n-grams in an efficient manner with multiple options (such as dictionary comparison) available. We also provide a means for the user to view the search results and browse full-text sources of records where found n-grams occur. Our application can be used with any MEDLINE search results as well as with other databases with no or little modification.

The presented application proves how the utilization of new technologies and well designed architecture can significantly help to achieve high speed of data processing and better quality of retrieved information. The performance of this application has been demonstrated and analyzed through a variety of examples involving data sets of different sizes and contents.

6.2. Future Work and Limitations

There is always room for improvement in computer programs. From the perspective of the software functionality, more data mining techniques can be incorporated into the application. Some examples are the prediction of the date when the input text was published or finding the text author by finding similar texts using n-gram-based search techniques.

The next version of the tool can also give users the option of saving the search results for further processing. Once the results are saved, it should be possible to load them from the file. It could help the user to work with different input datasets and then keep the search results so that there is no need to extract n-grams each time.

REFERENCES

- [1] K.G. Coffman and A.M. Odlyzko, "Internet growth: Is there a Moore's Law for data traffic?", In Handbook of Massive Data Sets, Springer, pp. 47-93, 2002
- [2] M. Sarr, "Improving Precision and Recall Using a Spellchecker in a Search Engine. Numerish analys och datalogi", Department of Numerical Analysis and Computer Science. Royal Institute of Technology. SE-100 44 Stockholm, Sweden, 2004
- [3] "N-gram", <http://en.wikipedia.org/wiki/N-gram>, n.d., last accessed: June 14, 2009
- [4] R. Poli and N.F. McPhee, "A Linear Estimation-of-Distribution GP System", In Genetic Programming: 11th European Conference, 2008
- [5] "Information Retrieval", http://en.wikipedia.org/wiki/Information_retrieval, n.d., last accessed: May 31, 2009
- [6] W. B. Cavner and J. M. Trenkle, "N-Gram-Based Text Categorization", In Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval, 1994
- [7] M. Agyemang, K. Barker, and R. S. Alhaji, "Mining Web Content Outliers Using Structure Oriented Weighting Techniques and N-grams", Proceedings of the ACM symposium on Applied computing, pp. 482-487, 2005
- [8] J. Brophy and D. Bawden, "Is Google enough? Comparison of an Internet search engine with academic library resources", Aslib Proceedings, Vol. 57, Issue 6, pp. 498-512, 2005
- [9] T. Strohmman, D. Matzler, H. Turtle, W. Croft, "Indri: A language-model based search engine for complex queries, Proceedings of the International conference on Intelligence Analysis, 2004
- [10] "Multitier architecture", http://en.wikipedia.org/wiki/Multitier_architecture, n.d., last accessed: May 31, 2009
- [11] "Histogram", <http://en.wikipedia.org/wiki/Histogram>, n.d., last accessed: May 31, 2009
- [12] "Google", <http://www.google.com>, n.d., last accessed: May 31, 2009

- [13] “Understanding Google New Algorithm”, <http://www.nicolasprudhon.com/google-seo/new-google-algorithm>, n.d., last accessed: May 31, 2009
- [14] S. Sekine, “A linguistic Knowledge Discovery Tool: Very Large Ngram Database Search with Arbitrary Wildcards”, *Coling: Companion volume – Posters and Demonstrations*, pp. 181-184, Manchester, August 2008
- [15] T. Hawker, M. Gardiner and A. Bennetts, “Practical Queries of a Massive n-gram Database”, *Proceedings of the Australasian Language Technology*, pp. 40-48, 2007
- [16] H. Cui, J. Wen, J. Nie, and W. Ma, „Query Expansion by Mining User Logs”, *IEEE Transactions on Knowledge and Data Engineering*, Vol 15. No 4., pp. 829-839, 2003
- [17] J. Lin and M. D. Smucker, “How Do Users Find Things with PubMed? Towards Automatic Utility Evaluation with User Simulations”, *Proceedings of the 31th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 08)*, 2008
- [18] PubMed, <http://www.ncbi.nlm.nih.gov/pubmed>, n.d., last accessed: May 31, 2009
- [19] Microsoft SQL Server, http://en.wikipedia.org/wiki/Microsoft_SQL_Server, n.d., last accessed: May 31, 2009
- [20] J. R. Hamilton, T. K. Nayak, “Microsoft SQL Server Full-text Search”, *Bulletin of the Technical Committee on Data Engineering*, Vol. 24 No. 4, pp. 9-10, 2001
- [21] S. Agrawal, S. Chaudhuri, G. Das, “DBXplorer: A System for Keyword-Based Search over Relational Databases”, *Proceedings. 18th International Conference on Data Engineering*, pp. 5-16, 2002
- [22] M. Torgersen, “Language Integrated Query: unified querying across data sources and programming languages”, *Dynamic Language Symposium. Companion to the 21th ACM SIGPLAN symposium on Object-Oriented programming systems, languages, and applications*, pp. 736-737, 2006
- [23] I. Hopkins, “Unified Theory of Data Access with LINQ”, University of Saskatchewan, Department of Computer Science SR Lab Meeting, March 6, 2008

- [24] A. Kennedy, D. Syme, “Design and Implementation of Generics for the .NET Common Language Runtime”, Proceedings of the ACM SIGPLAN 2001 conference on Programming language design and implementation, pp. 1-12, 2001
- [25] J. Hamilton, “Language integration in the common language runtime”, ACM SIGPLAN Notices, Vol. 38, Issue 2, 2003
- [26] A. Acheson, M. Bendixen, J. A. Blakeley, “Hosting the .NET Runtime in Microsoft SQL server”, International Conference on Management of Data. Proceedings of the 2004 ACM SIGMOD international conference on Management of data, pp. 860-865, 2004
- [27] Q. Chen, Y. Kambayashi, “Nested Relation Based Database Knowledge Representation”, Proceedings of ACM SIGMOD, Vol. 20(2), 1991
- [28] Q. Chen, “A Rule-based Object/Task Modeling Approach”, ACM SIGMOD, 1986
- [29] D. T. Jim Gray, M. Liu, M. A. Nieto-Santisteban, „Scientific Data Management in the Coming Decade”, SIGMOD Record 34(4), 2005
- [30] M. Hsu, Y. Xiong, “Building a Scalable Web Query System”, LNCS, Vol. 4777. Springer, Heidelberg, 2007
- [31] Q. Chen and M. Hsu, “Correlated Query Process and P2P Execution”, LNCS Vol. 5187, pp. 82-92, 2008
- [32] Table-Valued User-Defined Functions, <http://msdn.microsoft.com/en-us/library/ms191165.aspx>, n.d., last accessed: May 31, 2009
- [33] “XmlTextReader”, <http://msdn.microsoft.com/en-us/library/system.xml.xmltextreader.aspx>, n.d., last accessed: May 31, 2009
- [34] K. Sriphaew and T. Theeramunkong, “Measuring the Validity of Document Relations Discovered from Frequent Itemset Mining”, IEEE Symposium on Computational Intelligence and Data Mining, pp. 293-299, 2007
- [35] C. Campeanu, H. Li, K. Salomaa and S. Yu, “Regex and Extended Regex”, In Proceedings of the CIAA’02, Paris, France, pp. 81 – 89, July 2003
- [36] “Longest word in English”, http://en.wikipedia.org/wiki/Longest_word_in_English, n.d., last accessed: May 31, 2009

[37] C.J. Van Rijsbergen, "Information retrieval", in London: Butter-worths, 1979

[38] "Skewness", <http://en.wikipedia.org/wiki/Skewness>, n.d., last accessed: May 31, 2009

[39] "Nucleoside", <http://en.wikipedia.org/wiki/Nucleoside>, n.d., last accessed: May 31, 2009

[40] "Scalability", <http://en.wikipedia.org/wiki/Scalability>, n.d., last accessed: May 31, 2009

[41] "Linear Scalability", http://www.adobe.com/livedocs/coldfusion/5.0/Advanced_ColdFusion_Administration/overview2.htm, n.d., last accessed: May 31, 2009

APPENDIX 1. A SAMPLE FRAGMENT OF XML FILE

```

<PubmedArticle>
  <MedlineCitation Owner="NLM" Status="MEDLINE">
    <PMID>18649720</PMID>
    <DateCreated>
      <Year>2008</Year>
      <Month>07</Month>
      <Day>24</Day>
    </DateCreated>
    <DateCompleted>
      <Year>2008</Year>
      <Month>09</Month>
      <Day>12</Day>
    </DateCompleted>
    <Article PubModel="Print">
      <Journal>
        <ISSN IssnType="Print">0029-6570</ISSN>
        <JournalIssue CitedMedium="Print">
          <Volume>22</Volume>
          <Issue>42</Issue>
          <PubDate>
            <MedlineDate>2008 Jun 25-Jul 1</MedlineDate>
          </PubDate>
        </JournalIssue>
        <Title>Nursing standard (Royal College of Nursing (Great Britain) :
1987)</Title>
      </Journal>
      <ArticleTitle>A holistic approach to caring for people with Alzheimer's
disease.</ArticleTitle>
      <Pageation>
        <MedlinePgn>50-6; quiz 58</MedlinePgn>
      </Pageation>
      <Abstract>
        <AbstractText>This article adopts a holistic view of Alzheimer's disease.
Biomedical, psychological and social aspects of the condition are discussed, and
aetiology, epidemiology, diagnosis and treatment explored. A range of approaches to
working with people with Alzheimer's disease, based on a psychological model of
dementia, is described including reminiscence and cognitive stimulation
therapy.</AbstractText>
      </Abstract>
      <Affiliation>University of Stirling, Stirling.
l.f.m.mccabe@stir.ac.uk</Affiliation>
    </Article PubModel="Print">
  </MedlineCitation Owner="NLM" Status="MEDLINE">
</PubmedArticle>

```

```

<AuthorList CompleteYN="Y">
  <Author ValidYN="Y">
    <LastName>McCabe</LastName>
    <ForeName>Louise</ForeName>
    <Initials>L</Initials>
  </Author>
</AuthorList>
<Language>eng</Language>
<PublicationTypeList>
  <PublicationType>Journal Article</PublicationType>
  <PublicationType>Review</PublicationType>
</PublicationTypeList>
</Article>
<MedlineJournalInfo>
  <Country>England</Country>
  <MedlineTA>Nurs Stand</MedlineTA>
  <NlmUniqueID>9012906</NlmUniqueID>
</MedlineJournalInfo>
<CitationSubset>N</CitationSubset>
<MeshHeadingList>
  <MeshHeading>
    <DescriptorName MajorTopicYN="N">Age
Distribution</DescriptorName>
  </MeshHeading>
  <MeshHeading>
    <DescriptorName MajorTopicYN="N">Aged</DescriptorName>
  </MeshHeading>
  <MeshHeading>
    <DescriptorName MajorTopicYN="N">Aged, 80 and
over</DescriptorName>
  </MeshHeading>
  <MeshHeading>
    <DescriptorName MajorTopicYN="N">Alzheimer
Disease</DescriptorName>
    <QualifierName MajorTopicYN="Y">diagnosis</QualifierName>
    <QualifierName MajorTopicYN="N">epidemiology</QualifierName>
    <QualifierName MajorTopicYN="N">psychology</QualifierName>
    <QualifierName MajorTopicYN="Y">therapy</QualifierName>
  </MeshHeading>
  <MeshHeading>
    <DescriptorName MajorTopicYN="N">Cognitive
Therapy</DescriptorName>
  </MeshHeading>
  <MeshHeading>

```

```

    <DescriptorName
MajorTopicYN="N">Communication</DescriptorName>
  </MeshHeading>
  <MeshHeading>
    <DescriptorName MajorTopicYN="N">Disease
Progression</DescriptorName>
  </MeshHeading>
  <MeshHeading>
    <DescriptorName MajorTopicYN="Y">Empathy</DescriptorName>
  </MeshHeading>
  <MeshHeading>
    <DescriptorName MajorTopicYN="N">Female</DescriptorName>
  </MeshHeading>
  <MeshHeading>
    <DescriptorName MajorTopicYN="Y">Holistic
Health</DescriptorName>
  </MeshHeading>
  <MeshHeading>
    <DescriptorName MajorTopicYN="N">Humans</DescriptorName>
  </MeshHeading>
  <MeshHeading>
    <DescriptorName MajorTopicYN="N">Male</DescriptorName>
  </MeshHeading>
  <MeshHeading>
    <DescriptorName MajorTopicYN="N">Medical History
Taking</DescriptorName>
  </MeshHeading>
  <MeshHeading>
    <DescriptorName MajorTopicYN="N">Mental Status
Schedule</DescriptorName>
  </MeshHeading>
  <MeshHeading>
    <DescriptorName MajorTopicYN="N">Models,
Nursing</DescriptorName>
  </MeshHeading>
  <MeshHeading>
    <DescriptorName MajorTopicYN="N">Models,
Psychological</DescriptorName>
  </MeshHeading>
  <MeshHeading>
    <DescriptorName MajorTopicYN="N">Narration</DescriptorName>
  </MeshHeading>
  <MeshHeading>

```

```

    <DescriptorName MajorTopicYN="N">Neuropsychological
Tests</DescriptorName>
  </MeshHeading>
  <MeshHeading>
    <DescriptorName MajorTopicYN="Y">Nurse's Role</DescriptorName>
    <QualifierName MajorTopicYN="N">psychology</QualifierName>
  </MeshHeading>
  <MeshHeading>
    <DescriptorName MajorTopicYN="N">Nurse-Patient
Relations</DescriptorName>
  </MeshHeading>
  <MeshHeading>
    <DescriptorName MajorTopicYN="N">Nursing
Assessment</DescriptorName>
  </MeshHeading>
  <MeshHeading>
    <DescriptorName MajorTopicYN="N">Patient-Centered
Care</DescriptorName>
  </MeshHeading>
  <MeshHeading>
    <DescriptorName MajorTopicYN="N">Prevalence</DescriptorName>
  </MeshHeading>
  <MeshHeading>
    <DescriptorName MajorTopicYN="N">Sex
Distribution</DescriptorName>
  </MeshHeading>
</MeshHeadingList>
<NumberOfReferences>32</NumberOfReferences>
</MedlineCitation>
<PubmedData>
  <History>
    <PubMedPubDate PubStatus="pubmed">
      <Year>2008</Year>
      <Month>7</Month>
      <Day>25</Day>
      <Hour>9</Hour>
      <Minute>0</Minute>
    </PubMedPubDate>
    <PubMedPubDate PubStatus="medline">
      <Year>2008</Year>
      <Month>9</Month>
      <Day>16</Day>
      <Hour>9</Hour>
      <Minute>0</Minute>
    </PubMedPubDate>
  </History>
</PubmedData>

```

```
</PubMedPubDate>
</History>
<PublicationStatus>ppublish</PublicationStatus>
<ArticleIdList>
  <ArticleId IdType="pubmed">18649720</ArticleId>
</ArticleIdList>
</PubMedData>
</PubMedArticle>
```